Mehmet Özer Metin

METU 2007

# CREATING APPLICATION SECURITY LAYER BASED ON RESOURCE ACCESS DECISION SERVICE

MEHMET ÖZER METİN

SEPTEMBER 2007

# CREATING APPLICATION SECURITY LAYER BASED ON RESOURCE ACCESS DECISION SERVICE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MEHMET ÖZER METİN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2007

Approval of the thesis:

CREATING APPLICATION SECURITY LAYER BASED ON RESOURCE ACCESS DECISION SERVICE

Submitted by **MEHMET ÖZER METİN** in partial fulfillment of the requirements for the degree of **Master of Computer Engineering in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Volkan Atalay _____
Department Chair, **Computer Engineering Dept., METU**

Dr. Cevat Şener _____
Supervisor, **Computer Engineering Dept., METU**

**Examining Committee Members:**

Assoc. Prof. Dr. Ali Doğru _____
**Computer Engineering Dept., METU**

Dr. Cevat Şener _____
**Computer Engineering Dept., METU**

Assoc. Prof. Dr. Ahmet Çoşar _____
**Computer Engineering Dept., METU**

Dr. Semih Çetin _____
**Management Board Member, Cybersoft**

Yenal Göğebakan, M.Sc. _____
**Management Board Member, Cybersoft**

Date: 03/09/2007

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Mehmet Özer Metin
Signature:

# ABSTRACT


CREATING APPLICATION SECURITY LAYER BASED ON RESOURCE ACCESS DECISION SERVICE

Metin, Mehmet Özer

M.S., Department of Computer Engineering

Supervisor: Instructor Dr. Cevat Şener

September 2007, 176 pages

Different solutions have been used for each security aspects (access control, application security) to secure enterprise web applications. However combining "enterprise-level" and "application-level" security aspects in one layer could give great benefits such as reusability, manageability, and scalability. In this thesis, adding a new layer to n-tier web application architectures to provide a common evaluation and enforcement environment for both enterprise-level and application level policies to bring together access controlling with application-level security. Removing discrimination between enterprise-level and application-level security policies improves manageability, reusability and scalability of whole system. Resource Access Decision (RAD) specification has been implemented and used as authentication mechanism for this layer. RAD service not only provides encapsulating domain specific factors to give access decisions but also can form a solid base to apply positive and negative security model to secure enterprise web applications. Proposed solution has been used in a real life system and test results have been presented.



Keywords: Access Control, Enterprise-level Security Policy, Web Application Security

# ÖZ

## KAYNAK ERİŞİM KONTROLU SERVİSİNİ KULLANARAK UYGULAMA GÜVENLİK KATMANI GELİŞTİRMEK

Metin, Mehmet Özer
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi Öğr. Gör. Dr. Cevat Şener

Eylül 2007, 176 sayfa

Erişim kontrolü, uygulama güvenliği gibi farklı güvenlik ihtiyaçlarının her birisi için farklı çözümler kullanılmaktadır. Fakat kurumsal seviyedeki ve uygulama seviyesindeki güvenlik ihtiyaçlarını tek bir katmanda birleştirmek, uygulamalara tekrar kullanılabilirlilik, kolay yönetilebilirlik ve ölçeklendirilebilirlik gibi önemli faydalar sağlayabilir. Bu tezde, çok katmanlı internet uygulamalarına yeni bir katman ekleyerek; kurumsal ve uygulama güvenlik politikalarının beraber yönetilip, işlenebildiği genel zorlayıcılığı olan ortak bir ortam geliştirilmiştir. Kurumsal ve uygulama güvenlik politikalarının arasındaki ayrımı kaldırarak uygulamanın genelinde tekrar kullanılabilirliğin, kolay yönetilebilirliğin ve ölçeklendirilebilirliğin artırılmasına çalışılmıştır. Bu katman için kaynak erişim kontrol (RAD) belirtimi geliştirilmiş ve erişim kontrol mekanizması olarak kullanılmıştır. RAD servisi sadece erişim kontrolünde tanım kümesine ait etmenlerin kullanılmasına izin vermekle kalmayıp, aynı zamanda internet tabanlı kurumsal uygulamalar için pozitif ve negatif güvenlik modellerini uygulamak için de sağlam bir taban oluşturabilir. Önerilen çözüm kullanıma geçmiş gerçek bir uygulamada denenmiş ve sonuçları sunulmuştur.

Anahtar Kelimeler: Erişim Kontrolü, Kurumsal Seviyeli Güvenlik Politikaları, İnternet Tabanlı Uygulama Güvenliği

To my parents who devoted themselves to their children and to my lovely sister.
Life is so beautiful, only with them.

# ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my supervisor Dr. Cevat Şener for his guidance and support throughout this research effort. I consider myself fortunate to have had a mentor with the strong work ethic and unyielding patience of Dr. Cevat Şener.

I would like to thank Egemen İmre and Evren Kapusuz for their invaluable reviews and critiques about my thesis. I would also thank to Ferhat Y. Savcı for his mentoring for technical background. I have learned a lot while working with him. I also would like to thank to Yenal Göğebakan for his foresight ideas and his wisdom. I feel myself very lucky to have worked for him.

CSAAS and EYEKS are trademarks and products of Cybersoft. Various people have been involved in developing CSAAS. I would like to praise them for their efforts. Thanks to Ceyhun and İbrahim Onur Yaranlı for developing earlier versions of CSAAS and special thanks to Evren Kapusuz and Aslı Acar for helping me to finalize the latest version of CSAAS.

My love Müge, I could not finalize this thesis without your help and motivation. Thanks for your endless support and trust.

# TABLE OF CONTENT

# LIST OF TABLES

# LIST OF FIGURES

.

# CHAPTER 1

# INTRODUCTION

## 1.1 Scope of the thesis

The Internet and World Wide Web brings about new rules about how the business conducted. It started a business revolution and a new era emerged. As business has evolved into e-business and governments became e-governments, the Internet is now forcing enterprises to implement collaborative business and governmental solutions that integrate internal systems. Many enterprises have integrated Enterprise solutions such as ERP (Enterprise Resource Plan) and CRM (Customer Relationship Management). These solutions, the so called Enterprise level software, provide business logic support functionality (such as accounting, production scheduling, customer information management, etc.) for an organization which aims to improve its productivity and efficiency.

Enterprise software is often categorized by the business function that it automates - such as accounting software or sales force automation software. E-Government is one of the examples which refer to government's use of information technology to exchange information and services with citizens, businesses, and other arms of government. E-Government may be applied by the legislature, judiciary or administration and the primary delivery models are Government-to-Citizen or Government-to-Customer (G2C), Government-to-Business (G2B) and Government-to-Government (G2G) & Government-to-Employees (G2E). The most important anticipated benefits of e-government include improved efficiency, convenience and better accessibility of public services. Health Informatics or e-Health domain can also be regarded as a good example of Enterprise applications. Health care information system builds on communication interface between various objects of health domain, starting from patients to doctors, hospital managements and finally governmental public health institutions. It also provides new point of views to traditional business models; patients to interact with their systems online (B2C = "business to consumer"); improved possibilities for institution-to-institution transmissions of data (B2B = "business to business"); new possibilities for peer-to-peer communication of consumers (C2C = "patients to patients or doctors to doctors").

Enterprise software is often designed and implemented by an Information Technology (IT) group within an organization. This in-house software may also be purchased from an independent software developer that often installs and maintains the software for their customers. Another model is based on a concept called on-demand software, or Software as a Service. Software as a service (SaaS) is a software application delivery model where a software vendor develops a web-native software application and hosts and operates the application for use by its customers over the Internet.

Because enterprise applications tend to have a broad spectrum of business requirements, starting from employee relationship to resource planning and customer management, integration and communication complexity become main concerns of enterprise applications. Middleware technologies have emerged to integrate these applications into an enterprise-wide solution, providing well-integrated, networked software infrastructure. Middleware, which is quickly becoming synonymous with enterprise applications integration (EAI), provides interoperability between different applications by placing middleware between layers of software to make the layers below and on the sides work with each other. Middleware technologies push applications out to distributed environments and unleashing the domain-specific value of each application. Consequently, this frees application developers to focus on higher-value development instead of repetitive and tedious application-communication and distribution tasks.

### 1.1.1 Enterprise Level Security

Nevertheless integration of these diverse systems also introduces a new burden to enterprise security. Each enterprise application comes with its own security rules and access policies as well as sharing business transactions over enterprise applications need a new set of enterprise security rules that must be handled organization-wide. Providing integrative security for diverse enterprise applications becomes more important than securing each of them independently. The problem of securing information enterprises has been the focus of intensive efforts from the industry. This is why it is an essential concern to every enterprise [1]. As a result, several well-known middleware systems have adapted their security model to construct scalable and flexible security for distributed environments. OMG's Corba [2], Microsoft's COM+ [3] and Sun's EJB [4] all include access control mechanism that depends on access control list (ACL). These middleware access control mechanism will be discussed in detail in section 2.5.1.

The main purpose of all of these security models is controlling object interactions with in an organization-wide, uniform and transparent way. However they all fail their expressiveness and granularity when we consider enterprise applications. ACL provides limited capabilities for handling complex policies and authorization decisions that are based on factors specific to an application domain [5] and also a single level of granularity which is object, does not support enough abstraction over enterprise policy rules. Enterprise applications consist of business transactions and business services that require much more abstraction to be controlled by object interaction access control.

The complexity of access control policies in enterprise applications comes from embedded business logic. Enterprise applications aim to map real world business rules, interactions, regulations and sometimes laws (e-government applications) to computer domain. This mapping must also be achieved for access control. As access control logic becomes closer to enterprise level, policy rules become more dynamic, more domain-specific and more contexts dependent. For example the current

state of a workflow process, the time or other contextual information may be relevant when making an access control decision. Göğebakan [6] and Metin [7] address the access control problems in enterprise applications. Implementing collaborative business and governmental solutions that integrate internal systems, introduces complex access control rules that originate from both business logic and integration of business transactions. At this point access control rules become so called “enterprise-level security policies”.

Since middleware infrastructures fail to evaluate enterprise-level security policies, most enterprise applications tackle this problem by embedding access control rules within an application code that handles domain-specific factors. The more access control rules are embedded in enterprise applications, the more reusability and manageability of whole system reduces. Beznosov has criticized this issue [5] and advised that the logic of security policy decision should be separated from an application system because all security related decisions made by an application depend not only on the application business logic but also on security policies that are enforced in the given organization and these enterprise-level security policies are subject to changed rapidly when legislation, regulations or company's businesses process changes. Besides, it is very hard for software vendors to know a priori security policies enforced across customers' enterprises.

Although will be widely discusses in section 2.3, generally speaking, current enterprise application solutions suffer from the following access control problems;

- The policy rules become too complex such that they are fine grain, domain-specific, dynamic and context sensitive to be executed in a traditional way. For example, an online banking application requires the EFT operation to be within a user-defined amount limit and to take place between 9:00 am and 4:00 pm.

- Largely embedded in application systems and as a result it becomes too difficult to manage and reuse.

- Need organization-wide enforcement because of potentially large number of heterogeneous distributed applications and users.

- Costly and error-prone because there are multiple points of control, every part of application implements their own access policies so lack of means to assure organization-wide consistency and end-to-end properties.

- Frequently subject to change due to legislation, regulations or businesses process changes of the company.

### 1.1.2 Application Level Security

Although not limited to web based, nearly all of the enterprise applications has web interface such as web services and/or web applications. Day by day more and more business is conducted via Internet while enterprises and governments offer online service. As organizations have been increasing their reliance on web applications, Attackers are turning their attention to these business applications. Although network-layer defenses have become steadily matured, traditional firewalls should not be the only protective measure in place to defend enterprise web applications. Neither other defense systems such as Network Intrusion Detection and Prevention Systems (NIDS/NIPS) can be enough to solve the problem. These solutions actively monitor traffic on the network for malicious activity. NIDS solutions are often set in passive or SPAN port mode. This means that NIDS can only send TCP resets to stop some of the bad TCP packets. A shortfall of a NIDS solution is that they can not actively block any UDP traffic. NIPS perform the same functionality as a NIDS, except that it sits actively inline with the data flow it is monitoring. This option is able to actively block any packet deemed inappropriate for that network segment. Host-Based Intrusion Detection Systems (HIDS) and Host-Based Intrusion Prevention Systems (HIPS) can also be used to protect servers. HIDS and HIPS are parasitic software that monitors respective hosts for anomalous behavior. This software can look for specific attacks directed at the server, whereas the network solutions monitor only the network traffic between them.

The reason behind incapability of network layer defenses to protect enterprise web applications is that web application attacks turns to threaten application layer instead of network layer. Application-level web security refers to vulnerabilities inherent in the code of a web-application itself. Attackers can use application's own code or business logic against itself by only tampering parameters that does not still violate network layer security policies. This makes impossible to be detected by network layer devices. Statistics collected from SANS Institute shows that [8]; from 1Q05 to 1Q06 there has been a 20% rise in the number of application-specific vulnerabilities identified and over 50% of these are based on web applications and greater than 80% of all malfunctions that emerged in the past year have focused on exploiting application-layer vulnerabilities.

The most dangerous and the most unnoticeable and therefore, the hardest to prevent type of attacks are these that exploit application layer vulnerabilities. Although these vulnerabilities have similar patterns, they are unique to the application. Web application vulnerabilities do not have to be as a result of common implementation bugs or mishandling of business rules. Consequently, there is no general catch-all solution to remove weaknesses or vulnerabilities from enterprise web application. The main reason behind web application vulnerabilities is that most of the time security is not considered as essential design concept of enterprise application development. It must be essential to build security concept into the Software Development Life Cycle by developing standards, policies and guidelines that work within the development life cycle [9] otherwise even a single inexperienced

software developer (in most cases, software developers does not have enough knowledge or experience about security) can cause serious security flaws.

This is why 95 % of highly used web applications have vulnerabilities [10]; even global leaders of IT sector suffer from serious security flaws. Netscape, Amazon, Google, MSN and MySpace have been reported to have cross site scripting vulnerabilities [11] which threatens clients of these sites and even more dangerous impacts like credit card losses can be occurred as seen in AT&T, RI Gov, TJX, Moneygram and PortTix cases [12].

There are various types of web applications vulnerabilities and attack vectors. The impacts also vary greatly. Most common impact is disclosure of information where it may be as simple as revealing the structure of web applications but may also be as dangerous as disclosure of sensible data like credit card numbers. Unauthorized access or modifications are other serious impacts that can be used to achieve various goals. Attack vectors are also vary a lot and are specific to the web application. Most of the time, a number of different attack techniques are used sequentially to maximize the success probability of the attack. The attacks, in most cases, target to reveal web application structure using directory traversal, then may continue with analyzing response headers and session management strategy and finally ends with injection types of attacks. These vulnerabilities are examined and discussed widely in section 2.4. There is also a vast amount of research to define and classify security incidents. Some researchers construct a list of terms that defines a number of attacks [13]. Some considers origin of the vulnerabilities to build the taxonomy [14, 15]; identifying the impact of vulnerability that describes the result of attack is another technique [16]. Stalling focuses on process, rather than a single classification category, in order to provide a successful classification scheme for Internet attacks [17]. There are also numerous vulnerability databases which concentrate on reporting rather than categorizing. With all these efforts of classification, there is confusion and fuzziness about the standardization of vulnerability names. Common Vulnerabilities and Exposures project tries to standardize the names for all publicly known vulnerabilities and security exposures which can be considered as a dictionary, not a database [11]. These efforts are presented in detail in section 2.1.

Some organizations and consortiums are also founded to concentrate only on web application vulnerabilities to increase public awareness about web application security and dedicated to find and classify possible web application attacks and offers countermeasures for them. Web Application Security Consortium (WASC) is one of them and releases threat classification of web application attacks [18]. Open Web Application Security Project (OWASP) is the other and publishes “Top Ten Most Critical Web Application Security Vulnerabilities” list every year to inform the public about the most dangerous web application vulnerabilities.

Although there are various types of vulnerabilities and attack vectors, the source of vulnerabilities is most of the time the same; improper handling of input validation and sanitation. Nearly 90% of web application vulnerabilities originate from parameter tampering like injection, cross site scripting

(XSS), file and command execution attacks. Invalidated input was the first item in OWASP top ten list in 2004 [19] but removed in latest list (2007) [20] because it is an essential step towards securing the application and therefore it is not a type of vulnerability but the root of many application security problems.

Some of the web application development frameworks like Struts have built-in data validation mechanisms. In fact, data validation can be regarded as one of the core subjects of the positive security model. Positive security model tries to define what is allowed or normal for the application. The situations that are not defined are regarded as abnormal and rejected. The anomaly can be evaluated by predefined rules (white list) or by learning. Predefined rules can be inferred automatically by web site crawling or manually defined by strict and comprehensive resource and parameter mappings like all web pages and their allowed parameters and headers. On the other hand, learning can take place using statistical methods [21] or neural networks [22]. If we consider web applications, positive security model should work with any granularity from raw HTTP packets to HTTP parameters and headers. Network intrusion detection systems fail to satisfy this level of granularity; mostly they evaluate only on raw packet but discard the content. Any web application security system must allow all legitimate, acceptable traffic and content requirements and deny everything else. This approach is highly effective at preventing unknown attacks and dramatically reduces an organization's attack surface by automatically eliminating exposure to all sorts of attacks. The opposite of positive security is negative security model which identifies traffic known to be threatening by checking traffic flows against attack signatures. However with attack vectors increasing at such a rapid pace, solutions have less and less time to react to new attacks. Attack signatures must be updated rapidly and frequently.

Since network layer defense systems fail to confront application level attacks, the solution has emerged in the shape of application level firewalls, namely Web Application Firewall (WAF). According to WASC [23] a web application firewall is *"An intermediary device, sitting between a web-client and a web server, analyzing OSI Layer-7 messages for violations in the programmed security policy. A web application firewall is used as a security device protecting the web server from attack."* WASC has also released web application evaluation criteria [23] that can also be used for standardization. As public awareness increases, Web application firewalls become an essential part to secure any Enterprise web applications. For example, according to Payment Card Industry (PCI) Data Security Standard (DSS) [24] companies must install an application layer firewall in front of Web applications or have all custom application code reviewed for vulnerabilities by an outside organization that specializes in application security.

### 1.1.3 Proposed Solution

To sum up, access control and security are most common problems of enterprise applications. Executing enterprise-level security policies that encapsulates domain specific factors to requests that

suffer from web application vulnerability could probably result in error-prone access decisions. In order to decide on enterprise-level security policies, web requests must be free from application-level security vulnerabilities. So a correct access decision can only be granted if a request satisfies both "*enterprise-level*" and "*application-level*" security policies. Enterprises require a comprehensive solution that provides centralized security management, from authentication to authorization and auditing.

The aim of this thesis to describe a centralized access and security mechanism that combines "*enterprise-level*" and "*application-level*" security aspects together and enforce these policies to be satisfied organization-wide and in a transparent manner. The proposed solution called EYEKS ("Erişim, YEtkilendirme ve Kişiselleştirme Sistemi" in Turkish, meaning "Access, Authorization and Personalization System") was presented in Security of Information Networks 2007 (SIN2007) [7] The main goal of EYEKS is to provide a common evaluation and enforcement environment for both enterprise-level and application level policies to bring together access controlling with application-level security. Removing separation between enterprise-level and application-level security policies improves manageability, reusability and scalability of whole system.

Beznosov showed that separation of access decision mechanism with application itself is essential to encapsulate domain specific factors for access decision [5]. In his model, a reference model evaluates access decision using authorization database, gives the evaluation result to the application and leave the enforcement to the application. Therefore EYEKS has been designed as reverse proxy that works inline mode, installed in front of the web application to control the traffic and enforce security policies to be satisfied. Architecture of proposed solution introduces a specific layer, so called application security layer that is created and placed in frontier. Posterior layers consist of real web applications and databases and have no direct access to the outside world. All communications from outside world to backend web application is intercepted and authorized from application security layer. Each request is parsed into HTTP headers, parameters and content, and passed to the request/response operation chain, which is the core of application security layer. Each operation in the chain is responsible for a specific operation like authentication, authorization, session management and logging.

The authorization mechanism of EYEKS has been chosen as Resource Access Decision (RAD) because this facility is one of the best solutions that can be used by security-aware applications [5 and as shown in section 3.2, is very suitable to solve access control problems of web applications. RAD is a specification released by The Object Management Group (OMG) to specify a mechanism for obtaining authorization decisions and administrating access decision policies [25]. EYEKS uses CSAAS (Cybersoft Authentication and Authorization System) as authorization and authentication engine which implements RAD specification with additional RBAC [26] capabilities [6]. The details of CSAAS will be described in section 3.1.

RAD specification requires resources and their valid operations to be well defined, “Resource” can be any entity in computer system and operation defines a valid procedure performed on any resource, therefore any level of granularity to address access control problems of enterprise applications can be achieved. Every resource-operation pair can be combined with a number of “policies” that defines access policies to do requested operation on that resource. Access is granted only if that operation satisfies attached policy rules on specified resource.

The access control problems of enterprise applications can be resolved by defining enterprise-level policies to CSAAS. Policies are evaluated using attributes of an operation. These attributes can be dynamic (attribute value is evaluated at the time of the request) or static (parameters that are passed directly with an operation.) According to these attribute values, a policy grants or denies an access. For enterprise web applications, this is a reasonable approach. A form within a web page, a whole web page, a directory or even a whole web application can be defined as a resource or operations for a resource in upper level of abstraction. As described before, middleware access control mechanisms cannot provide such level of abstraction.

Any web application that requires to be controlled by EYEKS must be mapped to RAD domain as described in section 3.2. This mapping requires all web pages and directory structure to be identified and manually constructing resource-operation pairs. After the whole web application is mapped, access policies (enterprise-level security policy) can be attached to suitable resource-operation pairs that define permission on that resource. Within these policies, any domain specific access rules can be encapsulated. EYEKS regards all request parameters as security attributes of corresponding business operation and passes them to CSAAS. Upon reception of a page request, these parameters are resolved and according to corresponding mapping they are passed to CSAAS with resource-operation pair. These parameters can then be used to evaluate access decision within corresponding policy.

Mapping from enterprise web application structure to RAD domain will also provide a common way to tackle with application security. This is believed to be the most important contribution of this thesis. Currently, enterprises must install different solutions to access control and application security. However removing discrimination between application-level and enterprise level security policies and handling them by a common infrastructure would improve manageability, reusability and scalability of whole system. This mapping directly leads us positive security model where resources and operation of the application are strictly defined. Upon this mapping, it is also possible to check parameters and headers of each request against allowed values, type or range.

RAD specification does not allow hierarchical resource definition, permitting only flat structure. However it is absolute that applications need organization wide security policies. In this thesis, this drawback has been overcome with some predefined resources. Application security layer asks for permission on these resources when a page request, request to access any page within a directory or more broadly any request to application has been received. By these predefined resources, it is

possible enforce global security policies on any page, directory or application hierarchically. Global security enforcement also provides a negative security model to be applied. Application-security policies that define signatures of known security exploits can be attached to these resources and application security layer guarantees that each request must satisfy all these application security policies in order to reach backhand enterprise applications.

To summarize, the main contributions of this thesis are the following:

- Defining an organization wide security enforcement mechanism for enterprise web applications.
- Transparently adapting an access control mechanism based on RAD specification that is capable of using domain specific factors in access decision to address access control problems of enterprise applications.
- Defining a common infrastructure where enterprise access rules and application security rules can be handled by organization wide policies.
- Applying positive and negative security models to enterprise web application with RAD based implementation.

## 1.2 Outline of the Thesis

This thesis is organized as follows. The next chapter contains background information about access control mechanisms and web application security. It begins with security taxonomies and continues with access control mechanism, further defines access control and application security problems of enterprise web applications. At the end of the chapter related works about these two subjects are presented. Chapter 3 describes our proposed architecture. The chapter starts with describing CSAAS, RAD based implementation that will be used as access control mechanism and continues with mapping web application to RAD domain. In the remaining section of this chapter, the inner structure of EYEKS is described in detail. The chapter ends with verification of solution section that tries to verify EYEKS implementation against the problems presented. Chapter 4 lists the experimental tests of the EYEKS. In Chapter 5, main contributions of this thesis and some ideas for future work are presented. In Appendix A, a complete analysis of web application security vulnerabilities is made. This chapter also contains a classification of web application attacks according to selected security taxonomies.

# CHAPTER 2

# BACKGROUND AND RELATED WORK

## 2.1 Security Incidents

### 2.1.1 Definition of Incidents

There is no single definition of security incidents and incident responses. Throughout the literature there is no agreement on what an incident is. Instead individuals, foundations and universities make their own assessments to define what an incident is.

Lucas and Moeller [27], like most practitioners, agree on the need for a solid definition that clearly differentiate "incident" from "non-incident". The Network Working Group of TERENA emphasizes the importance of a common language on security incidents, on their RFC 3067 and states that: [28]

*"Computer Incidents are becoming distributed and International [sic] and involve many CSIRTs across borders, languages, and cultures. Post-Incident information and statistics exchange is important for future Incident prevention and Internet security improvement. The key element for information exchange in all these cases is a common format for Incident (Object) description."*

The first classification of computer incidents was defined by Nancy and Peter Finn in an article on Computerworld published in 1984. They divide computer crime into five categories: financial crime, information crime, theft of property, theft of services and vandalism. However they only focus on crime-related threats but discards the accidental or non-malicious aspects. Howard and Longstaff define an incident as "a group of attacks that can be distinguished from other attacks because of the distinctiveness of the attackers, attacks, objectives, sites, and timing." [29] An attack is defined as "a series of steps taken by an attacker to achieve an unauthorized result." And an attacker is "an individual who attempts one or more attacks in order to achieve an objective." TERENA defines an attack more solidly as "an assault on system security that derives from an intelligent threat to evade security services and violates the security policy of a system. Attack can be active or passive, by insider or by outsider, or via attack mediator." In both definitions, the term attack means a malicious intent to break down the system, however a sizeable proportion of incidents are the result of accidents or actions undertaken without seeing any negative consequences. This strict focus on malicious attacks does comprise just a portion of total incidents. So a more general definition of attack is needed to cover all types of security incidents.

Grace, Kent and Kim also emphasize the need for a clear definition of what an incident are [30]. They consider this as an inevitable aspect to create an effective indent response team. They state that "an

event is any observable occurrence in a system or network" and "adverse events" as "events with a negative consequence, such as system crashes, network packet floods, unauthorized use of system privileges, defacement of a Web page, and execution of malicious code that destroys data." This creates an important distinction; they called any occurrence of changes in the system that cause any effect as "events" and also adds the term "adverse event" that cause negative effect to the system. These authors make this distinction in ways that other authors have not.

Van Wyk and Forno focus on examples of incidents. They state simple definition of incident [31]. "In the most basic terms, an incident is a situation in which an entity's information is at risk, whether the situation is real or simply perceived". The significant part of their work is expanding the definition of incident to include situations that are false alarms. This definition adds another perspective to the computer security, they state that if only real incidents attracted the attention of incident response teams, in order to prevent the further attacks and collect knowledge, the damage or exposure would have to occur. Van Wyk and Forno states that perceiving an incident is possible by studying previous attacks, false alarms and possible vulnerability sources to take a more proactive approach

### 2.1.2 Taxonomies of Incidents

Taxonomy is a classification scheme that partitions a body of knowledge and defines the relationship of the pieces [32] Classification is the process of using taxonomy for separating and ordering. Using these separations and ordering generalizations can be made about them, so we can say that classifications have explanatory value. Taxonomies can also be used to predict the existence of specimens that have not been seen before by extrapolating from the known specimens so taxonomies have also predictive value.

Edward Amoroso, in his book Fundamentals of Computer Security Technology [33] defines what the characteristics, a satisfactory taxonomy must have. These are;

**Mutually Exclusive:** classifying in one category excludes all others because categories do not overlap,

**Exhaustive:** taken together, the categories include all possibilities,

**Unambiguous:** clear and precise so that classification is not uncertain, regardless of who is classifying,

**Repeatable**: repeated applications result in the same classification, regardless of who is classifying,

**Accepted:** logical and intuitive so that categories could become generally approved,

**Useful:** could be used to gain insight into the field of inquiry.

**Ivan** Victor Krsul defines these characteristics in a more compact way and mention four distinctive characteristics, which are; [34]

**Objectivity:** The features must be identified from the object known and not from the subject knowing. The attribute being measured should be clearly observable.

**Determinism:** There must be a clear procedure that can be followed to extract the feature.

**Repeatability:** Several people independently extracting the same feature for the object must agree on the value observed.

**Specificity:** The value of the feature must be unique and unambiguous.

Ulf Lindvist and Erland Jonsson were more or less in agreement with these characteristics but also added two important characteristics [35]

**Comprehensible:** A satisfactory taxonomy must be able to be understood by who are in the security field, as well as those who only have an interest in it.

**Complying Terminology:** An accepted terminology should be used in taxonomy to avoid confusion and to build on previous knowledge

### 2.1.3 Existing Taxonomies

As some authors like Landwher and Bishop, focus on mainly attacks, some authors like Cohen, Howard take a broader view of the taxonomies such as considering the attacker, tool or natural disasters. So computer and security taxonomies do not necessarily focus on attacks. Regardless of whether the taxonomy focuses on attacks or not, the common element of these taxonomies is classifying attacks.

There are various works on creating taxonomies for computer security incidents and nearly all of them can be categorized by their common properties

#### 2.1.3.1 List of Terms

One of the popular and simple taxonomy of computer incidents is giving a list of single and defined terms. Icove [13] proposed 24 terms as taxonomy, as shown below.

*"Wiretapping, Dumpster diving, Eavesdropping on Emanations, Denial-of-service, Harassment, Masquerading, Software piracy, Unauthorized data copying, Degradation of service, Traffic analysis, Trap doors, Covert channels, Viruses and worms, Session hijacking, Timing attacks, Tunneling, Trojan horses, IP spoofing, Logic bombs, Data diddling, Salamis, Password sniffing, Excess privileges, Scanning."*

Cohen defined 39 terms in his paper. [36] and extend this list to 100 terms [37] but also added that the classification is descriptive, non-orthogonal, incomplete, and of limited applicability. The defined 39 terms are shown below.

*"Trojan horses, Toll fraud networks, Fictitious people, Infrastructure observation, E-mail overflow, Time bombs, Get a job, Protection limit poking, Infrastructure interference, Human engineering, Bribes, Dumpster diving, Sympathetic vibration, Password guessing, Packet insertion, Data diddling, Computer viruses, Invalid values on calls, Van Eck bugging, Packet watching, PBX bugging, Shoulder surfing, Open microphone listening, Old disk information, Video viewing, Backup theft, Data aggregation, Use or condition bombs, Process bypassing, False update disks, Input overflow, Hang-up hooking, Call forwarding fakery, Illegal value insertion, E-mail spoofing, Login spoofing, Induced stress failures, Network services attacks Combined attacks."*

Ambiguities are almost inevitable when preparing lists of terms for taxonomies; the terms tend not to be mutually exclusive, which is the main characteristics of a satisfactory taxonomy. For example, the terms virus and logic bomb are not mutually exclusive since a virus may contain a logic bomb. And also attacks do not consist of only one type of attack but a combination of different methods. As a result, developing a comprehensive list of methods for attacks would not provide a classification scheme that yields mutually exclusive categories (even if the individual terms were mutually exclusive), because actual attacks would have to be classified into multiple categories

### 2.1.3.2 List of Categories

Listing of categories is a variation of the list of terms. Lists of categories are in fact distinctive categories holding definitions of underlying terms. Cheswick and Bellovin in their paper on firewalls [14] classify attacks into seven categories as follows;

*"1.Stealing passwords - methods used to obtain other users' passwords,*

*2. Social engineering - talking your way into information that you should not have,*

*3. Bugs and backdoors - taking advantage of systems that do not meet their specifications, or replacing software with compromised versions,*

*4. Authentication failures - defeating of mechanisms used for authentication,*

*5. Protocol failures - protocols themselves are improperly designed or implemented,*

*6.Information leakage - using systems such as finger or the DNS to obtain information that is necessary to administrators and the proper operation of the network, but could also be used by attackers,*

*7. Denial-of-service - efforts to prevent users from being able to use their systems."*

Aslam develops a classification scheme which focuses on security faults that result in security incidents. [38] In this narrow point of view, he divides software faults into two broad categories. *Coding Faults* that result from errors in programming logic, missing requirements, or design error; and *Emergent Faults* resulting from improper installation or administration of software so that software faults are present even if there are no faults in coding part.

One of the newest taxonomy falls into this kind of taxonomy is Lough's taxonomy. In 2001 Daniel Lough proposed another taxonomy named VERDICT (Validation Exposure Randomness Deallocation Improper Conditions Taxonomy) that is based on characteristics of attacks. Lough proposed four characteristics of attacks; [15]

**Improper Validation:** Insufficient or incorrect validation results in unauthorized access to information or a system.

**Improper Exposure:** A system or information is improperly exposed to attack.

**Improper Randomness:** Insufficient randomness results in exposure to attack.

**Improper Deallocation:** Information is not properly deleted after use and thus can be vulnerable to attack.

List of categories are an improvement because the taxonomy have some structure of terms, but this type of taxonomy suffers from the same problems as on large list of terms. For example Bishop and Bailey [39] shows that Aslam classification does not satisfy the specificity requirement as it is possible to classify a fault in more than one classification categories. So this type of taxonomies also suffers from satisfying mutual exclusive characteristic.

#### 2.1.3.3 Result Categories

This classification identifies the impact of vulnerability; in fact it is another variation of the list methods to group all attacks into basic categories that describe the result of attack. Cohen's taxonomy [36] also covers result categories such as *corruption, leakage and denial*, where corruption is the unauthorized modification of information, leakage is when information ends up where it should not be, and denial is when computer or network services are not available for use [36]. Russell and Gangemi use similar categories but define them using opposite terms: 1) secrecy and confidentiality; 2) accuracy, integrity, and authenticity; and 3) availability [40].

This type of taxonomy has a useful framework because most individual attacks eventually fall into one of these categories. Although the attack techniques, tools can be various, the impact list would be compact. One drawback of these taxonomies is an attack can result in not only direct impact but also indirect impact. So there might be confusion on which categories it belongs to. However result

categorization scheme ends up with empirical, lists, vulnerability databases and decision tree taxonomies, which are serious improvements in taxonomies of security incidents

### 2.1.3.4 Empirical Lists

A variation of result categories is to develop a longer list of categories based upon a classification of empirical data. Neumann and Parker classified actual attacks and came up with eight categories. Three classical categories (corruption, leakage and denial) now extended into eight distinct categories so that it covers more types of attack impact, which would not be classified by Cohen's taxonomy. Neumann and Parker list it as follows; [41]

*"External Information Theft (glancing at someone's terminal).*

*External Abuse of Resources (smashing a disk drive).*

*Masquerading (recording and playing back network transmission).*

*Pest Programs (installing a malicious program).*

*Bypassing Authentication or Authority (password cracking).*

*Authority Abuse (falsifying records).*

*Abuse Through Inaction (intentionally bad administration).*

*Indirect Abuse (using another system to create a malicious program)."*

However Amoroso critiques the list as follows; [33]

*"A drawback of this attack taxonomy is that the eight attack types are less intuitive and harder to remember than the three simple threat types in the simple threat categorization. This is unfortunate, but since the more complex list of attacks is based on actual occurrences, it is hard to dispute its suitability."*

Such extended lists of result categories can be suitable for classifying large number of actual attacks and if carefully constructed, these list would have satisfy able taxonomies characteristics stated above. However, being able to classify known attacks is not sufficient, as Amoroso said, a successful taxonomy must be logical and intuitive, so that new attacks can also be classified using the same taxonomy. There must be additional structure showing the relationship of the categories.

### 2.1.3.5 Matrices

The most used representation style of taxonomies is in matrix form. Perry and Wallich create one of the first matrix taxonomy. They present a classification scheme based on two dimensions;

vulnerabilities and potential perpetrators. This allows categorization of incidents into a simple matrix [83]. The individual cells of the matrix represent combinations of potential perpetrators: operators, programmers, data entry clerks, internal users, outside users, and intruders, and the potential effects: physical destruction, information destruction, data diddling, theft of services, browsing, and theft of information.

One of the most valuable works of matrix approach to security incident taxonomy is found in Landwehr's "A Taxonomy of Computer Program Security Flaws, with Examples" paper [84]. They present taxonomy of computer security flaws based on three dimensions;

- Genesis: How a security flaw occurs,
- Time of Introduction: In which life-cycle of the software, a security flaw arises,
- Location: Where and in which state, a security flaw occurs,

Although, Landwehr's taxonomy is a good effort for classifying security incidents, it has many drawbacks. First of all Landwehr used the terms, such as Trojan horse, virus, trapdoor and logic/time bomb for which there are no accepted definitions. The taxonomy includes several "other" categories which make the flaws not to represent an exhaustive list. On the other hand most of the attacks could use several flaws and behave differently in different platforms. So it is hard to classify entire attack using this taxonomy.

#### 2.1.3.6 Process-Based Taxonomy

The focus of this kind of taxonomies is toward a process, rather than a single classification category, in order to provide both a successful classification scheme for Internet attacks, and also a taxonomy that would aid in thinking about computer and network security.

Stallings presents a simple process model that classifies security threats [17]. The model is focused only information in transit. Stallings defines four categories of attack as follows:

*"1.Interruption - An asset of the system is destroyed or becomes unavailable or unusable.*

*2. Interception - An unauthorized party gains access to an asset.*

*3. Modification - An unauthorized party not only gains access to, but tampers with an asset.*

*4. Fabrication - An unauthorized party inserts counterfeit objects into the system."*

Interception is viewed by Stallings as a passive attack, and interruption, modification and fabrication are viewed as active attacks. While this is a simplified view with limited utility, its emphasis on the process of attack is useful.

### 2.1.3.7 Threat Classification

The classification of the threat due to the vulnerabilities was designed by Power [42]. In this classification, *figure 1*, threats are divided into four categories, threats that threaten availability and usefulness, integrity and authenticity, confidentiality and possession, exposure to threats. Each threat category is divided into possible outcomes. But this classification is critiqued to be ambiguous. The categories *Observe* and *Access* are concrete actions while the category *Steal* is subjective, also it is possible to *Access* and *Steal* simultaneously.

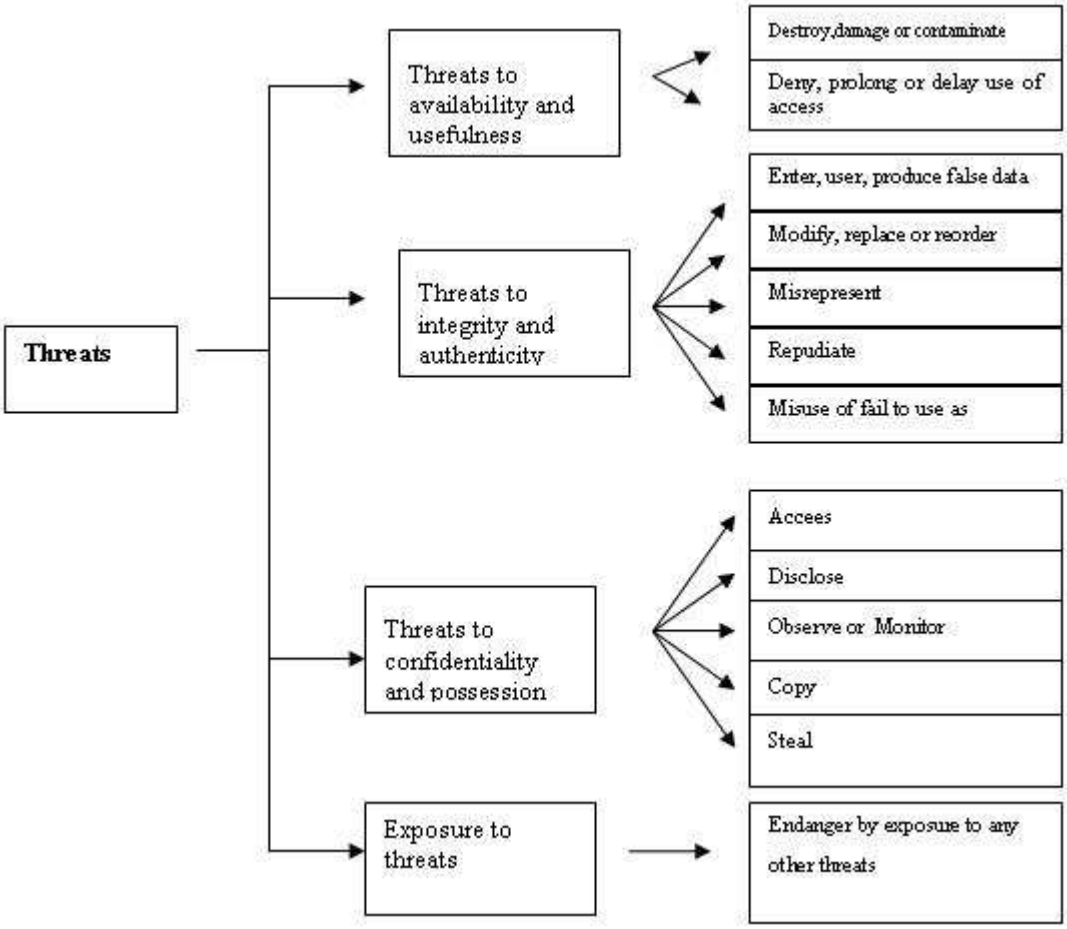


**Figure 1 Threat Classification**

#### 2.1.3.8 Vulnerability Databases

Several groups have constructed vulnerability databases. Private databases of restricted distribution include the CMET database at the Air Force Information Warfare (AFIW) Center; the database maintained by Mike Neumann; the database at the Computer Emergency Response Team (CERT); the database of the Australian Computer Emergency Response Team (AUSCERT); and the internal vulnerability databases at Netscape, Sun, and Haystack Labs.

National Vulnerability Database is a comprehensive security vulnerability database that integrates all publicly available U.S. Government vulnerability resources and provides references to industry resources. It is based on CVE vulnerability naming standard. It integrates together all publicly available U.S. government vulnerability resources within a single search engine and an average of 18 vulnerabilities is added on their database everyday.

These databases are freely available in the Internet and commonly used by various security related organizations and companies. Most of them have a simple characterization that includes information regarding the systems affected by the vulnerability and the potential ultimate impact that the vulnerability can have in a system and manner of possible attack. However, these categorizations are list type and fail to be a successful taxonomy.

### 2.1.4 Vulnerability Naming Standards

#### 2.1.4.1 Preliminary List of Vulnerability Example for Researchers

The Preliminary list of vulnerability examples for researchers is written by Steve Christey and is a working document that lists over 1400 diverse, real-world examples of vulnerabilities, identified by their CVE number [85]. Apart from past efforts that have largely focused on high-level theories, taxonomies, or schemes that do not sufficiently cover the wide variety of security issues, PLOVER provides an effective vocabulary for describing vulnerabilities at a low level of detail within a detailed conceptual framework.

In section 3 and 4 of this document, Christey gives definitions of security concepts with corresponding naming standards. He defines attack as follows; [PLOVER 2006, [DEFS].CDEFS.Core definitions]. *"The set of actions by which an ATTACKER follows an ATTACK VECTOR to exploit a VULNERABILITY to achieve a desired CONSEQUENCE."* In this definition, *"attack vector"* stands for a set of *"manipulations"* and *"channels"* where "channels" defines an interface between two entities of any system (Figure 7). Channels divided into three remote, local and physical. Remote channels mean any user to server or server to server interactions. Local channels are program interactions with local environment such as memory, file or programmatic interactions such as process invocation, object reference, data stream. Physical channels include serial ports, keyboard, CD drive, etc. Manipulations can be data or step manipulations.

Christey defines vulnerability as [PLOVER 2006, [DEFS].CDEFS.Core definitions]; *"A WIFF in a specific product, or a design intended for a class of products that provide the same functionality that has at least one ATTACK VECTOR."* where WIFF's are *"Weakness, Idiosyncrasy, Flaw, or Fault. An algorithm, sequence of code, or a configuration in the product, whether it arises from implementation, design, or other processes, that can cross data or object boundaries that could not be crossed during normal operation of the product."*

Attack vector consist of minimal set of MANIPULATIONS, and CHANNELs, that are required to cause the product to reach a WIFF. This definition is an important definition because Christey now able to categories attacks that use multiple WIFF's where most of the taxonomies fail to be mutually exclusive. Christey introduces, MULTI-FACTOR VULNERABILITY as *"A vulnerability that contains two or more WIFFs, two or more manipulations, or two or more attack channels."* and MULTI-CHANNEL VULNERABILITY as *"A vulnerability whose attack vector contains two or more attack channels that must be controlled by the attacker."*

Christey also categories vulnerabilities according to their origin, identifying in which phase of software life cycle the vulnerability is introduced. According to Christey, although most vulnerability tends to occur in any of several phases, some vulnerability can be introduced in one phase or another. In section 8, "Genesis of vulnerabilities", he divided the origin into 9 categories; design, implementation, bundling, distribution, installation, configuration, documentation, patch and removal.

As a result of PLOVER work, the vulnerabilities are organized within a detailed conceptual framework that currently enumerates 290 individual types of WIFFs and lists over 1400 diverse, real-world examples of vulnerabilities, identified by their CVE names. This work is a great step over standardization of enumeration of vulnerabilities and lead to OVAL (Open Vulnerability and Assessment Language), the standard for determining vulnerability and configuration issues on computer system. Depending on OVAL, the Department of Defense, give the statement of works to explain the relevant requirements that must be met by software suppliers, assessment and reporting tool developers, remediation tool developers.

#### 2.1.4.2 Common Vulnerabilities and Exposures (CVE)

CVE is a list of information security vulnerabilities and exposures that aims to provide common names for publicly known problems. It is a dictionary and a result of collaborative efforts of CVE Editorial Board, which consist of numerous security-related organizations such as security tool vendors, academic institutions, and government as well as other security experts. It is freely available for both download and review. CVE was founded in 1999 and since then it tries to enumerate common vulnerabilities. It does not provide any taxonomy; instead CVE is designed to allow vulnerability databases and other capabilities to be linked together, and to facilitate the comparison of security tools and services. As such, CVE does not contain information such as risk, impact; fix

information, or detailed technical information. CVE only contains the standard name with status indicator, a brief description, and references to related vulnerability reports and advisories.

CVE gives two new definition to the term vulnerability; “universal vulnerability” and “exposure” [11]. CVE defines universal vulnerability as follows”*A "universal" vulnerability is one that is considered vulnerability under any commonly used security policy which includes at least some requirements for minimizing the threat from an attacker.”* And states “exposure” as

*“An exposure is a state in a computing system (or set of systems) which is not a universal vulnerability, but either:*

*1. allows an attacker to conduct information gathering activities*

*2 .allows an attacker to hide activities*

*3 .includes a capability that behaves as expected, but can be easily compromised*

*4 .is a primary point of entry that an attacker may attempt to use to gain access to the system or data*

*5 .is considered a problem according to some reasonable security policy”*

In fact, these two definitions are very broad and it is hard to decide whether a security incident is a *“universal vulnerability”* or an *“exposure”*. However the term *“universal vulnerability”* is used for entries, which are considered as vulnerabilities under any security policy and exposure as entries, which violate some of the security policies. Under these definitions, “denial of service by flooding a network” and “remote command execution as user nobody" are examples of universal vulnerability and “running services such as finger” and “inappropriate settings for Windows NT auditing policies” can be called exposures.

#### 2.1.4.3 Common Weakness Enumeration (CWE)

PLOVER is a starting point for creation of CWE. CWE tries to give a formal enumeration of the set of security Weakness, Idiosyncrasies, Faults, Flaws (WIFFs) to serve as a common language for describing software security vulnerabilities. Although the basis of CWE is PLOVER work, CWE also includes the thoughts in the McGraw/Fortify “Seven Kingdoms” taxonomy, Howard, LeBlanc & Vieag’s 19 Deadly Sins and Secure Software’s CLASP.

At the top of the hierarchy, CWE categories WIFFs into two; By Location and By Motivation/Intent. Motivation/Intent group is divided into Intentional and Inadvertent. Intentional WIFFs are weakness that occurs intentionally and Inadvertent flaw may occur in requirements and as well as during specification and coding. Intentional flaws are also divided into malicious flaws and non-malicious flaws. Malicious flaws cover Trojan horses, trapdoors and other malicious software that can leak into

the software. Functional requirements that are written without regard to security requirements can lead to non-malicious flaws.

Location category describes the origin of flaws, a flaw can occur because of the environment used, some faults in configuration or the coding mistakes. Coding faults are divided into two source code and byte/object code. Source code describes coding errors that lead to weakness and byte/object code is a category that tries to describe the weaknesses that rise from bad linking and complying practices. Source code category is divided into seven categories, these are; Data Handling, API Abuse, Security Features, Time and State, Error Handling, Code Quality, Encapsulation which describes bad coding practices that can leak to vulnerabilities. A high level hierarchy of CWE is given in *figure-2*.

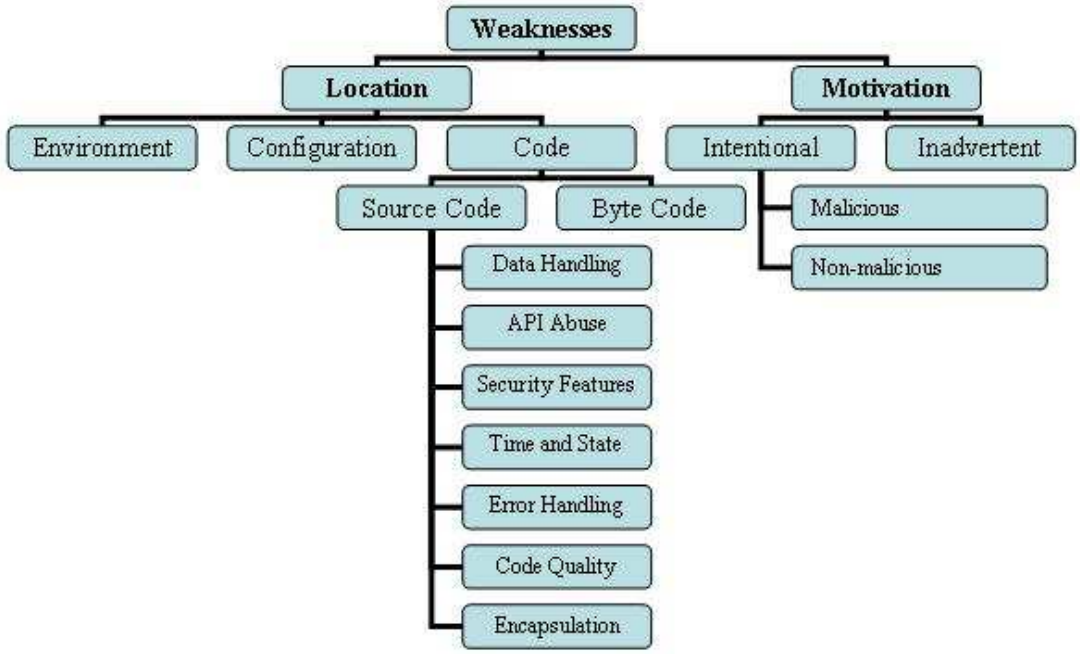


**Figure 2 CWE Enumeration**

### 2.1.4.4 WASC Threat Classification

Web application security consortium has been released a classification of web application threats. According to this classification, web application suffers from 6 classes of attacks; Authentication, Authorization, Client-side Attacks, Command Execution, Information Disclosure and Logical Attacks.

Authentication type of attacks covers attacks that target a web site's validation of the identity of a user, service or application mechanism. Authorization type of attacks aims bypassing authentication mechanisms to perform any action without sufficient permissions. Client-side Attacks focuses on the abuse or exploitation of a web site's users. The Command Execution section covers attacks designed to execute remote commands by injecting malicious input on the web site. Information Disclosure types of attacks tries to reveals sensitive data, such as developer comments or error messages or the full structure of web site which may aid an attacker in exploiting the system. Logical Attacks section covers the abuse of a web application's logic flow; attacker may bend expected procedural flow in order to perform a certain malicious action

## 2.2 Access Control Mechanism

Security of computer systems can be conventionally defined by two terms, protection and assurance. Protection is based on the idea that it is always possible to define most of the threats that may happen, and to build mechanisms that can prevent the threats [2]. The protection mechanisms must provide the essential services of accountability, availability and authorization. Accountability mechanisms make sure that any actions done by the users or other system active entities (subjects) towards the system resources (objects) are logged and the logs should be sufficient to map the subject to a controlling user. Availability mechanisms ensure either service continuity or service and resource recovery after interruption. Authorization mechanisms should ensure that the rules governing the use of system resources are enforced application-widely. Access control mechanisms allow system owner to define these governing rules and to enforce them. The term "authorization" also implies the process of making access control decisions.

In any access control model, the entities that can perform actions in the system are called subjects; and the entities representing resources to which access may need to be controlled are called objects (see also Access Control Matrix). Subjects and objects should both be considered as software entities, rather than as human users: any human user can only have an effect on the system via the software entities that they control. Access control has been exercised at different places and levels of abstraction, e.g. network, database, operating system and middleware controls, each with different emphasis. Control to protected resources can also be addressed from a single system or an organization point of view.

Broadly, access control models used by current systems tend to fall into one of two classes: those based on capabilities and those based on access control lists (ACLs). In a capability-based model, access to the object requires holding a capability that object defines; another party provides access by transmitting such a capability over a secure channel. In an ACL-based model, a subject's access to an object depends on whether its identity is on a list associated with the object; editing the list controls access.

Beznosov clarifies the structure of traditional access control mechanisms using the conceptual model of reference monitor [43]. A reference monitor is a part of the security subsystem, responsible for mediating access by subjects to system resources as shown in *figure 3*. So the access control becomes the act of checking access requests against authorization rules from the authorization database when a subject requires action on system objects and enforcing them. A set of the rules is sometimes called a policy. Authorization rules commonly have a subject-action-object structure, which specifies what subject(s) can perform what action(s) on what object(s). Permitted actions are called access rights. Thus a subject has a particular access right to an object if the action is permitted towards that object. So a reference monitor requires authorization rules and three groups of information: 1) the access request, 2) the subject who made the request, and 3) the object to be accessed to make an authorization decision.

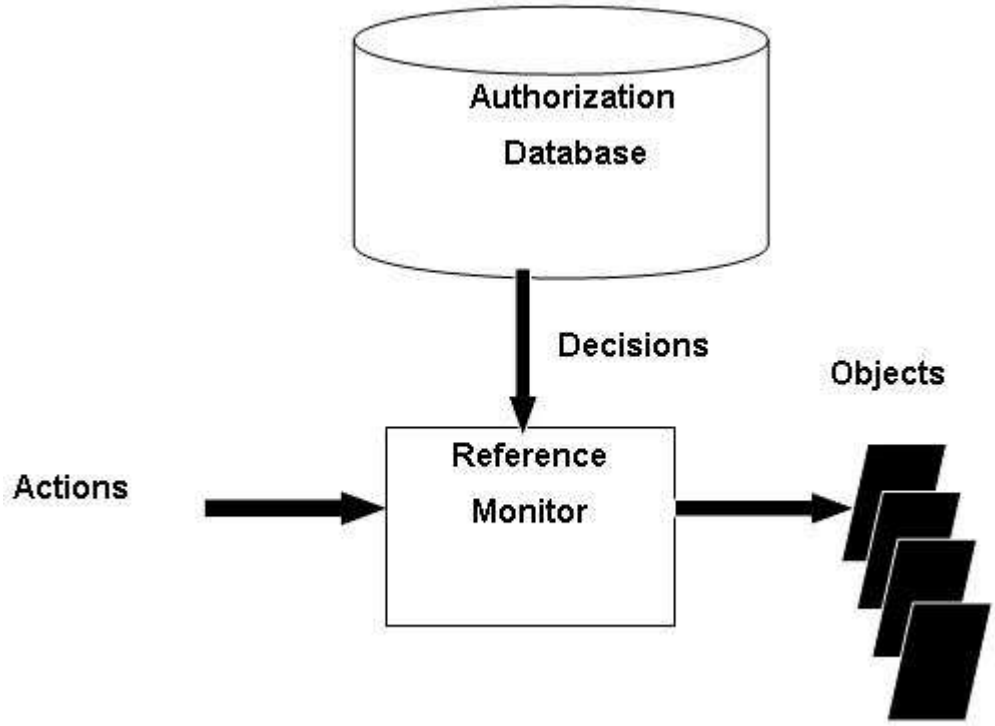


**Figure 3 Conceptual Model of Access Control**

### 2.2.1 Discretionary Access Control

Discretionary access control (DAC) is a kind of access control, defined by the TCSEC [44] as *"A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by Mandatory Access Control)."*

The basis of this kind of security is that an individual user, or program operating on the user's behalf, is allowed to specify explicitly the types of access other users (or programs executing on their behalf) may have to information under the user's control. Access controls may be discretionary in capability, profile, access control list, protection bits and password based [45].

Discretionary security differs from mandatory security in that it implements the access control decisions of the user. Mandatory controls are driven by the results of a comparison between the user's trust level or clearance and the sensitivity designation of the information. Discretionary controls are not a replacement for mandatory controls. In any environment in which information is protected, discretionary security provides for a finer granularity of control within the overall constraints of the mandatory policy. However Discretionary access control mechanisms restrict access to objects based solely on the identity of subjects who are trying to access them. This basic principle of discretionary access control contains a fundamental flaw that makes it vulnerable to Trojan horses [46].

### 2.2.2 Mandatory Access Control

Mandatory access control, as defined in the DoD's Trusted Computer Security Evaluation Criteria [44], is *"A means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e. clearance) of subjects to access information of such sensitivity."*

MAC's basic idea is denying users to full control over the access to resources that they create. The system security policy entirely determines the access rights granted, and a user may not grant less restrictive access to their resources than the administrator specifies. For MAC, the access control decision is granted by verifying the compatibility of the security properties of the data and the clearance properties of the individual MAC is most commonly applicable to Classified National Security Information where best effort mechanisms are inadequate; absolute enforcement is mandated.

If individuals or processes exist in the system environment that may be denied access to any of the data in the system environment, then the system must be trusted to enforce MAC. This implies varying degrees of robustness in the system. For example, more robustness is indicated for system environments containing classified Top Secret information and uncleared users than for one with Secret information and users cleared to at least Confidential. To promote consistency and eliminate

subjectivity in degrees of robustness, an extensive scientific analysis and risk assessment of the topic produced a landmark benchmark standardization quantifying security robustness capabilities of systems and mapping them to the degrees of trust warranted for various security environments.

Such architecture prevents an authenticated user or process at a specific classification or trust-level from accessing information, processes or devices in a different level. This provides a containment mechanism of users and processes, both known and unknown (an unknown program (for example) might comprise an untrusted application where the system should monitor and/or control accesses to devices and files).

### 2.2.3 Lattice-based Access Control

Lattice-based access control (LBAC) is a complex method to control information flow of the system. It decides access on combination of objects and subjects by checking partial ordering of the security levels.

A lattice is used to define the levels of security that an object may have, and that a subject may have access to, in such a way that any two security levels always have a greatest lower bound and least upper bound. If two objects A and B are inherited by another object C, that object is assigned a security level formed by the join of the levels of A and B, and if two subjects need to access some secure data, their access level is defined to be the meet of the subjects' levels. A subject is allowed to access an object only if the security level of the subject is greater than or equal to that of the object.

### 2.2.4 Rule-based Access Control

Rule-based access control is an example of mandatory access control where the system decides on actions of subjects on objects by evaluating a chain of rules that have been defined previously. In fact all MAC-based systems implement a simple form of rule-based access control to determine whether access should be granted or denied, however rule-based access control differs from others due to expressional ability. With a set of well defined rules, the access control logic can be embedded purely in rules that can be evaluated at run time.

Rule-based access control is a strategy to manage user access to one or more systems, where business changes trigger the application of rules, which specify access changes. These rule evaluation can give system dynamic manner, when rules depends dynamic variables that can be changed during execution

### 2.2.5 Role-based Access Control

Role-based access control (RBAC) is an alternative to traditional discretionary (DAC) and mandatory access control (MAC) policies [47] [48]. RBAC's main motivation is the ability to specify and enforce enterprise-specific security policies in a way that maps naturally to an organization structure. RBAC suits well for expressing policies particularly suited for commercial application.

Within RBAC, access control policies must be expressed in terms of the organization structure and roles that individuals have. There is a direct mapping from organization view to access control domain view so that it is not necessary to translate a natural organization view into access control mechanism. In fact RBAC is a form of non-discretionary access control that the users are constrained by the organization's protection.

Within the RBAC framework, a user is a person, a role is a collection of job functions, and an operation represents a particular mode of access to a set of one or more protected RBAC objects. And there is a many-to-many relationship between users, role and operations as shown in *figure* 4. For example, a single user can be associated with one or more roles, and a single role can have one or more user members. Roles can be created for various job positions in an organization. System objects that requires authorization is matched with possible operations.

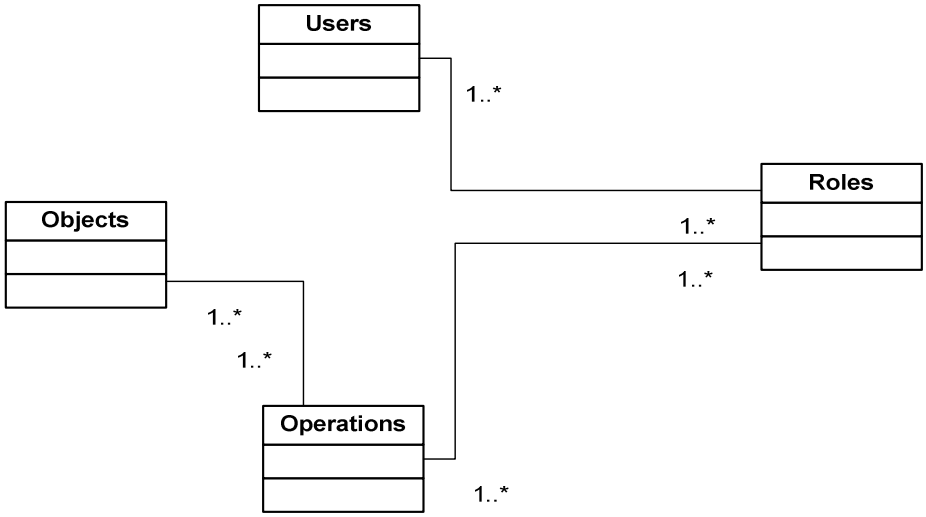


**Figure 4 RBAC Role Model**

Roles can have overlapping responsibilities and privileges, that is, users belonging to different roles may need to perform common operations, so that it would be inefficient to specify repeatedly these operations for each role. Consequently, RBAC introduces the concept of role hierarchies. A role hierarchy defines roles that have unique attributes and that may "contain" other roles, that one role may implicitly include the operations, constraints, and objects that are associated with another role.

With these basic foundations, Ferraiolo defines nine rules for RBAC access control; [49]

**Rule 1 (Role Hierarchy):** If a subject is authorized to access a role and that role contains another role, then the subject is also allowed to access the contained role.

**Rule 2 (Static Separation of Duty):** A user is authorized as a member of a role only if that role is not mutually exclusive with any of the other roles for which the user already possesses membership.

**Rule 3 (Cardinality):** The capacity of a role cannot be exceeded by an additional role member.

**Rule 4 (Role Authorization):** A subject can never have an active role that is not authorized for that subject.

**Rule 5 (Role Execution):** A subject can execute an operation only if the subject is acting within an active role.

**Rule 6 (Dynamic Separation of Duty):** A subject can become active in a new role only if the proposed role is not mutually exclusive with any of the roles in which the subject is currently active.

**Rule 7 (Operation Authorization):** A subject can execute an operation only if the operation is authorized for the role in which the subject is currently active.

**Rule 8 (Operational Separation of Duty):** A role can be associated with an operation of a business function only if the role is an authorized role for the subject and the role had not been assigned previously to all of the other operations.

**Rule 9 (Object Access Authorization):** A subject can access an object only if the role is part of the subject's current active role set, the role is allowed to perform the operation, and the operation to access the object is authorized.

Sandru [48] provides a characterization of RBAC models as follows;

1. RBAC0: The basic model with users associated with roles and roles associated with permissions.

2. RBAC1: Role hierarchies are added to RBAC0.

3. RBAC2: RBAC1 with adding constraints on user to role, role to role and role to permission associations.

RBAC system enables administration of a broad range of authorized operations more easily and provides great flexibility and breadth of application. System administrators can control access at a level of abstraction that is natural to the way that enterprises typically conduct business rules. This is in contrast to conventional methods such as access control list (ACL), capabilities models.

### 2.2.6 Resource-based Access Control

The Resource Access Decision (RAD) specification released by The Object Management Group (OMG) is a mechanism for obtaining authorization decisions and administrating access decision policies [25]. In Beznosov's work this facility is cited as one of the best solutions that can be used by security-aware applications [5].

The major motivations behind RAD specification can be listed as follows;

- The application logic must be separated from authorization logic by providing a logically single point of administrative reference monitoring separated from application systems.

- The authorization decisions for resources (objects) must be defined for any nature and granularity as long as those resources defined according to RAD's resource naming scheme.

- More than one authorization engine for decisions can be consulted about the same request or different requests. These engines can support different authorization policies, can be integrated with legacy systems and can be managed by independent authorities.

- Authorization decisions can be granted using request-specific or user-specific factors which may dynamically changed during execution.

The main objective of RAD is to separate authorization logic from application logic. Authorization logic is encapsulated into an authorization service external to the application. The interaction diagram is shown in *figure* 5.

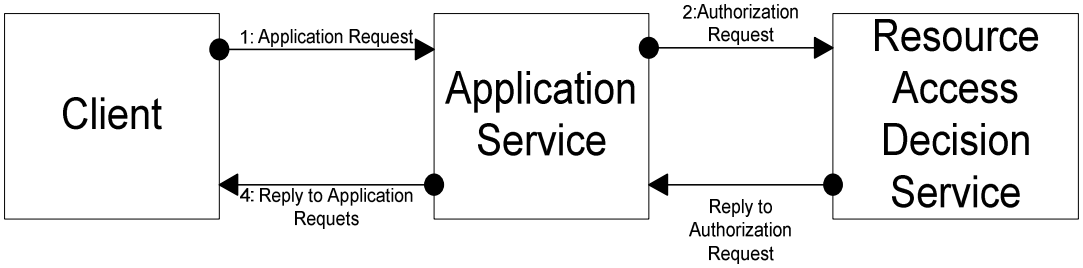


**Figure 5 RAD Interaction Diagram**

RAD specification requires resources and their valid operations to be well defined, resource can be any entity in the computer system and operation defines a valid procedure performed on any resource. Every resource-operation pair can be combined with a number of “policies” that defines access policies to do requested operation on that resource. Access is granted only if that operation satisfies attached policy rules on specified resource. Policies are evaluated using attributes of an operation. These attributes can be dynamic (attribute value is evaluated at the time of the request) or static (parameters that are passed directly with an operation.) According to these attribute values, a policy grants or denies an access. A conceptual model of these relations is given in *figure* 6.

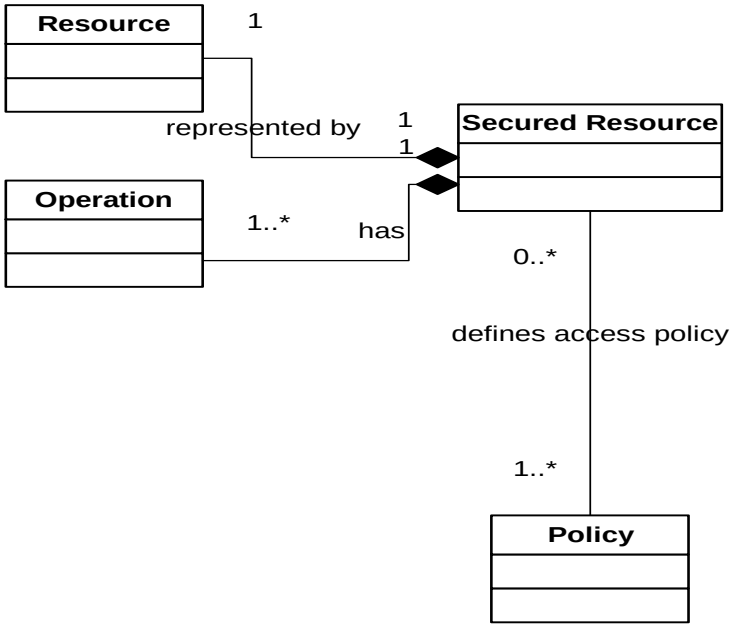


**Figure 6 RAD Secured Resource**

RAD allows different kinds of policy evaluators that can be plugged in to the system. All kinds of policies can be evaluated by adding capable policy evaluator. Access decisions can be evaluated by combining different kinds of policy evaluators to evaluate different kinds of policies.

Barkley shows that RBAC can be combined with RAD by introducing RBAC policy evaluator so that RAD can also use all capabilities and advantages of RBAC system [2]. In this thesis, an access control mechanism that uses RAD specification and with RBAC policy evaluator is implemented. The details of the concept can be found in section 3.1.

## 2.3 Access Control Problems in Enterprise Applications

The Internet is forcing enterprises to implement collaborative business and governmental solutions that integrate internal systems. Enterprise applications such as ERP (Enterprise Resource Plan), CRM (Customer Relationship Management) and SCM (Supply Chain Management) have now all become online and web-based. Enterprise information portal technologies have emerged to integrate these applications into a cohesive whole. These enterprise applications must satisfy complex access control rules that rise from both business logic and integration of business transactions in order to be secure. At this point access control rules break from black-list or white-list implementations, but become so called *"enterprise-level security policies"*. However as access control logic becomes closer to enterprise level, policy rules become more dynamic, more domain-specific, and more contexts dependent. In fact, all business objects in enterprise applications can be a source of access policies with their underlying business rules. And collection of these domain-specific access policies defines the "enterprise-level security" policies. Traditional access control mechanisms fail to employ domain-specific factors in access decisions and therefore unsuitable to fulfill the needs of enterprise access control. [5, 43]

Most enterprise applications tackle this problem by embedding access control rules within an application code that handles domain-specific factors. The more access control rules are embedded in enterprise applications, the more reusability and manageability of whole system reduces. But according to the separation of concerns principle [50] "enterprise-level security" policies must be separated from application code and handled independently by the external access control mechanism. However, even if the application leaves access control decisions to the external system and interact with the access control service though API, it is the developer's responsibility to enforce the application-specific access policy in the code. [51]. Embedding this imperative access control makes it difficult to adapt the access logic to policy or application changes. Imperative access control enforcement is error prone and hard to spread over organization-wide.

Beznosov [43] lists 7 evaluation criteria to comprise an access control system, which also directly reflects access control problems in enterprise applications;

**Granularity of protected resource:** Enterprise applications may require different granularity levels to protect resources. Whole application can be defined as a resource as well as database tables, sensitive methods and interfaces can be regarded as a resource for enterprise application. So enterprise applications require allowing authorization decisions on fine-grained resources.

**Support for policies specific to an organization or application domain:** There are different kinds of access control mechanism as described in section 2.2. Enterprise application may require different kind of access control mechanisms in order to reflect all kind of enterprise policies, so in general the

more access control policy types an access control system supports, the easier it is to configure for enterprise policies.

**Information used for making authorization decisions:** Information about the subject can be divided into two types, security-related and security-unrelated. Typical access control mechanism only use security related information such as subject's identity, group membership, security clearance, however enterprise applications also requires security-unrelated information about the subject such as person age, data comes from work-flow execution.

**Use of application-specific information:** Beznosov [5] has defined domain (application)-specific factors in security decisions as follows; "An application-specific factor is a certain characteristic or property of an application's resource, produced, modified and processed in the course of normal application execution and not for the sole purpose of a security policy decision." In this point of view all business objects in enterprise applications can be the source of access policies with their underlying business rules and collection of these domain-specific access policies defines "enterprise-level security" policies. These policies are domain-specific, dynamic and context sensitive to be executed in a traditional way. For example, an online banking application requires for EFT operation to be in an amount limit that is predefined by the user and only between 9:00 am and 5:00 pm.

**Support for consistency of policies across multiple applications:** Enterprise applications are increasingly interconnected to form information enterprise, which consists of many self-contained, heterogeneous and yet integrated application systems. The basic problem of access control in such an environment is to enforce organization wide security policies across these applications. On the other hand, application developers tend to embed access control rules in application systems hard coded which result in costly and error-prone application because there are multiple points of control, every part of application implements their own access policies. With the separation of concern principle [50] "enterprise-level security" policies should be handled with in a uniform, fine-grained and transparent way.

**Support for changes:** Enterprise application policies tend to be changed frequently as business rules changes; hence the access control mechanism that governs enterprise application must easily reflect this dynamic manner. However in traditional information systems, access rules are largely embedded in application systems and as a result it becomes hard to be manageable and reusable.

**Scalability:** Enterprise applications are most likely to be increased in the number of users and integrated application systems; therefore access control mechanism must scale well as business changes.

## 2.4 Web Application Security Vulnerabilities

There is an increasing tendency that web application attacks are becoming dominant over other software security attacks. Confronted with steadily maturing network-layer defenses, attackers are increasingly turning their attention to the application layer and the corresponding business applications that are being served. At the same time, organizations have been increasing their reliance on web applications, in particular to meet the needs of the extended enterprise – that is, the growing population of distributed users. According to statistics regarding web application vulnerabilities shows the growing problem [8].

- From 1Q04 to 1Q05 there has been a 20% rise in the number of application-specific vulnerabilities identified.

- Over 50% of all new vulnerabilities being identified on a weekly basis are attributed to web applications.

- Greater than 80% of all malfunctions that emerged in the past year have focused on exploiting application-layer vulnerabilities (estimate compiled from various sources).

CVE statistics also gives the same result; the most notable trend is the sharp rise in public reports for vulnerabilities that are specific to web applications. These statistics are shown in *Table 1 and 2*. *Table 1* shows, the number of reported security attacks and their types between 2001 and 2006. *Table 2* shows the percentages of these security attacks. These statistics give some important clues about attack trends.

- Buffer overflows (mostly targeting products) were number one year after year, but that changed in 2005 with the rise of web application vulnerabilities, including cross-site scripting (XSS), SQL injection and remote file inclusion. In fact, so far in 2006, buffer overflows are only fourth.

- The increase of percentage of web application attacks is tremendous, by the year 2006, the percentage becomes over %70.

- Even if web application attacks are dominant over other kind of security attacks, it is only the tip of the iceberg since most of the web security incidents are not reported; on the other hand product based security incidents are well-reported.

- As the importance of web application increases, attackers seem to turn their attention to web applications and web applications have become a must for all kind of businesses. Security impacts on web applications have become too risky.

- Although buffer overflows and other product or vendor based attacks can also be a reason for web application attacks. XSS and injection attacks directly aim web applications and web application server products have nothing to do to eliminate these kinds of increasing trends.

- In 2001, the percentage of XSS attacks was below 5%, SQL injection and PHP file inclusion was near 0%. In five years the total percentage of these attacks has increased over 45%.

- Although the total number of incidents has increased fourfold between the years 2001 and 2006, the number of product vendor, network and OS based incidents remained steady. This shows that either the security mechanisms have become more successful in preventing these kinds of attacks or the attackers have turned their attention to web application attacks.

One can easily say that the main boosting factor of web security attacks is turning the spotlight to web applications because of the increase in global e-commerce, increase in global use of valuable information online and increase of global internet use, the web application domain has important contributing factors to this increase in web vulnerabilities, these could be;

- Most of the web application attacks depend on basic data manipulations that are very simple to perform.

- There is a plethora of freely available web applications. Much of the code is alpha or beta, written by inexperienced programmers with easy-to-learn languages such as PHP, and distributed on high-traffic sites. Even some important web applications use these freely distributed web applications. The large number of these applications is probably a major contributor to the overall trends.

- With injection vulnerabilities including XSS, every input has the potential to be an attack vector, which does not occur with other vulnerability types. This leaves more opportunity for a single mistake to occur in a program that otherwise protects against these attacks.

- Apart from usage of freely available web applications, there is an increasing trend to use open source projects for web applications. A fully functional web application can be built in easily by integrating these open source projects together. There is always greater risk for these open source projects, since most of them did not consider security breaches only concentrate on the functionality.

- Web applications tend to have larger attack surfaces than other applications as; there are lots of input entry points to web applications. Most of them have not centric input handling mechanisms, consequently, nearly all web application developers in a project handles user input and even if one entry point is vulnerable, the whole web application will be under great risk.

- Most security professionals concentrate on the network or product security. However applying security patches, integrating antivirus and anti-trojan software are not enough to securing web applications.

**Table 1 Number of security incidents**

| **Security Attacks** | **Total** | **2001** | **2002** | **2003** | **2004** | **2005** | **2006** |
|---|---|---|---|---|---|---|---|
| *Total* | *16192* | *1434* | *2138* | *1173* | *2534* | *4538* | *4375* |
| Cross-site scripting | 2247 | 32 | 187 | 88 | 276 | 725 | 939 |
| Buffer overflow | 2156 | 279 | 433 | 264 | 391 | 445 | 344 |
| SQL injection | 1416 | 6 | 38 | 35 | 140 | 584 | 613 |
| Directory traversal | 764 | 127 | 110 | 34 | 104 | 195 | 194 |
| PHP remote file inclusion | 561 | 1 | 6 | 9 | 36 | 95 | 414 |
| Information leak | 540 | 37 | 89 | 30 | 95 | 175 | 114 |
| DoS caused by malformed input | 463 | 69 | 110 | 29 | 87 | 82 | 86 |
| Symbolic link following | 329 | 64 | 45 | 41 | 72 | 87 | 20 |
| Format string vulnerability | 296 | 46 | 39 | 32 | 61 | 76 | 42 |
| Cryptographic error | 261 | 55 | 58 | 18 | 22 | 68 | 40 |
| Privilege Errors | 233 | 36 | 46 | 12 | 32 | 67 | 40 |
| Metachar injection | 218 | 55 | 56 | 8 | 26 | 59 | 14 |
| Permission Errors | 215 | 39 | 39 | 15 | 24 | 48 | 50 |
| Numeric Errors | 160 | 1 | 8 | 16 | 47 | 36 | 52 |
| DoS caused by flooding | 131 | 29 | 36 | 6 | 31 | 10 | 19 |
| Default or hard-coded password | 125 | 16 | 27 | 2 | 28 | 36 | 16 |
| Weak/bad authentication | 124 | 22 | 27 | 6 | 17 | 21 | 31 |
| Sensitive data under web document root | 88 | 2 | 5 | 3 | 5 | 33 | 40 |
| Form-field Error | 81 | 10 | 17 | 6 | 6 | 19 | 23 |
| Untrusted search path vulnerability | 71 | 12 | 6 | 10 | 14 | 15 | 14 |

**Table 2 Percentage of security incidents**

| **Security Attacks** | **Overall** | **2001** | **2002** | **2003** | **2004** | **2005** | **2006** |
|---|---|---|---|---|---|---|---|
| Cross-site scripting | 13.9% | 2.2% | 8.7% | 7.5% | 10.9% | 16.% | 21.5% |
| Buffer overflow | 13.3% | 19.5% | 20.3% | 22.5% | 15.4% | 9.8% | 7.9% |
| SQL injection | 8.7% | 0.4% | 1.8% | 3.0% | 5.5% | 12.9% | 14.% |
| Directory traversal | 4.7% | 8.9% | 5.1% | 2.9% | 4.1% | 4.3% | 4.4% |
| PHP remote file inclusion | 3.5% | 0.1% | 0.3% | 0.8% | 1.4% | 2.1% | 9.5% |
| Information leak | 3.3% | 2.6% | 4.2% | 2.6% | 3.7% | 3.9% | 2.6% |
| DoS caused by malformed input | 2.9% | 4.8% | 5.1% | 2.5% | 3.4% | 1.8% | 2.0% |
| Symbolic link following | 2.0% | 4.5% | 2.1% | 3.5% | 2.8% | 1.9% | 0.5% |

**Table 2 (continued)**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Format string vulnerability | 1.8% | 3.2% | 1.8% | 2.7% | 2.4% | 1.7% | 1.0% |
| Cryptographic error | 1.6% | 3.8% | 2.7% | 1.5% | 0.9% | 1.5% | 0.9% |
| Privilege Errors | 1.4% | 2.5% | 2.2% | 1.0% | 1.3% | 1.5% | 0.9% |
| Metachar injection | 1.3% | 3.8% | 2.6% | 0.7% | 1.0% | 1.3% | 0.3% |
| Permission Errors | 1.3% | 2.7% | 1.8% | 1.3% | 0.9% | 1.1% | 1.1% |
| Numeric Errors | 1.0% | 0.1% | 0.4% | 1.4% | 1.9% | 0.8% | 1.2% |
| DoS caused by flooding | 0.8% | 2.0% | 1.7% | 0.5% | 1.2% | 0.2% | 0.4% |
| Default or hard-coded password | 0.8% | 1.1% | 1.3% | 0.2% | 1.1% | 0.8% | 0.4% |
| Weak/bad authentication | 0.8% | 1.5% | 1.3% | 0.5% | 0.7% | 0.5% | 0.7% |
| Sensitive data under web document root | 0.5% | 0.1% | 0.2% | 0.3% | 0.2% | 0.7% | 0.9% |
| Form-field Error | 0.5% | 0.7% | 0.8% | 0.5% | 0.2% | 0.4% | 0.5% |
| Untrusted search path vulnerability | 0.4% | 0.8% | 0.3% | 0.9% | 0.6% | 0.3% | 0.3% |

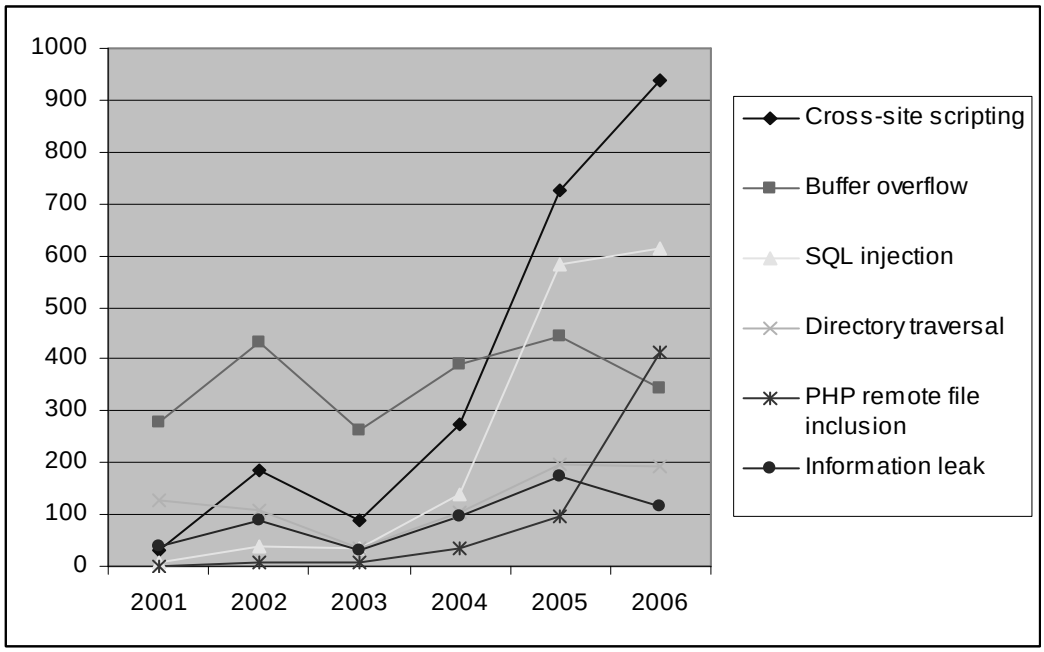


**Figure 7 Top 6 security attacks between 2001 and 2006**

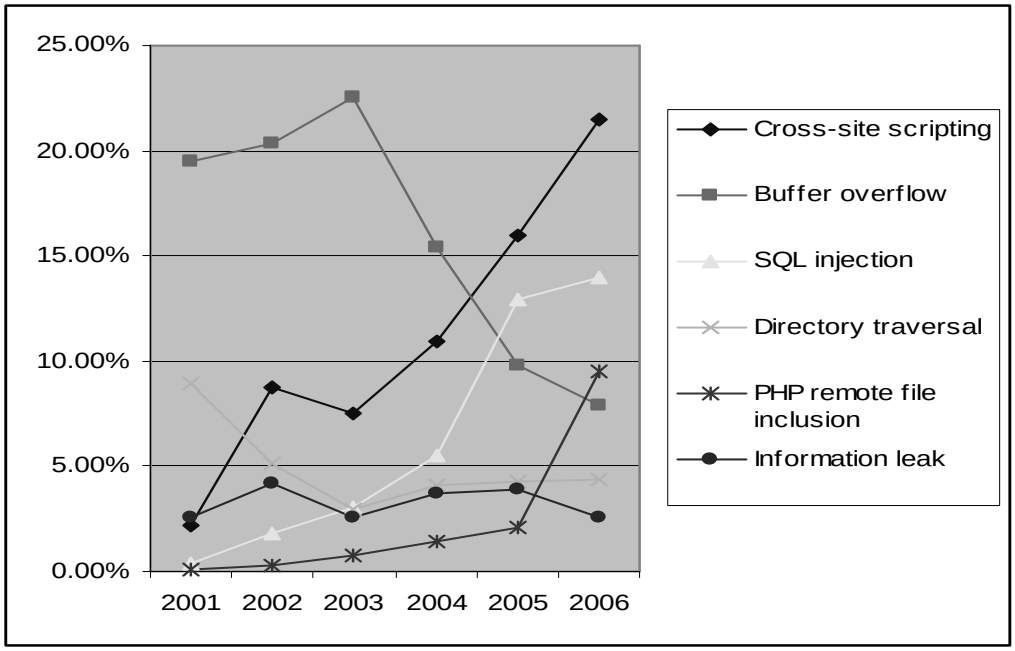


**Figure 8 Percentage of top 6 security attack between 2001 and 2006**

### 2.4.1 Common Vulnerabilities

A typical web application attack executes 5 main scenarios, starting with vulnerabilities scan to launching the attack. The steps are listed below;

**Act 1: The Scan**

The hacker starts by running a port scan to detect the open HTTP and HTTPS ports for each server and retrieving the default page from each open port.

**Act 2: Information Gathering**

The hacker then identifies the type of server running on each port and each page is parsed to find normal links (HTML anchors). This enables the hacker to determine the structure of the site and the logic of the application. Then the attacker analyzes the found pages and checks for comments and other possibly useful bits of data that could refer to files and directories that are not intended for public use.

**Act 3: Testing:**

The hacker goes through a testing process for each of the application scripts or dynamic functions of the application, looking for development errors to enable him to gain further access into the application.

**Act 4: Planning the Attack**

When the hacker has identified every bit of information that can be gathered by passive (undetectable) means, he selects and deploys attacks. These attacks center on the information gained from the passive information gathering process.

**Act 5: Launching the Attack**

After all of these procedures, the hacker engages in open warfare by attacking each Web application that he identified as vulnerable during the initial review of the site.

The Open Web Application Security Project (OWASP) [52] is one of the foundations that is dedicated to find and classify possible web application attacks and offers countermeasures for them. OWASP publishes "Top Ten Most Critical Web Application Security Vulnerabilities" list to inform the public about the most dangerous vulnerabilities. The Top Ten list is generated according to data accumulated by MITRE's [53] vulnerability trend list consisting of CVE's [11] data. According to "Top Ten Most Critical Web Application Security Vulnerabilities" list published in 2007; Cross-side scripting (XSS), Injection Flaws, Malicious File Execution, Insecure Direct Object Reference, Cross Site Request Forgery (CSRF), Information Leakage and Improper Error Handling, Broken Authentication and Session Management, Insecure Communications and Failure to Restrict URL Access. The occurrence percentages of these vulnerabilities according to MITRE's date are given in *figure* 9.

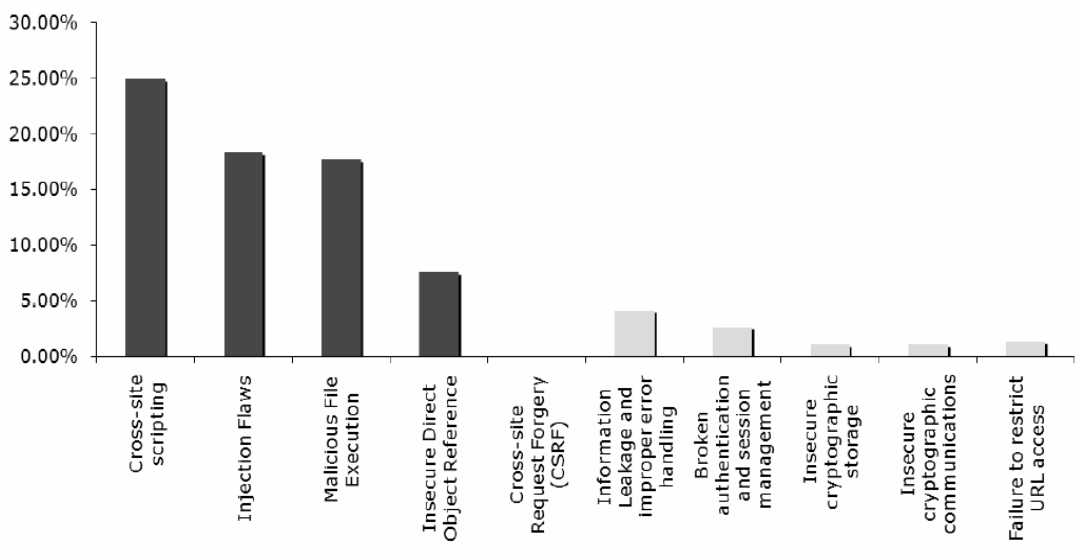


**Figure 9 Percentage of Vulnerabilities (2007)**

### 2.4.1.1 Cross Site Scripting (XSS) Attacks

A web application is vulnerable to XSS attacks when they allow injection of malicious scripts as inputs of user and as a result of generating dynamic pages from this infected input, these malicious scripts could be executed from client browsers and could affect all web site clients. Although secure execution of JavaScript code is based on a sandboxing mechanism, which allows the code to perform a restricted set of operations only and JavaScript programs downloaded from different sites are protected from each other using a compartmentalizing mechanism, called the same-origin policy, scripts may be confined by the sand-boxing mechanisms and conform to the same-origin policy, but still violate the security of a system. This can be achieved when a user is lured into downloading malicious JavaScript code (previously created by an attacker) from a trusted web site.

Two main classes of XSS attacks exist: stored attacks and reflected attacks. In a stored XSS attack, the malicious JavaScript code is permanently stored on the target server (e.g., in a database, in a message forum, in a guestbook, etc.). In a reflected XSS attack, on the other hand, the injected code is “reflected” off the web server such as in an error message or a search result that may include some or all of the input sent to the server as part of the request. Reflected XSS attacks are delivered to the victims via e-mail messages or links embedded on other web pages. When a user clicks on a malicious link or submits a specially crafted form, the injected code travels to the vulnerable web application and is reflected back to the victim’s browser

#### 2.4.1.2 Injection Flaws

Injection Flaws are one of the most common web application vulnerabilities and consist of various attack techniques such as SQL, LDAP and XML injection. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. Without validation, the attacker can easily manipulate command or query with insertion of special characters and command. SQL injection can be given as an example of injection flaws. SQL injection attacks are one of the most dangerous instantiation of injection attacks. In this attack technique malicious SQL commands are injected into request parameters in order to affect the execution of predefined SQL commands. SQL injection attacks threats most of the subjects of computer security;

**Confidentiality:** Most common consequence of SQL injection attacks is loss of confidentiality. Since SQL databases hold sensitive data, unauthorized access to these data could generate more dangerous consequences.

**Authentication:** Most of the applications use SQL databases for storing authentication data. If a SQL injection occurs in the authentication part of the system, the attacker can bypass all authentication mechanisms.

**Authorization:** Authorization modules that use the SQL database are another critical part of the web application. If they are vulnerable to SQL injection attacks, it would be possible to change authorization information and a security breach can be opened for an application.

**Integrity**: By SQL injection, it is also possible to make changes or deletions that threats integrity of whole database.

#### 2.4.1.3 Malicious File Execution

Application developers will often directly use input and stream file functions that come from the user directions. Without necessary checks, an attacker can manipulate the application to execute malicious commands and files. Code injection can be studied as an example of this kind of attack. In code injection, the application allows inputs to be fed directly into an output file that is later processed as code. Different from XSS or HTML injection techniques which is executed on the client side, direct static code injection vulnerability enables malicious codes to be executed at server side but this can result from XSS or HTML injection as the same special characters can be involved. One example of direct static code injection is Server-Side Includes (SSI) injection, which is a server-side exploit technique that allows an attacker to send code into a web application, which will later be executed locally by the web server. SSI Injection exploits a web application's failure to sanitize user-supplied data before they are inserted into a server-side interpreted HTML file. Before serving an HTML web page, a web server may parse and execute Server-side Include statements before providing it to the user. In some cases (e.g. message boards, guest books, or content management systems), a web

application will insert user-supplied data into the source of a web page. If an attacker submits a Server-side Include statement, he may have the ability to execute arbitrary operating system commands or include a restricted file's contents the next time the page is served

#### 2.4.1.4 Insecure Direct Object Reference

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. An attacker can manipulate direct object references to access other objects without authorization. Path manipulation attack can be an example of this kind of attack. This attack technique involves adding special characters in file and directory names. These manipulations are intended to generate multiple names and therefore multiple access points for the same object. Just like path traversal attacks, path equivalence attacks also threaten disclosure of information. If any application restricts directory access programmatically, these restrictions can be bypassed by adding special characters in requested file or directory. Thus, application might fail to parse requested URL and misinterpret the request. Path equivalence attacks can also used for bypassing security restrictions depends on black list. Consider an example of an application that allows uploading and a black list to eliminate malicious file formats such as symbolic links. An attacker can bypass this black list check by adding trailing dots to extension of a file, allowing him to traverse to the target file or directory. When an attacker collects enough information about the application using path traversal and path equivalence attacks then he could plan new attacks to break into the application.

#### 2.4.1.5 Cross-Site Request Forgery (Session Riding)

Cross-Site Request Forgery is about forcing an unknowing user to execute unwanted actions on a web application in which he is currently authenticated. CSRF is an attack that tricks the victim into loading a page that contains a malicious request. It is malicious in the sense that it inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf, like changing the victim's e-mail address, home address, or password, or making a purchase. CSRF attacks target functions that cause a state change on the server.

CSRF works like XSS attack: An attacker identifies a URL on a Website that initiates typical Web functions such as making a purchase, changing an email address or transferring funds and takes that URL and loads it to a web page he controls with malicious code injected to be executed later.

#### 2.4.1.6 Information Leakage and Improper Error Handling

A system information leak occurs when system data or debugging information leaves the program through an output stream or logging function. An attacker can cause errors to occur by submitting unusual requests to the web application. The response to these errors can reveal detailed system information, deny service, and cause security mechanisms to fail or crash the server.

#### 2.4.1.7 Broken Authentication and Session Management

Without using proper authentication and session management techniques, an attacker can hijack user or administrative accounts, bypass authorization controls and cause privacy violations. Session hijacking is a typical example of such attacks. Using session hijacking attack, the attacker tries to take control of a user session by obtaining or generating an authentication session ID. Session hijacking involves an attacker using captured, brute forced or reverse-engineered session IDs to seize control of a legitimate user's session while that session is still in progress. In most applications, after successfully hijacking a session, the attacker gains complete access to all of the user's data, and is permitted to perform operations instead of the user whose session was hijacked.

#### 2.4.1.8 Insecure Cryptographic Storage

Protecting sensitive data using cryptography is a common technique for web applications, however applications frequently uses poorly designed cryptography either using unproven algorithms or improper implementation of strong algorithms.

#### 2.4.1.9 Insecure Communication

Transferring sensitive data on an unsecured channel could cause stealing of private information. Sniffing application traffic can be given as an example. Sniffing application traffic simply means that the attacker is able to view network traffic and will try to steal credentials, confidential information, or other sensitive data. Anyone with physical access to the network is able to sniff the traffic. Also, anyone with access to intermediate routers, firewalls, proxies, servers, or other networking gear may be able to see the traffic as well. By sniffing application traffic, an attacker gain sensitive information about the web site. If this communication is not protected, the attacker can reveal user cookies, session id, user id and password that can be used to generate other attacks later.

#### 2.4.1.10 Failure to Restrict URL Access

Frequently, the only protection for a URL is that links to that page are not presented to unauthorized users. However, a motivated, skilled, or just plain lucky attacker may be able to find and access these pages, invoke functions, and view data. Path traversal is typical example; this attack technique involves providing relative or absolute path information as a part of request information. Such attacks try to access files that are normally not accessible by anyone and if such a request is received, it must be denied. This attack risks information disclosure of systems. Although it does not directly threaten the integrity of the system, the attacker can gain access to sensitive data such as password and configuration files and by using it, he can do more dangerous attacks to the system.

Apart from these techniques, there are numerous number of different types of attacks, A detailed list (consisting 58 different vulnerabilities) and description of web application vulnerabilities with classification according to Plover taxonomy described in section 2.1.4 are given in Appendix-A.

## 2.5 Related Works

As discussed in section 1.1, the scope of this thesis covers two important problems; one is encapsulating domain specific factors in access control and the other is web application security vulnerability. Encapsulating domain specific factors in access control is not new concept and has been investigated in several researches. One of the earliest examples can be found in OSI access control framework [54] published in 1994. From then on, both academic researchers and various distributed application systems vendors have tried to encapsulate domain-specific factors. On the other hand, confronting web application security vulnerabilities is considerably new and mainly handled by software security vendors. The solution is called, web application firewalls. According to web application security consortium (WASC) [23] a web application firewall is *"An intermediary device, sitting between a web-client and a web server, analyzing OSI Layer-7 messages for violations in the programmed security policy. A web application firewall is used as a security device protecting the web server from attack."*

### 2.5.1 Approaches to Encapsulate Domain Specific Factors

These approaches can be classified into three categories;

**Middleware infrastructures:** Most common distributed application technologies, such as J2EE [4], .NET [55], DCOM [3], JAAS [56] and CORBA [2] has integrated access control engines. These middleware technologies has been deeply discussed and compared in Beznosov's works [5] [43] [57] [58].

CORBA Security service (CS) [2] defines interfaces to a collection of objects to enforce a range of security policies. It provides abstraction from an underlying security technology so that CORBA-based applications can be independent from the particular security infrastructure provided by the user environment. This generality makes CS free from any particular access control model. Instead, it could be configured to support various access control models.

Access control in Java Authentication and Authorization Service (JAAS) is enforced only on system resources, such as files, sockets, etc., but not on Java objects and other application resources. JAAS has very generic and extensible support for different privilege attributes that can be easily defined via new classes. The security basics depend on code bases of Java classes, the identity of the code signer and the value of the subject privilege attribute. These attributes are all passed to JAAS via Policy class interface for authorization decisions.

The security model of DCOM is based on access control lists (ACL) to code authorization policies. DCOM provides DCOM Security API to enforce policies outside of objects with the presence of process and host-specific policies. DCOM defines component-specific policy where there is no distinction among different objects and their methods in the same OS process and host-wide policy to define interaction of object within the same host. Although component- and host-wide policies can be used to implement a fine grain access control in an application-specific way, application-specific policies cannot be enforced and only security-related attributes of subjects and objects can serve as input for external access control mechanisms.

The core of J2EE depends on EJB security architecture where each EJB or each method of EJB can be mapped to an allowed role and users can be assigned to a role depending on RBAC fashion. EJB security allows any user attributes to be reduced to roles and so that the domain specific rules can be evaluated. However EJB security fails to be fine-grained, the only resources of the system are EJB's methods no other abstraction can be possible.

The main purpose of all these technologies is to control object interactions within an organization-wide, uniform and transparent way. However they all fail their expressiveness and granularity when we consider enterprise applications. Enterprise applications consist of business transactions and business services that require much more abstraction to be controlled by object interaction access control.

**Access control frameworks:** A sizeable amount of research has been conducted on access control frameworks [59-62]. The main idea of these frameworks is to supply a uniform access control interface that requests access permissions from the centralized authorization engine. Authorization engines are able to interpret and execute enterprise-policy rules that are defined policy specification languages such as Ponder [63] and eXtensible Access Control Markup Language (XACML) [62].

Ponder tries to define a common declarative, object-oriented language that will provide a unified approach to specify security and management policies for distributed object systems. It enables non-discretionary access control where administrators have the authority to specify security policies that are enforced by the access control system. Ponder supports access control by providing authorization, delegation, and information filtering and refrain policies. These policies can be made up of composite policies to facilitate policy management in large, complex enterprises. They provide the ability to group policies and structure them to reflect organizational structure. Users can be assigned to roles and groups as in RBAC. XACML is development effort of a standard access control policy language that enables the use of arbitrary attributes in policies to encapsulate domain-specific factor, role-based access control and dynamic policies to reflect required changes to the applications. Some important terms that differs XACML from other access control languages are; XACML specifies an "Access Control Decision Function" (ADF), and defines its interactions with an "Access Control Enforcement Point" (AEF) so provides differentiation of ADF from AEF. XACML defines a "Policy Decision

Point" (PDP), and defines its interactions with a "Policy Enforcement Point" (PEP) so not only provides a framework for policies but as well as a language.

These frameworks are powerful for expressiveness in enterprise-level security policies, however they are not transparent. It is the developer's duty to ask for authorization request whenever required; if the developer misinterprets the policy or forgets to ask for authorization then there will be no access control on sensible data. Thus, it is hard to be organization-wide and avoid being error-prone. To guarantee access control over the whole application, one method is code weaving using aspect oriented languages [51, 64]. In these methods, source codes are weaved to use access control frameworks. This guarantees that application layer is weaved to be under access control, however the presentation layer of enterprise web application is still open to unauthorized request.

**Commercial Access Managers:** Most of the commercial application server venders have access manager's products such as BEA WebLogic [65], Oracle [65], and IBM WebSphere [65]. These access managers have also the capability to integrate into other application servers. There are also other vendor's product that can integrate into variety of application servers such as AssureAccess [68] and WebDeamon [69]. The common strategy of these products is managing user identities and roles assigned to appropriate privileges. They control access over the presentation layer, control web resources, however apart from BEA WebLogic Enterprise Security; they all suffer from supporting domain-specific access control policies [5]. All of them uses RBAC [26] method, however RBAC fails to separate enforcement function and decision function that is needed to evaluate domain-specific access policies [5]. WebLogic Enterprise Security uses a somewhat different strategy, although it also uses RBAC to manage user identities and roles, it introduces policy evaluators that can also control application-layer of web applications by evaluating request attributes.

### 2.5.2 Web Application Firewalls

Nowadays there are both academic proposals for web application firewalls [70], as well as open-source [71] and commercial ones [72] [73]. David Scott and Richard Sharp [70] propose a Security Gateway in front of the application and web servers to validate and transform client request. They construct a *Security Policy Description Language* (SPDL) to specify a set of validation constraints and transformation rules. With in these rules, a security officer or developers can define parameter names; maximum and minimum length of parameter values and appends a MAC code for security-critical hidden-form parameters to prevent users from modifying data. Upon reception of a client request by security gateway the request parameters are checked according to the rules defined in SPDL.

ModSecurity [71] is a fully open source web application firewall that is designed as a module of Apache Server. It implements the ModSecurity Rule Language and policy rule evaluator to work with HTTP transaction data. ModSecurity also provides a core set of rules to detect violations of the HTTP

protocol and a locally defined usage policy. These core sets are designed in a way that they provide protection from common web attacks, automation detection, Trojan protection and error hiding. Security officers or developers can customize the behavior of ModSecurity by adding validation rules for request parameters. ModSecurity divide the execution of HTTP requests into 5 phases; request headers, request body, response header, response body and logging. The core rules and validation rules can be attached to any of these phases.

Traffic Shield [72] is a commercial web application product implemented by the vendor *f5*. Its countermeasure against web application attacks is so called application flow model which is in fact a detailed model (or policy) of the ways users interact with the application. The product learns the allowed operations of the application by analyzing the incoming and outgoing traffic and tailors its model accordingly. For each web page presented to the user, the model describes the structure of the HTTP or HTTPS requests that are generated by the client side source code of the Web page and the authorized transitions to other Web pages. The model, or policy, can be built using only a few key factors (in order to minimize complexity) or using very detailed descriptions (in order to increase granularity) or anywhere in between depending on the desired security posture of the application.

Secure Sphere [73] from Imperva is a full compact product that has network firewall, intrusion prevention systems (IPS), intrusion detection systems and a built-in web application firewall. Like Traffic Shield, Secure Sphere has an automated process called *Dynamic Profiling* that examines live traffic to create a model of application structure and dynamics. It also allows manual tailoring of its model. The main difference of Secure Sphere from Traffic Shield is that Secure Sphere can work on passive mode as well as inline mode; on the other hand traffic shield only operates as inline mode. In passive mode the flow of web traffic is not intercepted but analyzed using sniffing and if any malicious request is detected, it will send TCP reset message. However in inline mode the firewall acts like an active device such as bridge, router or reverse proxy, intercepts coming connections and control the flow of information.

In either case, all web application firewall products obey the WASC's definition of web application firewalls. They are all installed in DMZ before web applications as a separate traffic. They work mostly on inline mode, intercepting coming traffic; analyze OSI Layer-7 messages for violations in the programmed security policy. Most of them have learning capability that by investigating web traffic of the application, they can adapt themselves by figuring out legal operations and construct a positive security model from these. They also have a negative security model which especially targets web application attacks. Like most the anti-virus products, they can upgrade and patches themselves for new kinds of attacks by connecting and fetching attack signatures from the product's main servers.

# CHAPTER 3

# ACCESS CONTROL AND SECURITY SOLUTION BASED ON RAD

In this chapter, the details of our proposed model to address the problems of access control and application security are described. Our solution EYEKS ("Erişim, YEtkilendirme ve Kişiselleştirme Sistemi" in Turkish, meaning "Access, Authorization and Personalization System") brings together encapsulating domain specific factors in access control and confronting application security vulnerabilities in enterprise web applications. EYEKS provides a modular access control service that can decides on application-specific policies that are required by the enterprise application's complex business logic as well as policies that controls web application security vulnerabilities.

EYEKS uses CSAAS as authorization engine which is in fact Resource Access Decision (RAD) implementation with additional RBAC [26] capabilities which was presented in Akademik Bilişim Conference 2005 [6]. RAD has been chosen as the access decision mechanism since this facility is one of the best solutions that can be used by security-aware applications as described in Beznosov's work [5]. The implementation details of RAD facility can be found in section 3.1. On the other hand, as it will be shown in section 3.2, web applications suit well to be controlled using RAD specification.

As mentioned in section 2.2, the best way to encapsulate application specific policies is to separate access decision mechanisms from the application itself and leaves enforcement to the application; EYEKS is designed to be a separate layer that can be integrated into any n-tier enterprise web application as a first layer that guarantees application wide enforcement. This, so called Application Security Layer, is installed in the Demilitarized Zone (DMZ) sits between external network and organization's internal network to increase the security (described in section 3.3.1). EYEKS intercepts user requests, checks violations of both enterprise policies and application security policies, authorize the request and using HTTP tunneling described in section 3.3.5 proxying the backend enterprise web application.

EYEKS is designed as a chained structure so that user requests are processed by traversing possible chain of execution. Possible operations can be added, removed, arranged according to application security requirements. Chain elements are allowed to extract information from user request, check for authentication and authorization, create or change HTTP session. Briefly it allows all kind of manipulation to user request and response. EYEKS also provides a powerful API, so that application developers can easily integrate their required chain operations into the system. The details of chained structure are given in section 3.3.2.

EYEKS adds session management capabilities apart from HTTP session. User requests are authorized and managed according to EYEKS session (section 3.3.4). A successful authentication creates EYEKS session and EYEKS guaranties this session is carried out through all user operations. EYEKS session can be based on header based or cookie based encrypted tokens. This session management facility also allows Single-Sign-On feature for the whole enterprise application.

EYEKS consult to access decision mechanism many times during execution of user request. This mechanism can be managed using RAD implementation to add new policies that must be satisfied during request. Both enterprise policies and application security policies can be controlled using single application. This architecture provides EYEKS to be dynamic so that it reflects any policy changes at the time of execution. Policy execution details are given in section 3.4.

The whole application security layer (EYEKS) can be built on any J2EE based application server. Alternatively, EYEKS can be executed as a stand-alone application without the need for a server, to be free from any security breaches of application servers. Integration issues will be covered in section 3.5.

In the following sections, the architecture of EYEKS will be described in detail. Then a verification of the solution will be introduced

## 3.1 RAD Implementation (CSAAS)

### 3.1.1 CSAAS Architecture

The overall architecture of CSAAS is given in *figure 10.* As shown, CSAAS consist of 4 main components, which are installed on different sites, and 8 sub-components within.

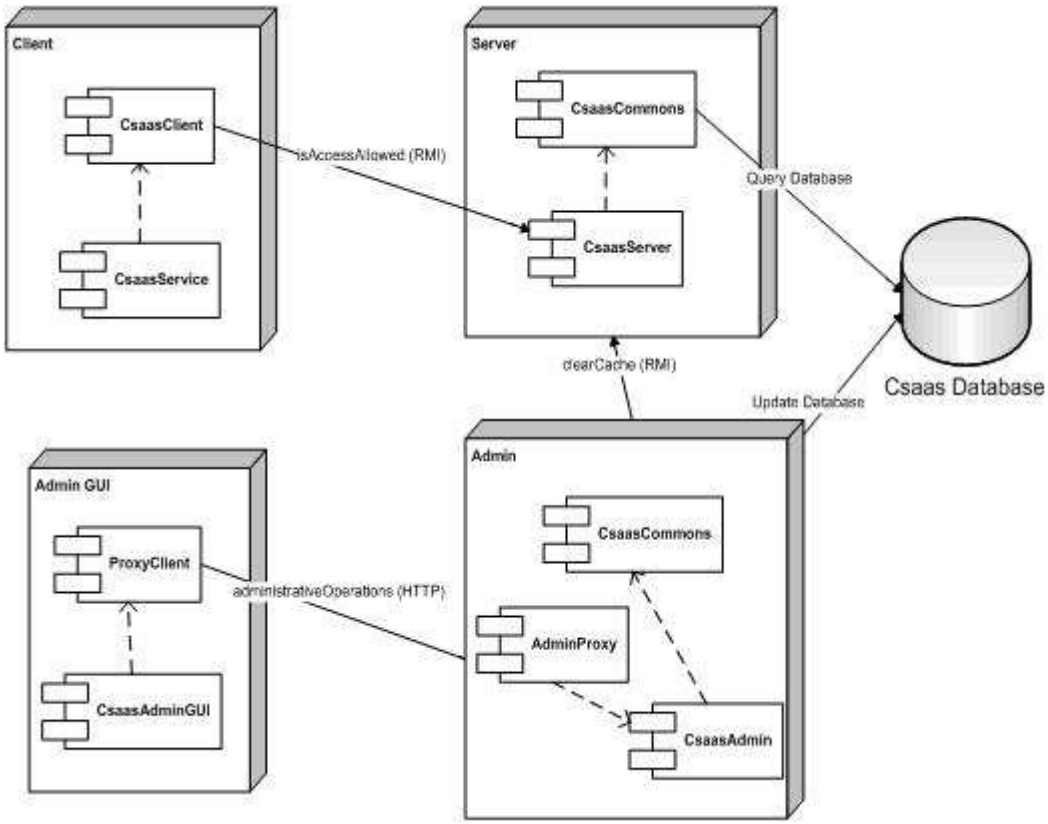


**Figure 10 CSAAS Architecture**

**Client:** This component is in fact not a process, but a library that is an interface between CSAAS Server component and application itself. In EYEKS, application security layer uses this library to ask access decisions to CSAAS Server. This component consists of 2 sub-components; CSAAS Client, which handles remote method invocation to CSAAS Server and CSAAS Service, is a wrapper of CSAAS Client and provides interface between the application program (Application security layer) and CSAAS Server.

**Server:** This component is a core part of CSAAS and works as a stand-alone process. Server component consists of two sub-components; CSAAS Server and CSAAS Commons. CSAAS Server is the implementation of RAD specification and responsible for deciding on authorization requests. A remote object, IRMICsaasInterface, is registered to RMI registry which provides access from CSAAS Clients. CSAAS Commons is a utility library and is used by both CSAAS Server and CSAAS Admin components. CSAAS Commons handles object to relational mapping of CSAAS objects and mainly responsible for querying and updating the CSAAS database. It also provides cache management of database objects that improves performance. The interface between CSAAS Client and CSAAS Server is shown in figure 11.

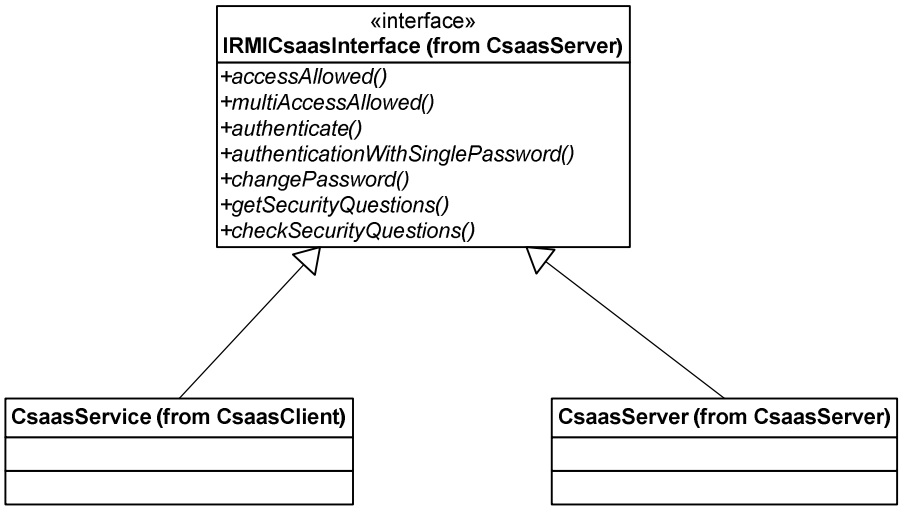


**Figure 11 CSAAS Server Interfaces**

**Admin:** This component implements management functionalities of CSAAS such as managing operations, resources, user groups, policies, dynamic attributes, policy evaluators and decision combinators. Because of security concerns, no direct access between client computers and Csaas network is allowed. Admin component also implements a proxy service that serves to Admin GUI component. Admin component consist of three sub-components; Admin Proxy provides necessary service proxies that bridge Admin GUI to CSAAS Admin. Admin Proxy implements a single action servlet that can be deployed on any J2EE supported application server such as Tomcat and receives any commands coming from Proxy Client sub-component of Admin GUI and invokes necessary operations from CSAAS Admin. CSAAS Admin is core implementation of CSAAS management and provides business operations. CSAAS Admin sends *"clear cache"* message to CSAAS Server whenever an update operation is done on CSAAS Server components described in section 3.1.2. As in Server component, CSAAS Component is also used for handling object to relational mapping of CSAAS objects and mainly responsible for querying and updating the CSAAS database.

**Admin GUI:** This package provides user interface to manage CSAAS functionalities. CSAAS Admin GUI and Proxy Client are two sub-components of Admin GUI. CSAAS Admin GUI provides graphical user interfaces based on Java Swing components. Admin GUI uses Proxy Client to access CSAAS Admin and carry out the administrative operations. Proxy Client provides service proxies that bridges Admin GUI to CSAAS Admin by inserting method name and method arguments that will be invoked on CSAAS Admin into HTTP request parameters. Whenever Admin Proxy receives such messages, it will unpack the messages and invoke corresponding method with given arguments and send the response back containing the result. The collaboration diagram of the communication

between Admin GUI and CSAAS Admin and also between CSAAS Admin and CSAAS Server is shown in *figure 12*.

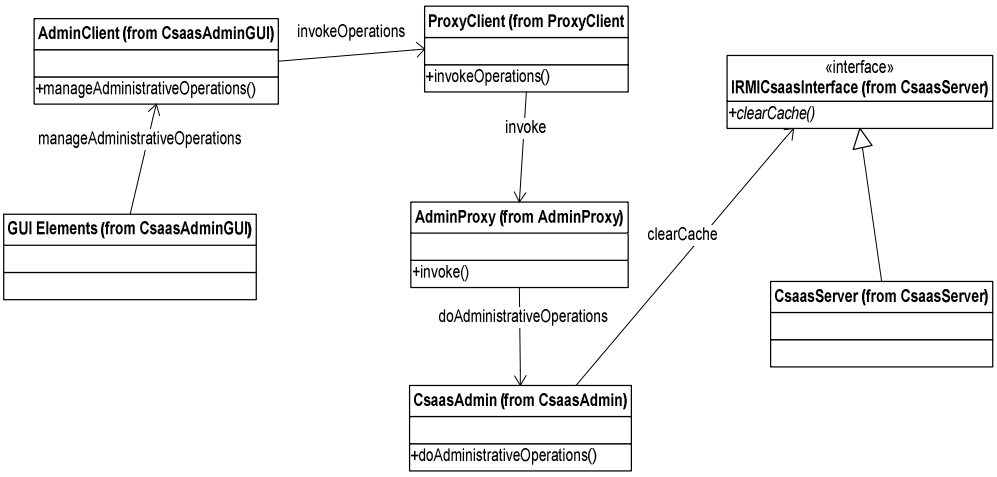


**Figure 12 Interactions of Admin Components**

### 3.1.2 Components of CSAAS Server

CSAAS Server is composed of following components as specified in OMG specification [25] and Beznosov's work [74]. All components and their interactions are well defined in OMG's specification and here the details are omitted. The relationships among these components are given in *figure 13*.

1. Access Decision Object (ADO)
2. Policy Evaluator Locator (PEL)
3. Dynamic Attribute Service (DAS)
4. Decision Combinator (DC)
5. Policy Evaluator (PE).

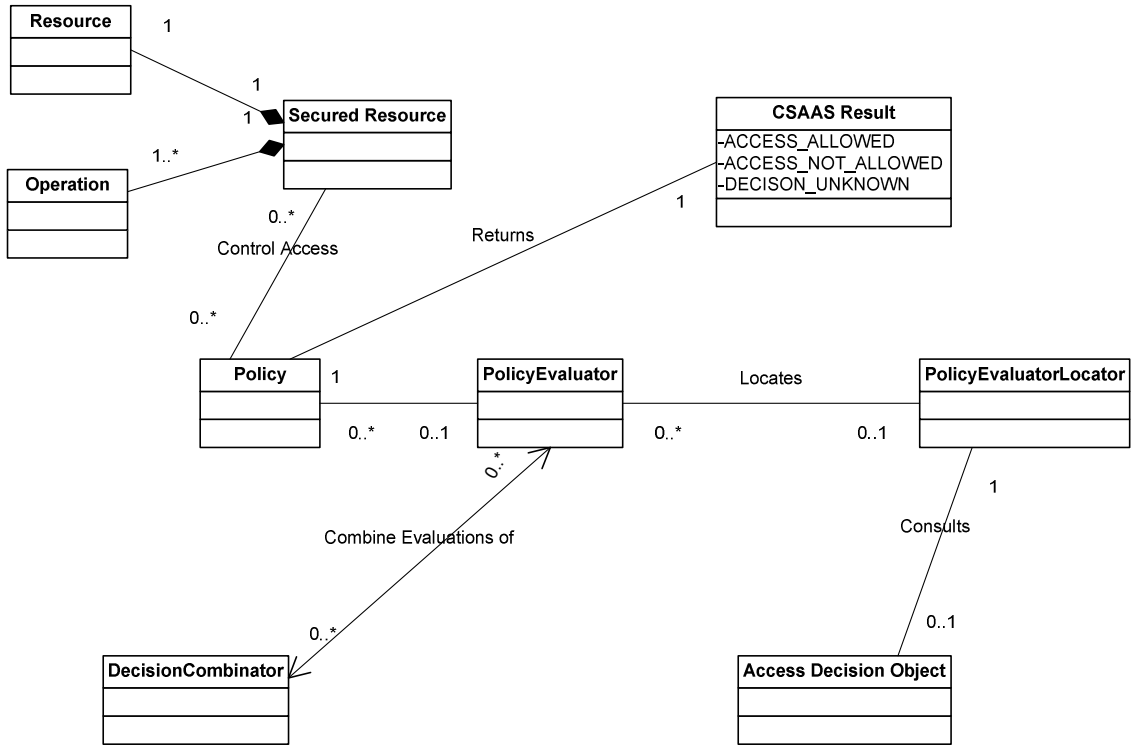


**Figure 13 Components of CSAAS Server**

Applications (especially in this case Application Security Layer) interact with CSAAS Server only through the Access Decision Object (ADO). ADO provides a single, uniform interface to the clients and other CSAAS interfaces. Whenever the ADO receives an authorization decision request, it consults Policy Evaluator Locator (PEL) object which decides what Decision Combinator (DC) and Policies Evaluators (PE) to be used. A PEL maintains mapping of resources to DC's and PE's. A secured resource access can be controlled by zero or more access control policies. Policy Evaluator (PE) is responsible for the evaluation of such policies. PE evaluates and returns a grant or denial of access, when the attributes of access result is not enough to evaluate an access, PE returns DECISION_UNKNOWN. There is a one-to-many relation from PE objects to policies and many-to-many relation from policies to resources. Because as a PE object can evaluate one or more policies for a given resource, policies associated with a resource don't have to be evaluated by a single PE object. Evaluation decision can be distributed among several PE objects.

The results of the access policies for a given resource can be combined under Decision Combinator (DC) object to combine all evaluations into an authorization decision which is sent back to client. A DC object is controlled by a combination policy which is in fact business logic under that DC. DC can

be added to the CSAAS Server using different implementations such as basic logical operation like AND, OR or complex combinators like hierarchies of PE objects where a decision from a higher level PE can override decisions from lower-level PE objects.

To evaluate an access policy, a PE needs security attributes that come with authorization request. PE objects use these security attributes as criteria for evaluating access control policies. The security attributes can contain both static and dynamic attributes. Static attributes represent the characteristic of the principal (such as user name, user role) or business logic (such as amount to be transferred of an EFT operation). Static attributes are supplied by the client and used without alteration. On the other hand, a dynamic attribute can only be determined at the time an access request comes and evaluated by CSAAS itself. Dynamic attributes most probably denote relationships between a principal and a resource, which also reflects business rules of the application. Whenever PE asks for a dynamic attribute to ADO, ADO delegates the discovery of dynamic attribute to Dynamic Attribute Server, which locates and finds the value of the attribute. The location of dynamic attributes can be database, other process or another object within the same virtual machine.

### 3.1.3 Execution Flow

Authorization decisions are computed though a sequence of operations carried out the CSAAS Server components. The sequence is triggered by an accessAllowed message coming from application system to ADO object. ADO object executes the flow and returns the result back to the system. The execution sequence can be found in *figure 14* and is described below;

1. An application server (AS for short) contacts the ADO server for an authorization decision to perform an operation on a resource by a principal with a list of security attributes.

2. The ADO object requests the PEL object to locate necessary DC and PEs associated with the resource.

3. The PEL returns to the ADO a reference to a DC and a set with zero or more references to PE objects.

4. The ADO requests the DAS for any dynamic attributes to evaluate dynamic attribute value.

5. The DAS returns to the ADO a set of dynamic attributes with their values to be used in obtaining an authorization decision. The DAS can add dynamic attributes or remove existing attributes from set.

6. The ADO sends to the DC a set of PE servers for evaluation of policies that control access to the resource.

7. The DC requests each PE in to authorize or deny the operation on the resource given the security attributes of the principal.

8. Each PE in evaluates zero or more access policies associated with the resource and sends back the result to DC.

9. The DC combines all replies from all PE and combines them into a single grant or denies response. This response, the authorization decision, is returned to the ADO server.

10. 10. The ADO returns the authorization decision from the DC server to the application system.

11. The application system (here Application Security Layer) receives the authorization decision from the ADO server and enforces it.

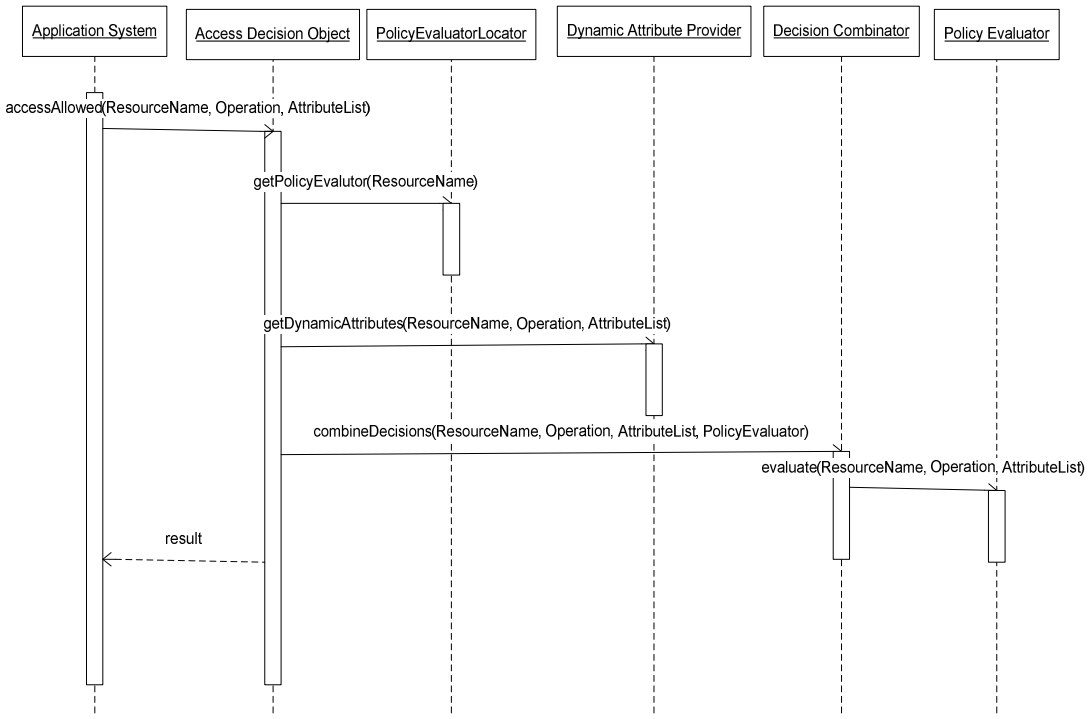


**Figure 14 Sequence Diagram of Access Decision**

### 3.1.4 Limitations and Improvements

CSAAS was implemented with some limitations and improvements over RAD specification. OMG specifies ADO, PE, DC, PEL and DAS as distributed objects so that any ADO, PE, DC object can be distributed among any RAD implementation that can accessed during execution but to simplify the design CSAAS only implements *IRMICsaasInterface* object as distributed which is a wrapper of ADO object. However PE, DC and DAS objects are invoked using reflection mechanism of Java so that any policy evaluator, decision combinatory, dynamic attribute service implementation can be added to CSAAS without altering the binaries of the implementation.

On the other hand, many improvements are made over RAD specification. These are;

**RBAC:** Since nearly all enterprise applications needs rule-based access control, CSAAS implements RBAC policy evaluator, which adds role-based, access capability to RAD.

**Cache Management:** To improve the performance cache management is also added whenever PEL locates policy evaluators or DAS locates dynamic attribute service corresponding data (for example, resource, operation pair, need policy evaluators, decision combinators) are fetched from database and inserted into the cache.

**Logging:** A logging mechanism is also added. CSAAS logs the time of access request decision, which operation is trying to be executed on which resource, what the decision is and how the decision is evaluated so that any malicious request can be tracked.

**Warnings:** On some very important and sensitive resources (such as banking accounts) or any incoming access requests carrying out by some users (for example, login operation of a suspicious user) can be reported as a warning message. Warning messages can be configured as to be inserted to database or to be sent as a TCP message to a specific location so that security administrations can monitor sensitive operations.

**Implemented Dynamic Attribute Providers:** To be evaluated by Dynamic Attribute Service, Java dynamic attribute provider (JDAP) and SQL dynamic attribute provider (SQLDAP) are implemented. JDAP enables any java class to be added to CSAAS on run time to evaluate dynamic attributes. Hence, this gives application developers the power to implement any kind of dynamic attribute evaluations. SQLDAP can be configured to connect any database and execute SQL statements that can be used as dynamic attribute.

**Implemented Decision Combinators:** Predefined basic logical operations such as AND, OR as well as AND over OR *(policy1 OR policy2) AND … (policyN OR policyN+1), OR* over *AND (policy1 AND policy2) OR … (policyN AND policyN+1),* First Couple AND then OR *(policy1 AND policy2) OR … policyN OR policyN+1*, First Couple OR then AND OR *(policy1 OR policy2) AND … policyN AND policyN+1* are implemented to combine policy evaluators.

**Implemented Policy Evaluators:** Most popular policy evaluators such as Java policy evaluator (JPE), JavaScript policy evaluator (JSPE), rule policy evaluator (RPE) and RBAC policy evaluator (RBAC) are implemented. RBAC evaluates access according to role-based access decision; JPE gives the application system developers the power to evaluate policies by java class. JSPE enables java script to be written to evaluate results and though RPE, security administrators can write basic logical rules to evaluator's policies.

## 3.2 Mapping Polices to CSAAS

As mentioned in section 2.2.6, RAD specification requires resources and their valid operations to be well defined. Therefore, the first step before mapping the policies is to define resources and their operations. RAD allows any granularity level of resources, so there is no common way as to how resources should be named. Depending on the structure of enterprise web application, security officer must decide what kind of resources and their possible operations will be defined. Although every enterprise application requires a distinct way of resource naming, some common patterns can be defined.

For basic kind of web applications, that does not build on any Model View Controller architecture. (For example web application consist of a number of jsp pages that manages view, control and model altogether as given in *figure 15*) the possible resource and operation definitions may be;

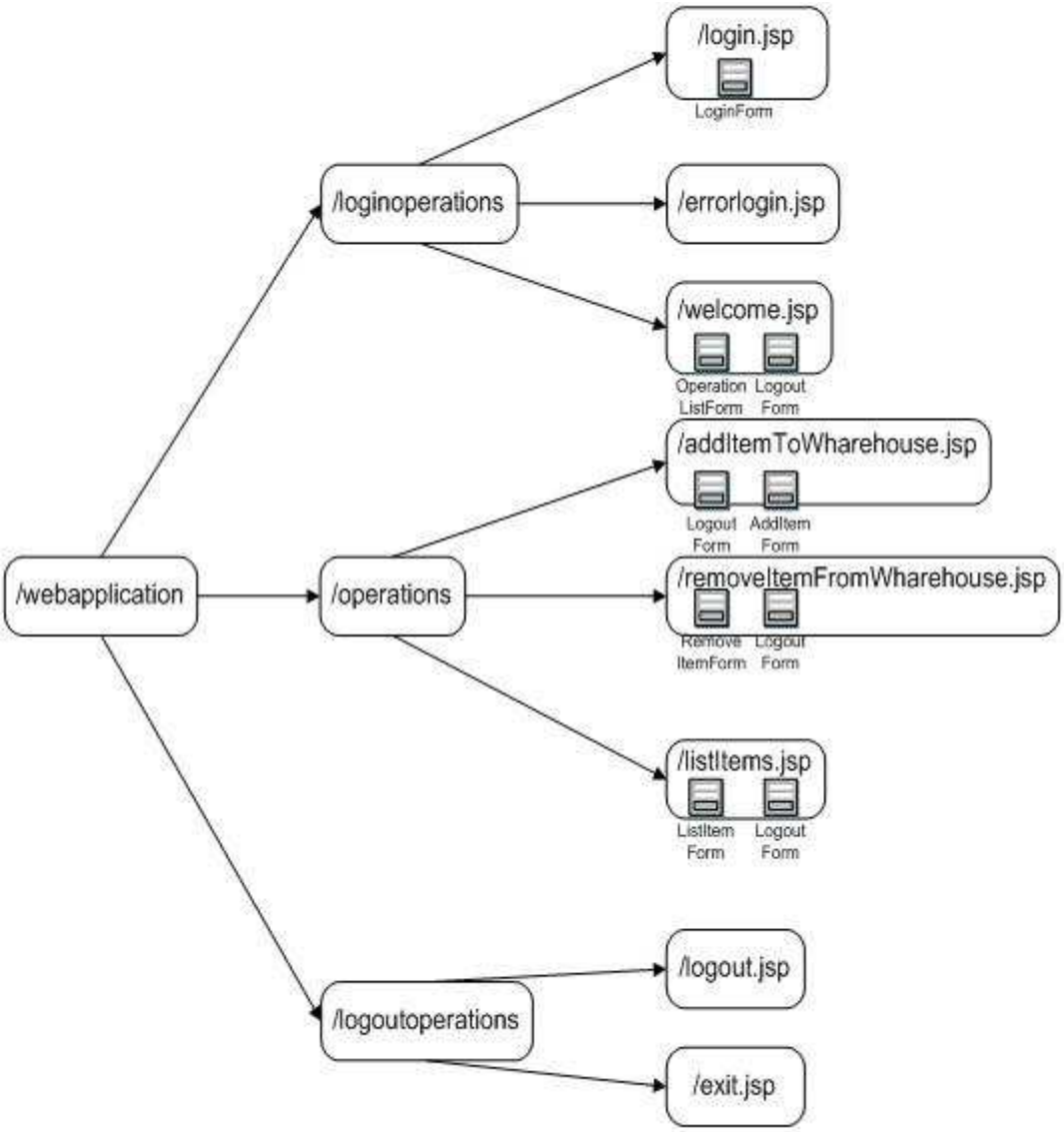


**Figure 15 Example Web Application Structure**

1. Choosing each jsp page as resources and defining VIEW and SUBMIT as operations on them where VIEW operation is responsible for viewing jsp page (e.g. HTTP get request on jsp page) and SUBMIT operation is responsible for any form submit operation resulted from that page. Resources and operations are shown below according to the given example in *figure 15*.

**Table 3 Example Mapping-1**

| Resource | Operations |
|---|---|
| login.jsp | VIEW,SUBMIT |
| Errorlogin.jsp | VIEW,SUBMIT |
| welcome.jsp | VIEW,SUBMIT |
| addItemToWharehouse.jsp | VIEW,SUBMIT |
| removeItemFromWharehouse.jsp | VIEW,SUBMIT |
| listItems.jsp | VIEW,SUBMIT |
| logout.jsp | VIEW,SUBMIT |
| Exit.jsp | VIEW,SUBMIT |

2. Adding one more level of granularity, html forms on pages can be chosen as resources and VIEW and SUBMIT operations can be defined as previously. Resource and operations are shown below according to this mapping.

**Table 4 Example Mapping-2**

| Resource | Operations |
|---|---|
| login.jsp | VIEW,LoginForm |
| Errorlogin.jsp | VIEW |
| welcome.jsp | VIEW,OperationListForm, LogoutForm |
| addItemToWharehouse.jsp | VIEW,AddItemForm, LogoutForm |
| removeItemFromWharehouse.jsp | VIEW,RemoveItemForm, LogoutForm |
| listItems.jsp | VIEW,ListItemForm, LogoutForm |
| logout.jsp | VIEW |
| Exit.jsp | VIEW |

3. If jsp pages are distributed among logical directories, directories can be chosen as resources and in this case operations will be jsp page names on those directories. Resource and operations are shown below according to this mapping.

**Table 5 Example Mapping-3**

| Resource | Operations |
|---|---|
| /webapplication/loginoperations | login.jsp, errorlogin.jsp, welcome.jsp |
| /webapplication/operations | addItemToWhareHouse,<br>removeItemToWhareHouse,<br>listItemToWhareHouse, |
| /webapplication/logoutoperations | logout.jsp, exit.jsp |

4. If the whole enterprise application is deployed on different web contexts (Web contexts can be deployed on the same application server or on different application servers.) the best way is to define each web context as resources and each page as operations on that web context. Resource and operations are shown below according to this mapping.

**Table 6 Example Mapping-4**

| Resource | Operations |
|---|---|
| Webapplication | login.jsp, errorlogin.jsp, welcome.jsp,<br>addItemToWharehouse.jsp,<br>removeItemFromWharehouse.jsp, listItems.jsp,<br>logout.jsp, exit.jsp |

On the other hand; if enterprise web applications use MVC pattern on their applications, the mappings are much more straightforward. Controller servlets (mostly called action or dispatcher servlets) become resources and their possible view actions becomes operations of CSAAS. To give an example, we can consider Struts framework, which is an open source framework that enables applications to use the MVC pattern. In Struts framework, actions and flow of web application is controlled by ActionServlets. Possible actions are passed to ActionServlets as request parameters. Consider a case where we have ***"Item.do"*** action servlet that has 3 actions *"viewItem","addItem" and "removeItem"* and according to these actions ***"Item.do"*** forward action to viewItem.jsp, addItem.jsp and

removeItem.jsp. Therefore, in this case, it is much more reasonable to define ***"Item.do"*** as resource and viewItem, addItem and removeItem actions become operations. If we reconsider the previous example and state that "Login.do" controls login and logout operations, "Operation.do" controls add, remove, list operations. Resource and operations are become as shown in *Table 7*.

**Table 7 Example Mapping-5**

| **Resource** | **Operations** |
|---|---|
| Login.do | login, errorlogin, welcome, logout, exit |
| Operation.do | addItemToWhareHouse,<br>removeItemToWhareHouse,<br>listItemToWhareHouse, |

As mentioned before, RAD specification, as well as CSAAS, does not limit the granularity level of resources, therefore, enterprise applications are free to define a mixture of mapping techniques. In order to comply with free of granularity level, EYEKS introduces Context concept, which will be covered more deeply in section 3.3.3. EYEKS's Context is well mapped to Web application's context but extends web application context. In EYEKS any path within the web application can be defined as EYEKS's context and mapping strategy is defined on that context. To use different kinds of resource mapping strategy in EYEKS, system administrations can assign different paths to different contexts and apply mapping strategies. Continuing the above example, *"/webapplication/loginoperation"*, *"/webapplication/*
*operations" and "/webapplication/logoutoperation"* can define 3 contexts and under loginoperation, the system can map resources as described in the first strategy above, under operation, mappings can be done according to the MVC pattern and under logoutoperation, mappings can be chosen as the second strategy described above.

When a request arrives to EYEKS, firstly EYEKS determines which context a request belongs to and depending on the assigned mapping strategy, resource and operations are tried to be identified. If the request does not satisfy the mapping strategy or resource and operation mapping is undefined, EYEKS simply rejects the request without doing other operations. So the whole web application must be mapped to the resource-operation pair. After successfully naming the resources and possible operations, web application becomes directly mapped to RAD domain.

### 3.2.1 Enterprise Policy Mapping

Defining resources and their possible operations to CSAAS, enables accessing to those resource possible. At this stage everyone can access to every resource on that enterprise application. Next step will be to write enterprise-level access policies that cover all business access rules of the system. However, it is important to distinguish business access rules from business flow rules. Business flow rules should not be regarded as enterprise-level access policies. Enterprise-level access policies must only define which conditions must be satisfied in order to access be granted. It is more reasonable if a different person other than the developer of an application (usually a security officer) to define the access policies. Access control policies most possibly will be discovered during the analysis phase of application domain. However it will be possible to be changed after the product is delivered.

Consider a typical example from the health informatics domain, where there are four roles; patient, physician, department secretary and general practitioner. Example access control rules are listed below;

**Rule 1:** A physician will be granted access to a patient's data if a contact exists to which he was assigned. The access rights are only valid until 30 days after the contact was closed.

**Rule 2:** The system provides the possibility to overrule the access decision, if the user requesting access to his own data.

**Rule 3:** A department secretary can create contract and assign a physician to a patient, can not see patient's data.

**Rule 4:** The patient's general practitioner has view access to all the patient's contacts, whether these contacts have been closed or not, however can not modify patient's data.

According to these rules; use cases of application can be ADDCONTRACT, CLOSECONTRACT, VIEWPATIENTDATA, MODIFYPATIENTDATA, and ASSIGNPYHSICIAN. At first glance, these use cases seems to be controlled only using RBAC policies, however defining contact validity period or viewing patient data requires more policies to satisfy the rules. Possible policies that controls these use cases can be seen in *table 8*.

**Table 8 Enterprise Policy Example**

| Use Case | Policy | Definition |
|---|---|---|
| ADDCONTRACT | AddContractPolicy | Can be controlled using only RBAC. |

**Table 8 (continued)**

| | | |
|---|---|---|
| CLOSECONTRACT | CloseContractPolicy | Can be controlled using only RBAC. |
| VIEWPATIENTDATA | ViewPatientDataPolicy | Check whether physician is assigned to that patient or whether he is patient's general practitioner. |
| MODIFYPATIENTDATA | ModifyPatientDataPolicy | Check whether physician is assigned to that patient |
| ASSIGNPYHSICIAN | AssignPyhsician | Check whether secretary assign physician under her department |

However, Rule 1 and Rule 2 are still not satisfied. CheckContactValidityPolicy must be added to satisfy Rule 1 and CheckUserReasonPolicy policy must be added to satisfy rule 2. And also ViewPatientDataPolicy and ModifyPatientDataPolicy can be redefined as CheckPatientPhysianPolicy and CheckPatientPractitioner policies. RAD specification, policies can made chain to control access, in short a full design of access policies will be;

**Table 9 Enterprise Policy Mapping Example**

| **Use Case** | **Controlled by** |
|---|---|
| ADDCONTRACT | RBAC (No need to define another policy) |
| CLOSECONTRACT | RBAC (No need to define another policy) |
| VIEWPATIENTDATA | (RBAC AND ((CheckPatientPhysianPolicy AND CheckContactValidityPolicy) OR CheckPatientPractitioner) OR CheckUserReasonPolicy |
| MODIFYPATIENTDATA | RBAC AND CheckPatientPhysianPolicy AND CheckContactValidityPolicy |
| ASSIGNPYHSICIAN | RBAC AND AssignPyhsicianPolicy |

As seen CheckContractValidityPolicy and CheckPatientPhysianPolicy can use both control VIEWPATIENTDATA and MODIFYPATIENTDATA use cases.

Another example can be given regarding the banking domain to show how mappings are done from web application page to RAD domain; consider an EFT operation that *doeft.jsp* is responsible, which has possible operations of VIEW and SUBMIT, where VIEW operation gets request parameters

USERID (which user is requested) and ACCOUNT_INFO (which account will be displayed) and generate a web page showing account details. On that page SUBMIT operation is possible that post USERID, ACCOUNT_INFO, TRANS_ACC_INFO, which denotes the account the money will be transferred to, and TRANS_AMOUNT, the amount of the money transfer. The parameters of *"doeft.jsp"* page can be regarded as security attributes that are used for evaluating policies which were described in section 3.1.

<doeft.jsp,VIEW> resource-operation pair can be linked with an enterprise security policy (EFTTimeCheckPolicy) that defines when a view operation is allowed, for example between working hours (9 am – 5 pm) and also with VIEWAccountPolicy that checks whether the account belongs to specified user. <doeft.jsp,SUBMIT> pair can also be controlled by the same policies as <doeft.jsp,VIEW> policy and additionally linked with a security policy that checks for whether the transfer could be allowed (TransAmountPolicy), for example checks whether the transfer amount is less than the upper limit of user defined EFT operation. These policies can be defined by security officer to CSAAS so that the access control rules will be separated from application code, which can be governed freely as access control rules changes without modifying the source code of the application. As seen in the example, all enterprise-level security policies can be linked with every related resource-operation and they are reusable. Mapping can be seen in table;

**Table 10 Mapping to EYEKS**

| Resource | Operation | Parameters | Policy |
|---|---|---|---|
| Doeft.jsp | VIEW | USERID<br>ACCOUNT_INFO | EFTTimeCheckPolicy<br>VIEWAccountPolicy |
| | SUBMIT | USERID<br>ACCOUNT_INFO<br>TRANS_ACC_INFO<br>TRANS_AMOUNT | EFTTimeCheckPolicy<br>VIEWAccountPolicy<br>TransAmountPolicy |

### 3.2.2 Application Security Policy Mapping

Nearly 80% of web application attacks are because of parameter manipulation or more generally data validation vulnerabilities. Improper input validation was on the top of OWASP Top Ten Security Vulnerabilities list, published in 2005. [19] A careful centric design of data validation would free web application from these vulnerabilities. However checking against possible vulnerability exploits and validating input at every point of entry to web application is costly, error-prone and unmanageable.

These “application-level” security policies to eliminate web attack risks must be taken into consideration.

CSAAS can also be used for evaluating “application-level” security policies. Different policies can be written to validate request parameters and to check request from known security exploits. Since in EYEKS, application security layer controls user’s request and asks for access permission to CSAAS, it is guaranteed that enforcement on request will not violate security policies organization wide.

Considering the previous example, a security officer can define DoEftViewSecurityPolicy on <doeft.jsp, VIEW> pair and DoEftSubmitSecurity Policy on <doeft.jsp, SUBMIT> that defines possible parameters and their expected values for each pair. On the other hand application security policies that checks for known security exploits can be added on resource to fulfill whole security policy chain.

For general use, some policies that check for known security exploits has already been implemented and built in to the system. Security officers can linked these policies to any <resource, operation> pairs in the application. These predefined policies start with SECURITY tag and can be extended or altered according to application security needs. These are;

**SECURITY_PARAM_REG_EX_POLICY:** Defines possible values for all request parameter in a regular expression format. That can be used to validate all possible parameter and values using regular expression.

**SECURITY_INJECTION_POLICY:** checks all request parameters against injection type of attacks that can be extended to cover all types of possible injection such as SQL injection XML injection and XSS.

These policies are not a full set of application security policies to eliminate web application vulnerabilities. However they can be seen as examples of securing web applications to prove that by extending these policies, it is possible that EYEKS provides a common way to fight against security exploits and can be used as a full defense system against them. The details of the application security policy execution mechanism will be given in section 3.4.

## 3.3 Operation and Architecture of EYEKS

Architecture of the proposed solution, EYEKS that allows CSAAS to be used to manage access control organization wide is shown in *figure 16*. A specific layer, so called application security layer, is created and placed in the frontier. Posterior layers consist of real web applications and databases and have no direct access to the outside world. All communications from outside world to backend web application is intercepted and authorized from application security layer. EYEKS runs as a service proxy for posterior web applications. Client requests, targeting web applications, are intercepted and EYEKS evaluates these requests according to enterprise and application level security

and redirects, alters or rejects. If a request is authenticated, it will replay the request to web applications and pass the response back to the client using HTTP Tunneling. The response can be controlled and filtered by EYEKS that eliminates risk of information disclosure

EYEKS provides authentication and secure session handling mechanisms. EYEKS can handle basic authentication methods and can also bridge to enterprise legacy authentication systems such as LDAP. EYEKS introduces EYEKS session, which can be done cookie-based, or parameter based. With session management, EYEKS eliminates unsafe handling of user sessions and also provides single-sign-on feature to posterior web applications.

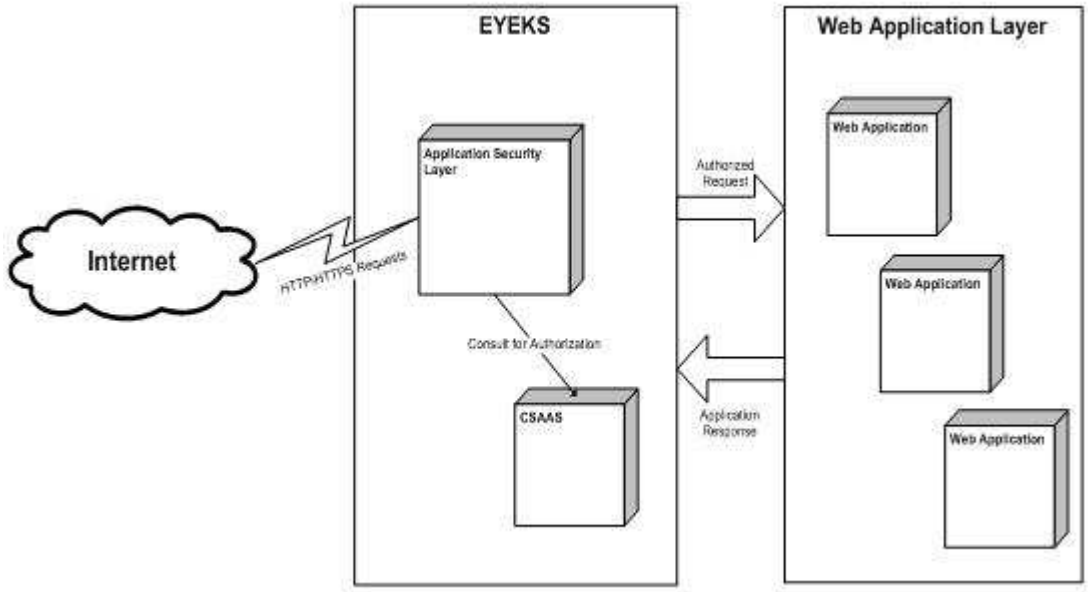


**Figure 16 Architecture of EYEKS**

CSAAS has been placed in this layer and can only communicate with application security server. Whenever a request is intercepted, EYEKS maps the request to RAD specification domain, determine resource operation pair as described in section 3.2. Request parameters and HTTP headers are extracted from the request and passed to CSAAS as security attributes with resource, operation pairs. CSAAS evaluates the request according to enterprise access rules and send the result back to application security layer, stating whether the request is authorized or not.

### 3.3.1 Application Security Layer

Application security layer can be run as stand-alone server or deployed on any kind of J2EE based application servers. After configured accordingly, it intercepts all requests coming from client. Although can be extended as described in section 3.3.2. Application Security Layer executes 3 main scenarios; Login, Page Request and Logout.

**Login Scenario:** This scenario begins if;

- Client makes a request to login page of any backend application.
- Client times-out and makes another request.
- Client makes a request which has invalid or no token.

Scenario continues sequentially as follows:

1. Client provides any login credentials such as user id password pair, hardware token, LDAP key.

2. Application security layer finds out what backend application the user wants to login from the requested login page.

3. Application security layer tries to authenticate the user using applicable authentication methods that the user provides.

4. If authentication fails, corresponding message is prepared and sent back to user as a response to the request.

5. If authentication succeeds, application security layer asks if user has LOGIN rights on the requested resource (web application) if yes scenario continues, otherwise login request is denied.

6. Distributed session for the user is created and also inserted in database or LDAP.

7. A unique token is created and encrypted from user id, current time and client IP.

8. Log manager creates a login log and stores it to the store provided (Database or file).

9. Application security layer makes a HTTP/HTTPS request to the proper web application and tries to fetch the welcome page.

10. When the welcome page is received, created token is inserted to the page and sent back to the client browser.

**Page Request Scenario:** This scenario begins at every page request made by the user.

Scenario continues sequentially as follows:

1. Token is resolved from the client request and decrypted. User id, next request id fields are extracted.

2. If token is valid (user is not timed-out and sequence of the request is true), all parameter and value pairs are extracted from the request. The resolution of which web application the request belongs to is carried out.

3. CSAAS tries to authorize the user request using page name as operation, web application as resource and parameter value pairs to be used in policy of authorization.

4. If the user is an authorized user to use that web page, user session is activated by session manager, otherwise the request is denied.

5. New token is generated from the coming token. (next request id is rewritten)

6. Application security layer makes a HTTP/HTTPS request to proper web application and gets the requested page.

7. Log manager creates an access log and stores it to the store provided (Database or file).

8. New token is inserted to the coming page and sent back to the client browser.

**Logout:** This scenario starts if a request is intercepted and identified as logout operation. Scenario continues sequentially as follows:

1. Token is resolved from the client request and decrypted. User id, next request id fields are extracted.

2. User session is dropped from database.

3. Log manager creates an access log and store it to the store provided (Database or file).

4. Application security layer make a HTTP/HTTPS request to proper web application and gets the logout page.

5. Logout page is sent back to client.

Application security layer is designed as logically layered structure. All user requests are captured at uppermost layer and processed though the inner layers and then dispatched to the backend web applications. The logical layers for application security layer are (from uppermost to innermost layer):

**1. Request Listener Layer:**

Request listener is the interception point of all client requests. This layer consists of two different implementations that implements ISessionHandler interface; one is for HTTP Component based stand-alone server implementation which will be described in section 3.5 and the other one is Servlet based implementation that can be deployed on any J2EE based application server. As well as any other web application, common listener port is 80 for HTTP requests and 443 for HTTPS requests. Free from interceptor interface, it must be configured to hold root address (/) so all requests to application security layer can be intercepted. Request Listener Layer extracts the HTTP Request, such as gathering headers, request parameters and user tokens, generates a SessionHolder object and passes it to Request Parser Layer.

**2. Request Parser Layer:**

Although there is not a strict logical border to distinguish this layer, in fact it is one of the pre-operations belonging to OperationsManager class from operation layer, since it is must and critical operation called ContextResolveOperation, it can be regarded as a specific layer. This layer is where all requests are evaluated to find which scenario it belongs which will be described in section 3.3.2. After finding operation command (from now on the word Command is used for scenario), a request operation chain is constructed as stated in configuration files and execution continues with operation layer.

**3. Operation Layer:**

Operation Layer is a core part of the whole application security layer; it is where all requests are evaluated and managed. As will be described in next section, various operations can be registered to all served contexts and can be sequentially executed to form an operation chain. These operations vary from context resolving, authentication and authorization to session management, request and content filtering. SessionHolder object, which is generated from request listener layer, is the connection point of all these operations. An operation can add, modify or remove attributes and their values for further use. Operations can break the chain by raising exceptions if any expected event or state is reached. For example, authorization, session management operations can raise exception if any defined security rule is violated. Operations can connect to CSAAS and ask for authorization or validate application security policies whenever needed. At some point in the execution of operations there must be a request dispatcher operation, which makes a request to backend web applications according to the current state of SessionHolder object, the response is stored again in SessionHolder and execution of operations continues.

Security officer can manage these operations according to the security needs of backend web application. Therefore if any web application or any path within a web application does not require any security mechanism, for example just consisting of images belonging to the web application, the operation chain will only contain request dispatcher operation.

**4. Request Dispatcher Layer:**

Request dispatcher layer is in fact, a specialized operation that sits in the middle of the operation chain. But since it is the boundary operation between application security layer and posterior web application, it can be regarded as a layer. Request dispatcher layer has two different implementations, as request listener layer, one depends on HTTPComponents libraries and the other uses Java built-in HTTP connection libraries. Request dispatcher module handles with HTTP tunneling, requesting the original page from backend web application. The details of HTTP tunneling concept will be given in section 3.3.5.

The sequence diagram of executing a client request is given in *figure 17*. Client requests are intercepted by Security Layer and depending on implementation (HTTPComponent based or Servlet based) though a suitable interface (SessionHandler), a SessionHolder object is created and passed to OperationsManager object. OperationsManager finds which command to be executed according to the request and executes corresponding operations chain. From one of the operations a request that mimics the original client request is sent to posterior web applications and the response is captured. The response is then processed through the operation chain again. After the whole operation chain is executed, SessionHolder object is passed to SecurityLayer and as in the case of interception the response is sent back to client though configured SessionHandler interface.

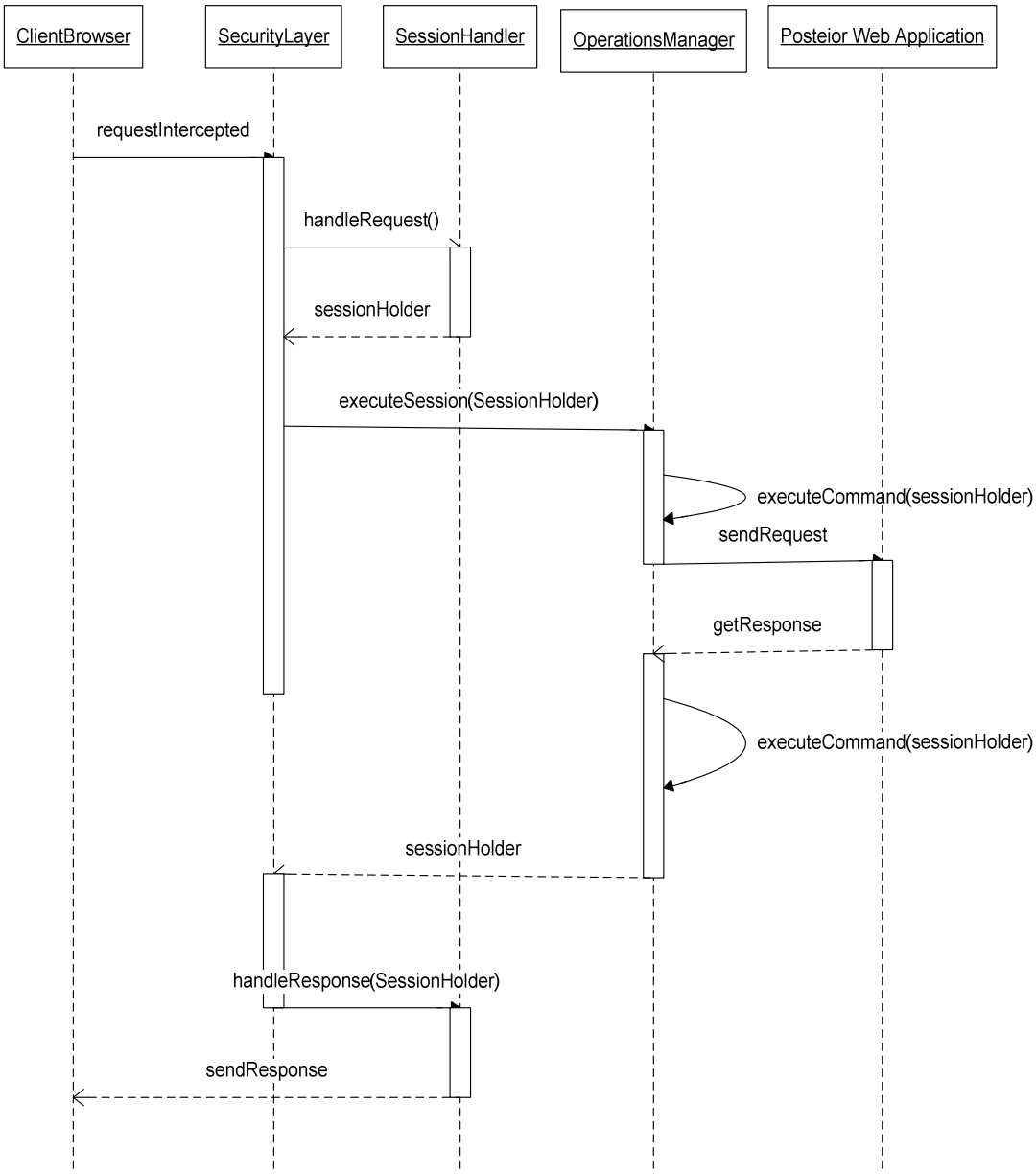


**Figure 17 Sequence Diagram of Request Execution**

### 3.3.2 Request/Response Operation Chain

Operation chain is controlled by a singleton class, OperationsManager, which is initialized as application security layer's startup and control execution of application security layer. After the

request listener layer intercepts a request, it creates and passes a SessionHolder object to OperationsManager, which holds all information related to request and response. The attributes of SessionHolder object are;

**int direction:** Responsible for holding execution direction. Initially set from SessionHandler class to FROM_BROWSER after redirection RedirectOperation class sets to TO_BROWSER.

**String method:** Responsible for holding HTTP Method. For example, GET, POST, SET from SessionHandler class.

**List<NameValuePair> headerMap:** Responsible for holding HTTP Headers. Initially set from SessionHandler class and then changed by RedirectOperation class.

**List<NameValuePair> requestParameters:** Responsible for holding Request Parameters. Initially set from SessionHandler class and then changed by RedirectOperation class.

**String content:** Responsible for holding String Content of HTTP response. Set from RedirectOperation class, after getting response from redirection used for text/html content type.

**byte[] binaryContent:** Responsible for holding binary Content of HTTP response. Set from RedirectOperation class after getting response from redirection used for binary content type.

**String contentType:** Responsible for holding content type of HTTP response, set from RedirectOperation class.

**String contentEncoding:** Responsible for holding content encoding of HTTP response, Set from RedirectOperation class.

**String targetURI:** Responsible for holding whole target URI of the request, for example /web-apps/content/index.jsp?parameter1=1&parameter2=2, set from SessionHandler class.

**String targetURIBase:** Responsible for holding targetURI without parameter part of the request, for example previous URI becomes /web-apps/content/index.jsp. Set from SessionHandler class.

**int statusCode:** Responsible for holding status code of HTTP response. Set from RedirectOperation class after redirection.

**IRequestContext requestContext:** Holds the context information about the request, Set from ContextResolveOperation.

**Token token:** Holds the encrypted token, which depends on user session.

**String requestIP:** Hold the IP information of the request.

**UserSession userSession:** Application security layer fetches user session information such as userid, sequence number of the request, timestamp from appropriate provider (LDAP or database).

**UserAccount userAccount:** Holds user account information retrieved from database such as last login details, successful and unsuccessful login information.

SessionHolder object exists during execution of a request and ends after a response is sent back to the client. All operations in operation chain can access the attributes of this object and modify them. It can be regarded as a communication interface for each operation.

As can be seen in class diagram given in *figure 18*; OperationsManager object consists of a list of commands and pre/post operations. Pre operations are executed before any command is executed and post operations are executed after successful execution of a command.

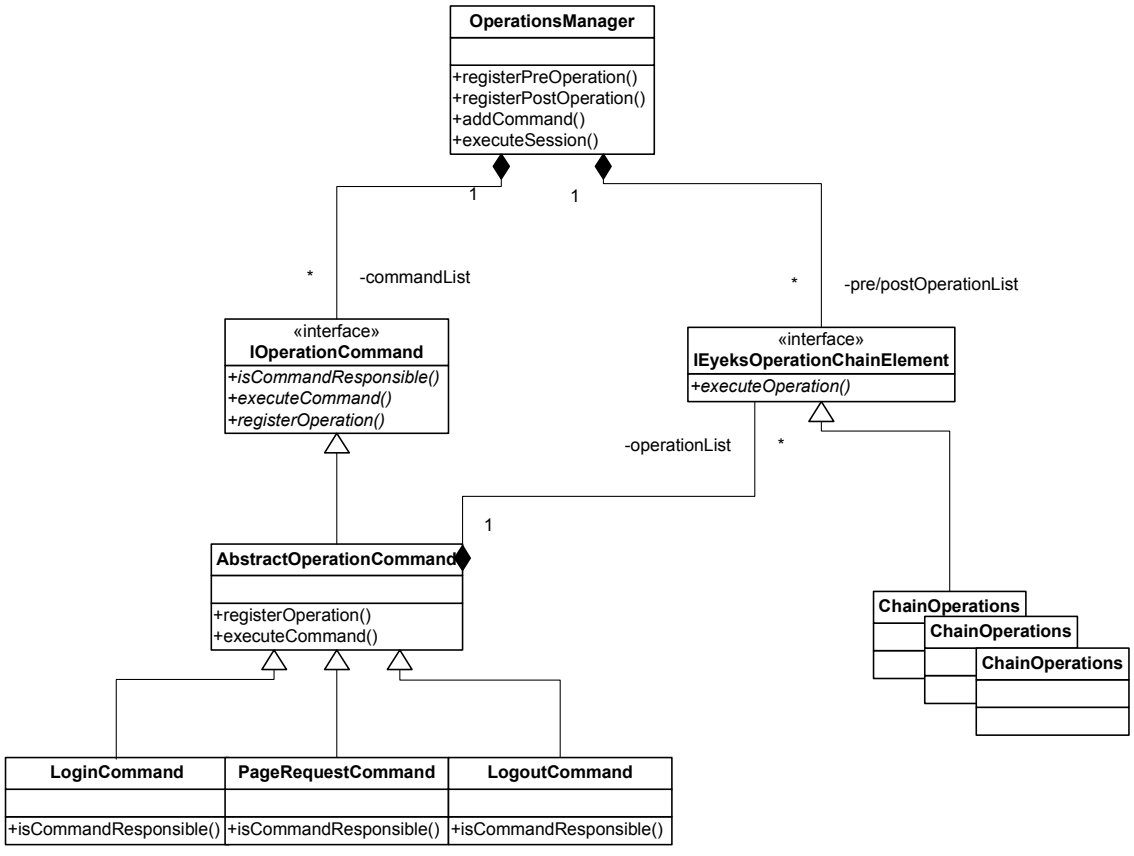


**Figure 18 Operation Class Diagram**

#### 3.3.2.1 Commands

System scenarios (Login, PageRequest, and Logout) are defined using Command classes that implement IOperationCommand and extend AbstractOperationCommand. The system is designed using Chain of Responsibility pattern so that each Command class must implement isCommandResponsible method that returns a Boolean value that defines responsibility. Whenever a request arrives, OperationsManager calls isCommandResponsible method of every command class that is registered and finds which command to be executed. Every command has registered operations that must be executed to fulfill the scenario.

isCommandResponsible method, has a parameter SessionHolder so Command class can define their responsibilities according to every attribute of a request (such as HTTP method type, the names and values of headers or request parameter, request URI, content type, etc) The system has 3 predefined commands that take responsibility according to request URI; LoginCommand checks if a request target a specific login URL that can be defined in a configuration file and LoginCommand checks for a specific logout URL that also defined in configuration file. PageRequest takes responsibility if a targeted URI is valid.

The system can be extended by implementing IOperationCommand interface and adding to class path. For example FileUploadCommand class can be defined to welcome uploaded files that takes responsibility if content type is “application/octet-stream” or “multipart/form-data” where HTTP method is a POST.

#### 3.3.2.2 Operations

The operations of Application Security Layer must implement IEyeksOperationChainElement interface by implementing executeOperation method that takes SessionHolder object. Operations can do any operations like modifying session holder values such as request parameters, headers, response content, checking access and authorization though CSAAS, adding logs or updating user account. Operations can be registered directly to OperationsManager as pre or post operations that are executed regardless of responsible command or registered to Command classes that define possible operations of commands. Any operation can break execution chain by raising an Exception that extends EyeksExceptionBase class. Like command classes, the system can be extended by adding new operation classes that implements IOperationCommand interface and added to class path.

#### 3.3.2.3 Request Execution Collaboration

A more detailed description of EYEKS request execution, which was given in section 3.3.1, is shown in *figure 19*.

1. The execution starts if a request is intercepted by Security Layer. Security layer passes the request object (for servlet based implementation the request object is HttpServletRequest and for HTTPComponent based implementation, it is HttpRequest object.) to SessionHandler object (for servlet based implementation session handler object is ServletSessionHandler and for HTTPComponent based implementation, it is HttpComponentsSessionHandler) by calling handleRequest method. SessionHandler returns SessionHolder object and Security Layer.

2. Security Layer passes SessionHolder object to OperationsManager by calling executeSession method.

3. OperationsManager fetches pre operations from previously registered pre operations and sequentially executes them by calling executeOperation method of each object. SessionHolder is passed as an argument.

4. One of the mandatory pre operations is ContextResolveOperation. ContextResolveOperation passes targeted URI (from SessionHolder) to ContextResolver object, demand to which context a request targeted. Context name is returned as string from resolvePath method of ContextResolver object then context name is inserted in SessionHolder object to be used in the future.

5. After all pre operation execution are finished, OperationsManager tries to find which command is responsible for handling the coming request by calling isCommandResponsible method of OperationCommand objects.

6. After finding responsible command, OperationsManager calls executeCommand method of responsible OperationCommand objects.

7. Responsible OperationCommand object fetches registered operations and executes them sequentially by calling executeOperation method of each operation.

8. OperationsManager fetches pre operations from previously registered pre operations and sequentially executes them by calling executeOperation method of each object.

9. SessionHolder object is passed to SessionHolder object by calling handleResponse

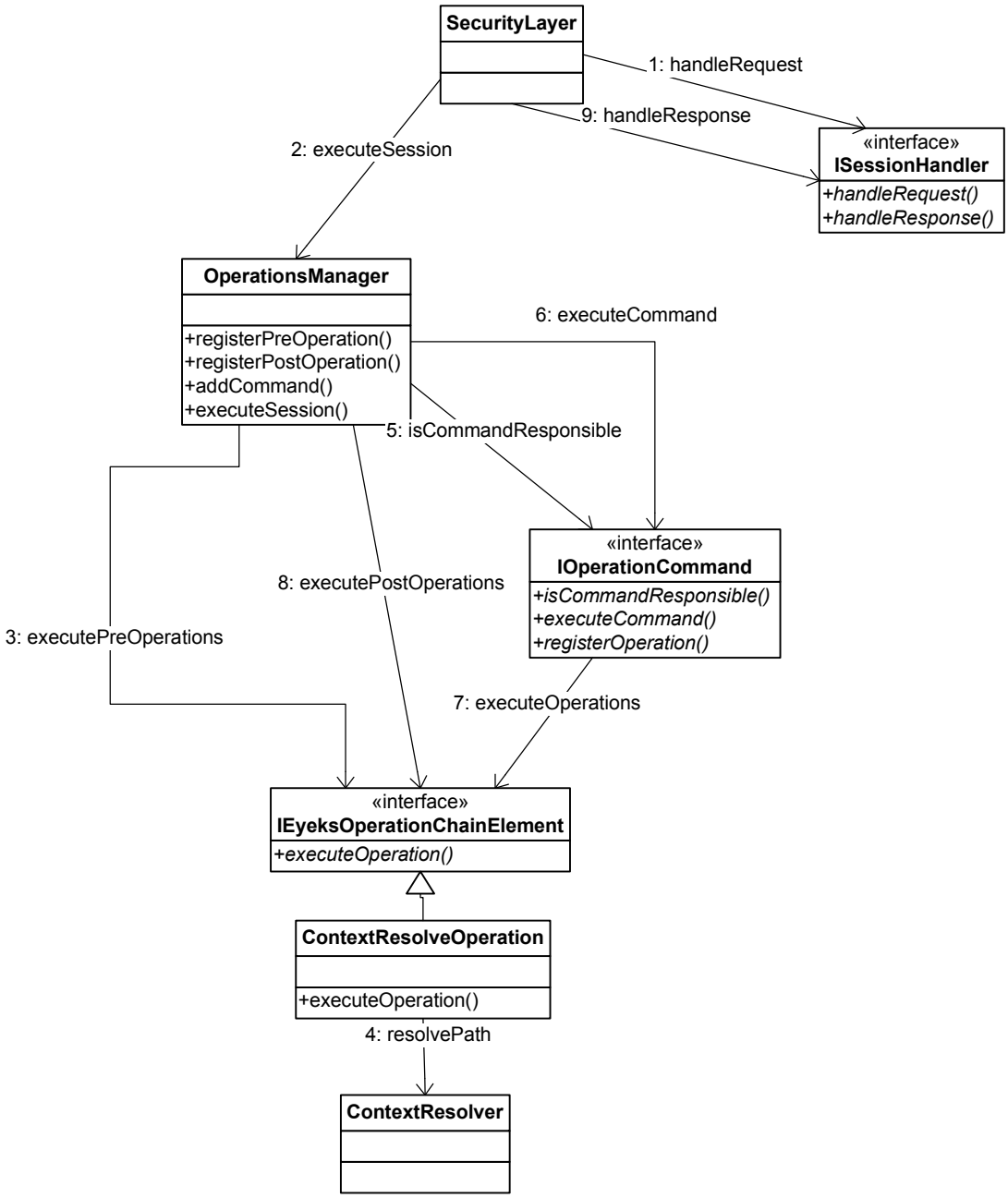


**Figure 19 Collaboration Diagram of Request Execution**

After execution of all necessary operations on the request, SessionHandler object creates response object (for servlet based implementation the request object is HttpServletResponse and for HTTPComponent based implementation, it is HttpResponse object.) and send it to client as a response of the request.

#### 3.3.2.4 Exception Handling

Any operation (object that implements IEyeksOperationChainElement) could break the execution chain by raising an exception that extends EyeksExceptionBase. EYEKS has 6 types of implemented exceptions that extend EyeksExceptionBase;

**Eyeks Authentication Exception:** captures authentication exception that can be raised from operations that are responsible for the authentication mechanism such as AuthenticationCheckOperation, UserLoginOperation. Unsuccessful login tries, tries to access locked accounts can lead to authentication exception. On the other hand, any exception, raised within the authentication mechanism, such as failure to communicate with external authentication system (CSAAS) is catch and converted to Eyeks authentication exception.

**Eyeks Authorization Exception:** captures authorization exceptions, for example when a request fails to satisfy enterprise access policies or application security policies. Any communication error with CSAAS or any unexpected errors coming from CSAAS is captured and converted to Eyeks authorization exception.

**Eyeks Context Resolver Exception:** captures any exception during context resolve operation. If any request is failed to be mapped any defined context, Context Resolver object raises this exception. The responsible operation, ContextResolveOperation does not catch this exception and directly throws it.

**Eyeks Page Request Exception:** If an exception occurs during fetching a page from backend web applications or any error occurs during HTTP tunneling, Eyeks Page Request exception is generated and raised.

**Eyeks Database Exception:** Any exception coming from database server. (Such as connection or SQL errors) is captured and converted to Eyeks Database exception. After conversion, the responsible operation raises it to break operation chain.

**Eyeks Session Exception:** If a request comes from unauthenticated user or if user session is invalid (possibly time out) or replay attack is detected. Eyeks Session exceptions are generated and thrown.

Security Layer captures any exceptions that are raised from operation chain and passes it to EyeksExceptionDispatcher object, which is a singleton object add responsible for generating appropriate error messages. Multi-language error messages are stored in a configuration files, called message.properties EyeksExceptionDispatcher object tries to map mnemonic of exceptions messages to original error messages.

The templates of error pages are stored with EYEKS. Error pages can be designed independently; however the place to show the message must be labeled with <MESSAGE> tag.

EyeksExceptionDispatcher search appropriate error page template for this tag and replace it with the original error message.

For example, AuthenticationCheckOperation, which is a check for authentication request added to LoginCommand, raises EyeksAuthenticationException with a message UNSUCCESS. Security Layer captures this exception and passes it to EyeksExceptionDispatcher object. This object loads appropriate configuration file according to request language (by checking Accept-Language header) and maps UNSUCCESS to error message, which is "Invalid user name or password." tr.com.eyeks.exceptions.EyeksAuthenticationException, UNSUCCESS, Unsuccessful login try.

### 3.3.3 Context Mapping

The initial step of nearly all web application attacks is to reveal underlying web application structure. Hence, web applications must prevent information leakage about the structure of the application. A web context is basically a directory or directory structure that is published on the web. Like reverse-proxies, EYEKS allows mappings of different contexts to virtual structure of application. So from the client's point of view, whole application seems to be served from only one web context, but since it is just virtual, the directory information of real web context will be safe.

An example of typical context mapping is given in *figure 20*. Assume that real web application is served from two different servers, where one is an application server that serves web application and the other is a web server that just serves static web content. The context deployed on application server can be labeled as Context 1 and the context deployed on web server is labeled as Context 2. As mentioned in section 3.2, to use different kinds of resource mapping strategy in EYEKS, a sub-directory consists of "shoppingChart" and "customer" directories, is labeled as a different context, called Context 3.

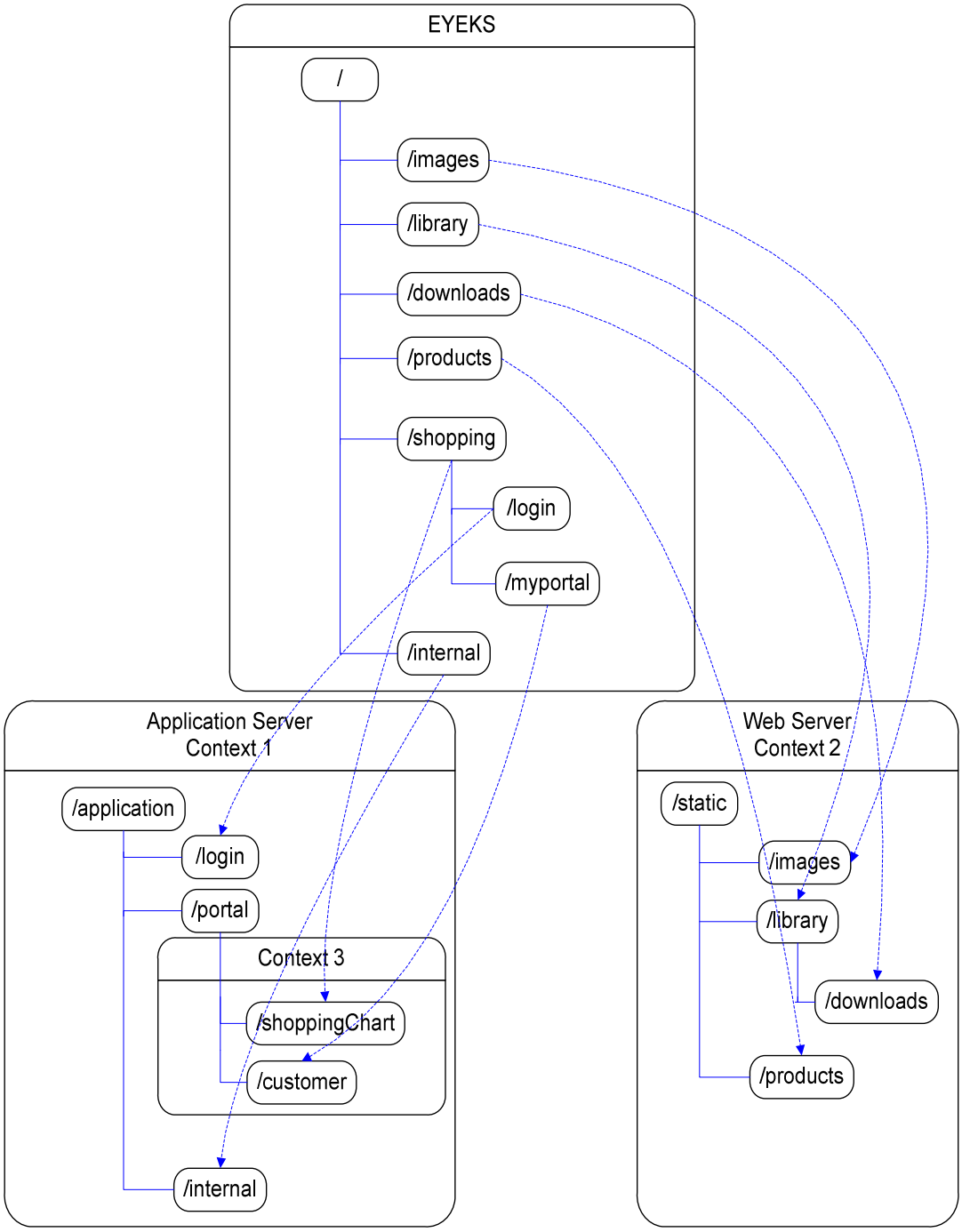


**Figure 20 Example Context Mapping**

As can be seen in the figure, original directories can be virtually mapped (context2's /static/product to Eyeks's /product) or renamed (context1's /application/portal/ customer to Eyeks's /shopping/my portal).

Therefore, from client's point of view, the whole web application consists of structure what is defined in EYEKS and there is no way to reveal real structure.

The context mappings are defined using two different configuration files; *"context.properties"* holds context definitions and their properties like the where the context is deployed, on which port it serves. A possible context configuration for a given example is below.

*eyeks.context1.name=context1*
*eyeks.context1.host=backhandserver1*
*eyeks.context1.port=8080*
*eyeks.context1.protocol=http*
*eyeks.context1.welcomepage=/login/welcome.jsp*

*eyeks.context2.name=context2*
*eyeks.context2.host=backhandserver2*
*eyeks.context2.port=80*
*eyeks.context2.protocol=http*

*eyeks.context3.name=context3*
*eyeks.context3.host=backhandserver1*
*eyeks.context3.port=8080*
*eyeks.context3.protocol=http*
*eyeks.context3.basepath=/application/portal*

As seen in the configuration file, a context can additionally have "welcomepage" and "basepath" properties, where "welcomepage" refers to the redirection path if a web application requires authentication. After successful login, EYEKS redirects the request to this path. "basepath" property can be used if a context is a sub-context of some other real context.

The mappings are defined in *"path.properties"* configuration file. This file has entities in a format like **"EyeksPath" => "context name","realpath"**. The configuration file for a given example will be.

1. */images => context2, /static/images*
2. */library => context2, /static/library*
3. */downloads => context2, /static/library/downloads*
4. */products => context2, /static/products*
5. */shopping => context3, /shoppingChart*
6. */shopping/myportal => context3, /customer*
7. */shopping/login => context1, /application/login*
8. */internal => context1, /internal*

The mapping rule can overwrite previous rules, for example 5$^{th}$ rule is overwritten by the 6$^{th}$ and 7$^{th}$ rules. Any request target in a location under /shopping will be mapped by 5$^{th}$ rule, but request target under /shopping/myportal will be mapped by 6$^{th}$ and under /shopping/login will be mapped by 7$^{th}$ rule. Therefore, for example /shopping/Chart/addItemToShoppingChart.do will be mapped by rule 5$^{th}$ to context3 and /shoppingChart/Chart/addItemToShoppingChart.do; /shopping/login/Login.do will be

mapped by rule 7th to context1 and /application/login/Login.do; /shopping/myportal/customerPortal /viewCustomer.jsp will be mapped by rule 6th to context 3 and /customer/customerPortal/view Customer.jsp. The details of redirection operation will be covered in section 3.3.5.

ContextResolver object is responsible for context mapping and path conversion operations. ContextResolver has one instance of ContextBuilder and one instance of PathBuilder objects; that manages context operations and path operations respectively. Init method of ContextResolver initialize configuration files (as described above) and passes file handlers to corresponding build methods of relevant objects. Build methods of both objects, read configuration files and load contexts and path mappings. On the other hand, ContextBuilder provides necessary methods to manipulate the mappings dynamically. Corresponding class diagram is given in *figure 21*.

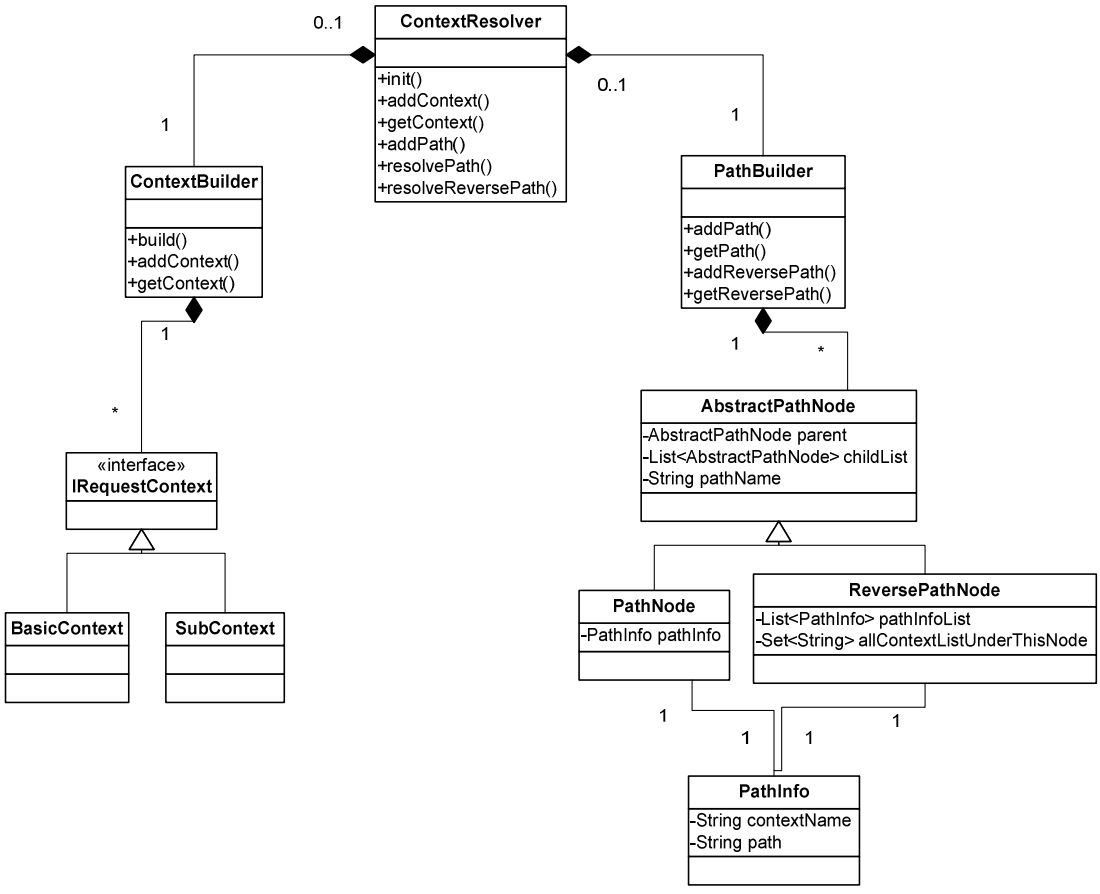


**Figure 21 Context Resolver Class Diagram**

PathBuilder object holds two different trees for path resolution, one of them consisting of PathNode objects and the other one consist of ReversePathNode objects. Each PathNode object has PathInfo object, which refers to which context it, belongs to and what path is used to transform. A corresponding path tree for given example is given below.

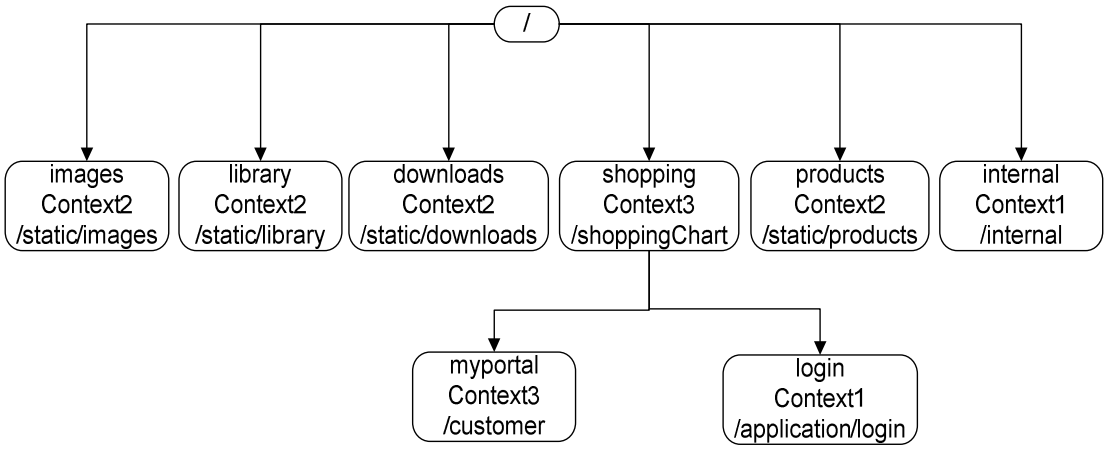


**Figure 22 Example Context Mapping Tree**

The resolve path algorithm, tries to find most logical match of targeting request path. For example if a request targets a path /shopping/myportal/customerPortal/view Customer.jsp, firstly root node is traversed, then shopping node and finally myportal node. The path to transform which is /customer is fetched on this node and the remaining part of target path which is /customerPortal/viewCustomer.jsp is added to this path, so the translated path will be /customer/customerPortal/view Customer.jsp and context is found on myportal node as Context3. On the other hand if a request targets /shopping/Chart/addItemToShoppingChart.do path. The search will be ended on shopping node, /shoppingChart is fetched and added to remaining path, which becomes /shoppingChart/Chart/addItemToShoppingChart.do and context is found on this node as Context3.

After redirecting the request to backhand servers, resolving reverse paths are also necessary, especially if the application requires HTTP session based on cookies. However resolving reverse path is more complex, Eyeks's paths must be unique so that virtual nodes can be mapped to only one real path. However, different contexts can most probably have the same directory name, so going backwards is problematic. To resolve this name conflicts we can use context name which is resolved

already during forward pass. To give more specific example that reflects reverse path resolution, consider the example below;

1. */path1 => context0, /web-apps/content*
2. */path1/path2 => context2, /content*
3. */path1/path2/path3 => context1, /web-apps*
4. */path1/path2/path4 => context3, /content*

Different from PathNode, for reserve path resolution ReservePathNode's are used for constructing tree. ReservePathNode has list of PathInfo objects where each of them holds preceding context names and paths. For the example above, the corresponding tree will be:

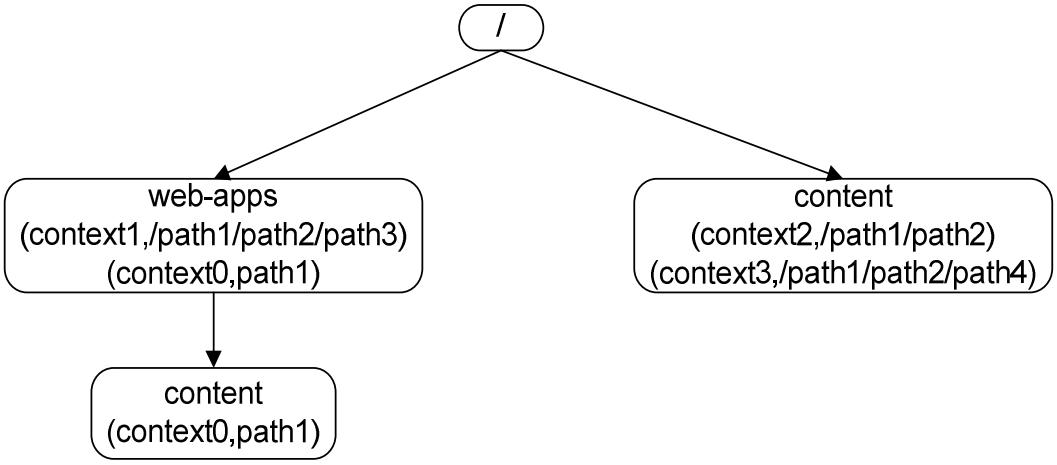


**Figure 23 Reverse Context Mapping Tree**

So web-apps node holds (context1, /path/path/path3) PathInfo for 3rd rule and also holds preceding node path information (context0, /path1) because of the 1st rule. On content node, there is a naming conflict, so it holds (contextt2, /path1/path2) for 2nd rule, and (context3, /path1/path2/path4) for 4th rule.

So if a request targets /path1/server/showStatistics.jsp path. At the first step, forward path is resolved to be /web-apps/content/server/showStatistics.jsp and context is context0 by applying 1st rule and after redirection, we have (context0, /web-apps/content/server/showStatistics.jsp) passed as parameter to resolveReversePath method. The algorithm traverses web-apps and then content node and finds

reverse path as path1 and append remaining part of the reserve path so that the full reverse path will be /path1/server/showStatistics.jsp.

**ContextResolveOperation** is a must and is one of the pre-operations of EYEKS operation chain. It is responsible for starting context resolve sequence. This operation gets client's target request URI from SessionHolder object and starts context resolving by calling "resolvePath" method of ContextResolver object. The return values, context and forward path, is set to SessionHolder object.

The reverse operation is done be **HeaderReverseDirectionOperation**, which is one of the post operations of EYEKS and is a mandatory operation if backend applications use HTTP Session. This operation's responsibility is to track headers that come back from backend applications after redirection to find "Set-Cookie" header. This header has "Path" property, reflecting which path a session cookie must send back from client browser to the server. Since we have used virtual paths in client browser and HTTP Sessions are created from backend servers depending on real paths, these real paths must be converted to virtual EYEKS path. After retrieving "Path" value in header, this operation calls resolveReversePath method of ContextResolver by passing context name and path. The old "Path" value is replaced by return value of this function, so that it will be inserted in "Set-Cookie" header.

### 3.3.4 Session Management

There are three typical session management techniques; cookie based URL rewriting, hidden form fields. EYEKS could allow backend applications to create and manage their HTTP Sessions only if all operations in the operation chain preserve HTTP headers. Also to use cookie based HTTP sessions, one of the post operations, HeaderReverseRedirectionOperation must also be added to operation chain. URL rewriting is the most insecure way of handling sessions, so EYEKS rejects any session carried out by URL rewriting. In order to use hidden form field based session handling, session parameter must be defined to CSAAS as a safe parameter for all possible resource-operation pairs.

Application security layer introduces EYEKS Session where the method not only considers security but also considers distribution execution so all backend web application can share the same session which is not possible using HTTP Session. Session management is handled using encrypted token, which holds user credentials such as user id and request sequence number to identify the user. Using sequence information avoids session hijacking so even if a malicious user hijacks this encrypted token, sending it back to the application security layer will not work. This token is inserted in every response to user request and it is granted that it will send back with the next user request. As in HTTP Sessions, EYEKS Session can be handled by two different methods, cookie based and hidden form field. Cookie based EYEKS Session management is done by HeaderManagedTokenGetOperation and HeaderManagedTokenPutOperation operations. Therefore, if cookie based mechanism is chosen, HeaderManagedTokenGetOperation must be inserted before redirection operation, where it checks

request headers for the Cookie called **EYEKSTOKEN,** decrypt it, and convert it to Token object and put this object to SessionHolder. HeaderManagedTokenPutOperation must be inserted to the operation chain after redirection operation, where it checks SessionHolder object for Token object, encrypt it and put the encrypted token to response headers by setting the cookie **EYEKSTOKEN**.

User session consist of userid, timestamp, a sequence number, login IP and can be stored either in database or LDAP where userid is a primary key if it is stored in database or DN (distinguished name) if stored in LDAP. Userid refers to the user that login to the system, timestamp holds the timestamp of the last request and sequence number refers to the last sequence number of the user's request.

Token object consist of userid, timestamp and a sequence number and is initially created by UserLoginOperation which is one of the login operations that can be added to the operation chain. Userid refers to the user id that login to the system, timestamp holds the timestamp of the token creation and sequence number refers to the sequence number of the user's request. After a successful login, UserLoginOperation creates token, sets sequence number to 1 and inserts to SessionHolder as well as creating a session record either in database or LDAP.

If added to the operation chain, it is SessionCheck operation that checks for the coming token. SessionCheck operation makes a request to CSAAS for every client request with resource as context name and operation as **PAGE_REQUEST** so that the validity of token can be controlled by CSAAS. SessionCheck operation also passes current timestamp, timestamp of the token, login IP, current request IP, sequence coming from the token and sequence from the database. System administrator can assign different kinds of policies on this <ContextName, PAGE_REQUEST> pair that controls Eyeks session. By default session policy controls whether login IP is equal to current request IP, sequence of the token is equal to sequence from the database, current timestamp is both bigger then timestamp of the token and within the time-out period.

The collaboration of creating user session during user login operation and checking user session during the page request is given in following figures.

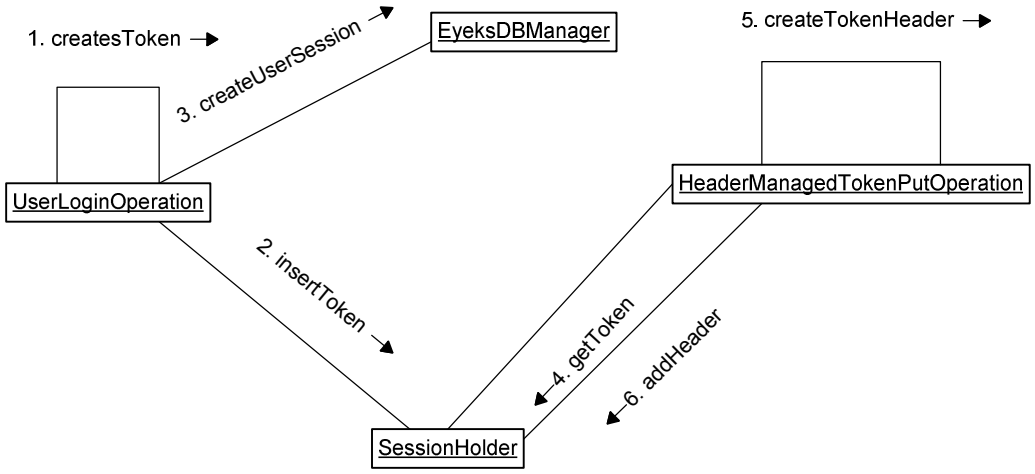


**Figure 24 The Collaboration of Creating User Session**

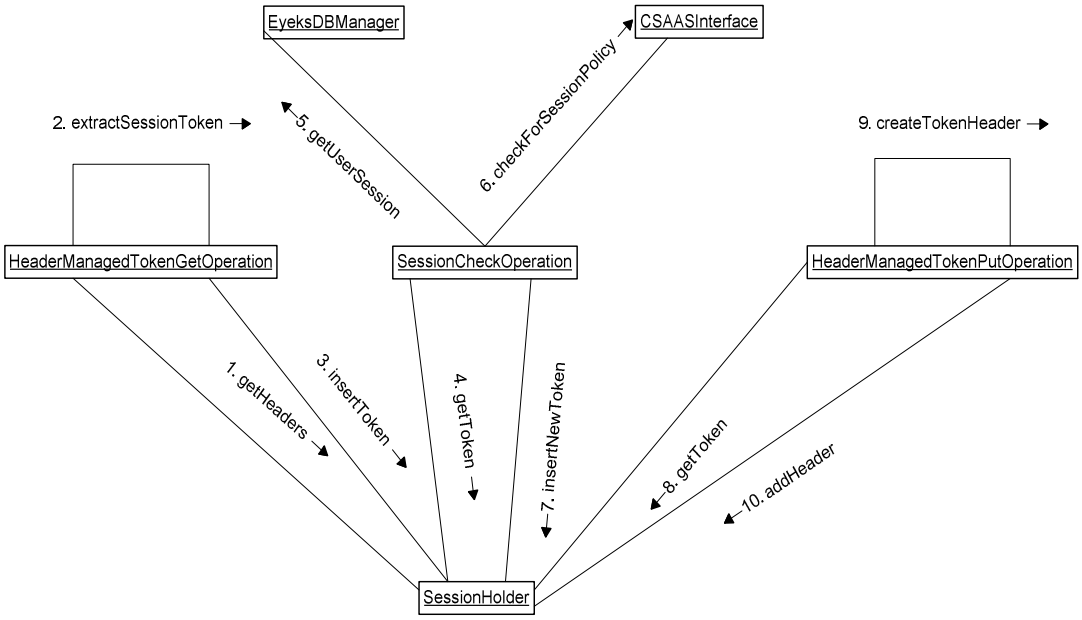


**Figure 25 The Collaboration of Page Request**

Before user session is created, users must login to Eyeks. In fact, UserLoginOperation is the last operation in the login sequence. Users must be authenticated and login policies like checking maximum login tries is exceeded or not and checking allowed login IP must be satisfied before any user session to be done.

In order to use Eyeks login and session, a user account must be created. Like Eyeks session, user account can be stored either in a database or in LDAP. In either way, user account consists of user id, status code referring account status (ACTIVE_ACCOUNT, LOCK, LOCKED_PASSWORD and DISABLED), last successful login time, last successful login IP, last fail login time; last fail login IP, current login try number, last password change time and description fields.

Two different Eyeks operations have been implemented that can be added to login sequence. PreAuthenticationCheckOperation retrieves userid which is a request parameter for login operation, using userid fetches user account and user session objects from database or LDAP, using these values make a request to CSAAS with resource as context name, operation as LOGIN_REQUEST and security attributes as user account attributes like login IP, account status code, last successful login time, last successful login IP, last fail login time, last fail login IP and current login try number. If CSAAS sends back access_allowed value, this operation ends successfully; on the other hand if CSAAS's response is access_not_allowed, PreAuthenticationCheckOperation creates an Eyeks Authentication Exception with REASON message coming from CSAAS's result list.

After pre authentication policies are satisfied, now authentication and login policies must be satisfied. AuthenticationCheckOperation is responsible for authentication check and asks CSAAS for access decision on login operation. This operation firstly, retrieves USERID and PASSWORD request parameters from session holder object and asks CSAAS for authentication. After CSAAS sends back the authentication answer (is authenticated or not), this operation makes another request to CSAAS to check login policies with resource as context name, operation as LOGIN with attributes authentication result and login try number. If user's login request satisfies all policies attached to LOGIN operation on requested context, corresponding updates will be done on user account and user session objects in database or LDAP, otherwise an Eyeks Authentication Exception will be thrown with REASON message coming from CSAAS's result list.

Using these two operations, Login behavior of EYEKS can be controlled by adding any policy <Context name, LOGIN_REQUEST> and <Context name, LOGIN> pairs. System administrator can force any login policy to be satisfied by each login request to fulfill enterprise application security needs.

### 3.3.5 Request Proxying

As mentioned before, application security layer is located in DMZ and no direct connection is allowed from client browser to backend servers as mentioned in section 3.3.1. In order to achieve this,

application security layer acts as a proxy that intercepts the client request, while keeping them alive, opens a new HTTP connection to backend servers by proxying the request, getting the response and dispatching it to original request as response.

Session Handler classes are responsible for capturing the request and creating a session holder object for each request that holds request attributes like request headers, parameters, content. Within the operation chain there must a redirection operation that takes request attributes from session holder object, creates a HTTP connection to backend servers and sets the response again to session holder object. After a successful execution of the whole operation chain, Session Handler classes take responsibility again, receiving the response in session holder and passing it to the original connection as response.

As in the case of Session Handler classes, there are two different implementations of request redirecting (proxying) operation depending on installation of the application security layer. If installation was chosen as stand alone server depending on HTTPComponents library, then request redirecting operation was HTTPComponentsRedirectOperation and otherwise (servlet base implementation deployed on third party application server) HTTPServletRedirectOperation. In either case, request headers, request parameters and request method (if it is a POST request, request content) is retrieved from session holder and creates a HTTP connection to a server that was previously identified before during context mapping operation (3.3.3). After a response is received, response is parsed into status code, headers, content type and content and added into session holder object.

## 3.4 Organization-Wide Policy Execution

During execution of the operation chain, Eyeks consults CSAAS to decide on the login behavior, validating domain specific enterprise rules and checking for known types of web application vulnerabilities. If the request does not satisfy one of these policies, Eyeks breaks the operation chain and responds to the client with an appropriate error message.

To apply some predefined policies organization wide, Eyeks introduces some predefined operations and resources that can be secured by applying policies defined by Enterprise access and security rules. System administrators can extend these rules by adding new policies to all of these operation-resource pairs for applying new rules.

**Login Policies:** control login request to backend application. If an application requires user login, System administrators should define LOGIN_REQUEST and LOGIN operations on the context resource and should attach policies to satisfy enterprise login rules. For example, if the enterprise rules require that the users coming from some predefined location should login to the system, a policy like IPCheckPolicy that checks the IP's of user whether they are coming from a safe location must be defined and attach this policy should be attached to <[Context Name], LOGIN> pair. Or if login to a web application is only allowed during a specific time interval (e.g.: working hours), a policy,

LoginTimeCheckPolicy, can be added to < [Context Name], LOGIN_REQUEST> pair and system administrator should check the system time.

< *[Context Name], LOGIN_REQUEST*> pair is called before any login operation is done. PreAuthenticationCheckOperation is the operation responsible for constructing and sending the request to CSAAS. Login IP, account status code, last successful login time, last successful login IP, last fail login time, last fail login IP and the current number of total login trials are passed to CSAAS to be used by attached policies. As an example, three predefined policies are attached to this operation-request pair (StatusPolicy, ReLoginPeriodPolicy and MultipleLoginPolicy). All of these policies are implemented as java policy evaluator and are added to CSAAS classpath as a jar file. StatusPolicy implements status check, if the status of the user account is open account, then StatusPolicy returns an *access_allowed* decision. If the user account is locked or disabled, it put the reason message to result list and returns *access_not_allowed*. ReLoginPeriodPolicy checks relogin time period if the user's account is locked for multiple unsuccessful login tries. It checks the account status, if it is locked, and then checks last fail login time. If the user last unsuccessful login trial was 30 minutes ago, it returns access_allowed otherwise returns *access_not_allowed*. MultipleLoginPolicy decides on whether multiple logins are allowed or not by checking session parameters, if a user has an active session it denies new login try and returns *access_not_allowed*, otherwise returns *access_allowed*. These three policies are combined with FirstCoupleOrThenAnd decision combinatory, that makes the rule (StatusPolicy OR ReLoginPeriodPolicy) AND MultipleLoginPolicy) means that the user can login to the system although the account is locked because of unsuccessful login tries and if the user waits for re-login time period but in either cases multiple login tries were denied.

< *[Context Name], LOGIN*> pair is called after checks on < *[Context Name], LOGIN_REQUEST*> pair are done. AuthenticationCheckOperation is the responsible operation for constructing and sending the request to CSAAS. Authentication status and number of login trials are passed to CSAAS to be used by attached policies. For instance, two policies are attached to this pair (LoginPolicy and RBAC policy). Login policy implements the java policy evaluator and checks whether authentication result is successful or not. If successful it returns access_allowed, otherwise checks the number of login trials whether it is smaller than the allowed maximum unsuccessful login number or not. If it is smaller, it puts the reason message as UNSUCCESS to result list, otherwise puts JUST_LOCKED. The other policy was RBAC policy, which executes RBAC rule and checks user's organization hierarchy if the user has a right to login to the application, defined by context name.

**Page Request Policies:** controls all page requests to backhand applications. Eyeks defines three page request operations; SessionCheckOperation, DirectoryCheckOperation and PageCheckOperation that consults CSAAS for access decision on < [Context Name], PAGE_REQUEST>, < [Context Name], [Directory Name]>, < [Directory Name], [Page Name]> operation-resource pairs respectively. Security administrators can attach security or enterprise access control policies on these operation-

resource pairs to control application behavior. These three pairs form a hierarchy of application resources where any policy attached to < [Context Name], PAGE_REQUEST> pair will be executed organization widely, < [Context Name], [Directory Name]> controls directory specific policies and < [Directory Name], [Page Name]> pair controls page specific policies with in specific directory.

SessionCheckOperation, as described in section 3.3.4, is mainly responsible to decide the validity of user session. However it can also be used for validating organization wide policies like security policies targeting web application attacks. It operates on a generic operation PAGE_REQUEST under context name that enables for every request, EYEKS asks CSAAS for access decision. So if any policy is attached to this operation-resource pair, it is guaranteed that it will be executed organization wide.

DirectoryCheckOperation can be mainly used against directory traversal attacks; however it can also be used for applying directory specific enterprise access rules. If added to the operation chain, EYEKS will ask CSAAS for access decision giving target directory as an operation and context name as resource. This enables if the enterprise application requires organization hierarchy for access control, RBAC policy to be assigned on this operation-resource chain.

PageCheckOperation is used for executing access control policies specific to a page. If added to operation chain, EYEKS will ask CSAAS for access decision giving target page as an operation and target directory as resource. DirectoryCheckOperation and PageCheckOperation can be executed using open-world or closed-world assumption. If closed-world assumption is chosen, then every possible directory and page must be defined as an operation and every directory must also be defined as a resource. On the other hand every possible operation-resource mapping must be defined as permission to CSAAS. So for example, if a web application has M directories and for each directory it has N possible pages, then there must be M resources, M+N operations, and MxN permissions to be defined to CSAAS. For open-world assumption, it is not needed to define every possible directory and page mappings. It is enough to define directories that need to extend organization-wide policies with directory specific policies and if any page specific policy is need; it is enough to define pages that need exceptions.

These three operations and respective operation-resource pairs are not intended to be used for encapsulating enterprise (applications specific) access control rules, but for organization-wide security policies like targeting web application attacks. Best way to encapsulate application specific access control rules is mapping these policies to resources of web application as described in section 3.2, Enterprise policy mapping.

**Security Policies:** As described in section 3.2.2, Eyeks provides a common way to verify application security policies, targeting web application attacks. To be applied organization-widely, these policies must be attached to operation-resource pairs that are given page request policies. For example if any security policy is attached to < [Context Name], PAGE_REQUEST> pair, it is guaranteed that it will

be executed for every page request. So there is no need to be attached these security policies to every possible operation-resource pair.

As mentioned in section 2.4, Web application security vulnerabilities, input validation is a crucial concept to fight against application security vulnerabilities. CSAAS provides a generic security policy **SECURITY_PARAM_REG_EX_POLICY** which enables all security attributes (request parameters of a request) will be matched with regular expressions that validate the possible safe values. If this policy added to any organization-wide operation-resource pairs (like PAGE_REQUEST operation) for every request the parameters are checked against any kind of manipulation and attack.

In order to achieve this, CSAAS provides a UI where the security administrator can define every allowed parameter with its expected regular expression for a web page. For example consider a web page doeft.jsp as given in section 3.2.1 where doeft.jsp has 2 operations VIEW and SUBMIT and possible parameters of VIEW operation are USERID and ACCOUNT_INFO; possible parameters of SUBMIT operation are USERID, ACCOUNT_INFO, TRANS_ACC_INFO and TRANS_AMOUNT. Security administrator can define possible values for these parameters using regular expressions. When Eyeks consults CSAAS for PAGE_REQUEST operation, **SECURITY_PARAM_REG_EX_POLICY** evaluates the request by finding which operation and resource pair is targeted to and finds possible request parameters and corresponding regular expression and evaluates these values using Java RegEx API. If any parameter is found not to match with regular expression, CSAAS returns access_not_allowed and the request will be denied by EYEKS. Like page request policies this policy can be executed using closed or open world assumption, where in closed world assumption it is mandatory to define all parameter with their values if any other parameter is found in the request, the request will be denied. It is enough to define only critical parameter if open-world assumption is chosen.

On the other hand, a security policy, **SECURITY_INJECTION_POLICY**, is written to check all request parameters against injection type of attacks that can be extended to cover all types of possible injection such as SQL, LDAP, XML injection and XSS. Although it is not an effective implementation of injection flaw detection mechanism, it is a demonstration of how security policies targeting specific web application attack could be written and executed by Eyeks. It uses blacklist implementation, where it checks against forbidden keywords like special characters as *** ' % @! ;** < > and special keywords as **script, select, cn** that can be used for SQL, LDAP and XML injection attacks and XSS.

## 3.5 Integration with Application Servers

Application security layer can run as a stand-alone server or can integrate into any J2EE based application servers like Tomcat or JBoss. Stand-alone HTTP server is implemented using HTTPComponents library which is an open source project supported by Apache itself.

HttpComponents provides abstraction over HTTP protocol and extends *java.net* package by providing an efficient, up-to-date, and feature-rich set of components that can be used to assemble custom, standards compliant client- and server-side HTTP services. HttpComponents project strives to conform to the following specifications endorsed by the Internet Engineering Task Force (IETF):

- RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0.
- RFC 2116 - Hypertext Transfer Protocol -- HTTP/1.1.
- RFC 2117 - HTTP Authentication: Basic and Digest Access Authentication.
- Netscape Cookie Draft - Persistent Client State. (HTTP cookies, preliminary specification)
- RFC 2109 - HTTP State Management Mechanism (HTTP cookies, version 1).
- RFC 2965 - HTTP State Management Mechanism (HTTP Cookies, version 1, second revision).

The application security layer stand-alone server implementation does not provide a fully functional web server like Apache, however, it implements a multi-threaded and thread-safe HTTP proxy that intercepts a HTTP request, handles the HTTP communication between clients and backhand servers. HTTPComponent library does not implement a server but provides basic building blocks of abstracting HTTP protocol.

The following are the components of the Apache core:

**HTTP_PROTOCOL:** contains routines that directly communicates with the client (through the socket connection), following the HTTP protocol. All data transfers to the client are done using this component.

**HTTP_MAIN:** the component that startups the server and contains the main server loop that waits for and accepts connections. It is also in charge of managing timeouts.

**HTTP_REQUEST:** the component that handles the flow of the request processing, dispatching control to the modules in the appropriate order. It is also in charge with error handling.

**HTTP_CORE:** the component implementing the most basic functionality that can be used by all other components.

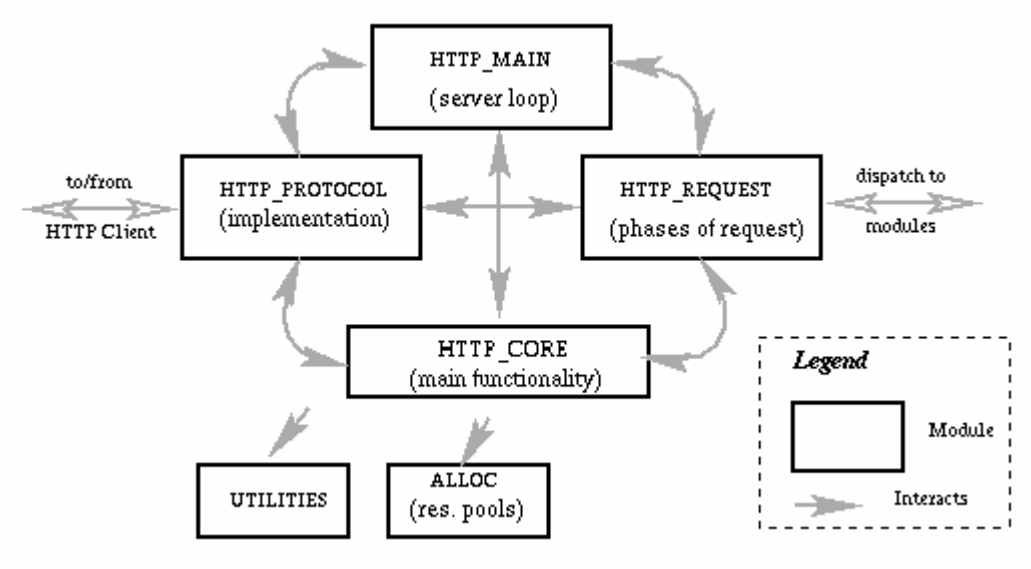


**Figure 26 Components of Apache Core**

On the other hand; HTTPComponents library does not contain HTTP_MAIN component of Apache, which implements the server and handles the connections but fully implements HTTP_PROTOCOL and HTTP_CORE components of Apache. So for this thesis, a basic implementation of a multi-threaded server that handles coming HTTP connections is implemented. In the figure, the components of the implemented HTTP server depending on HTTPComponents library are shown. HTTP_MAIN component is replaced by the application security layer main process and HTTP_REQUEST is replaced by the HTTP Request Handler, that is a worker thread that dispatches the request and response to HTTPComponentsSessionHandler object that creates SessionHolder object and passes it to OperationManager object to start the execution of the operation chain (in section 3.3.2)

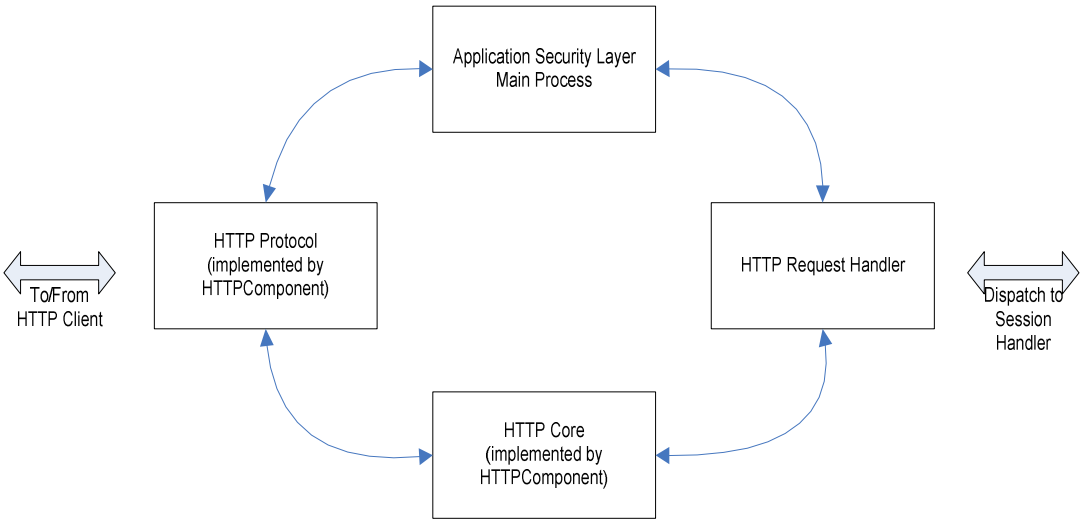


**Figure 27 Components of EYEKS Stand-Alone Server**

The other implementation, which is Servlet based, can be integrated into any kind of J2EE application server. Servlet based implementation is designed as a typical web application and consists of only one servlet, which is called as EyeksMainServlet that must be mapped to the root context. A typical web.xml for Eyeks as a web application is;

```
<web-app>

  <display-name>
   Eyeks Application Security Layer
  </display-name>
  <description>
   Application Security Layer interface for Servlet based implementations
  </description>

  <servlet>
   <servlet-name>EyeksMainServlet</servlet-name>
   <description>Main listener servlet</description>
   <servlet-class>tr.com.eyeks.securitylayer.EyeksMainServlet</servlet-class>
   <init-param>
    <param-name>configuration_folder</param-name>
    <param-value>configuration</param-value>
   </init-param>
  </servlet>

  <servlet-mapping>
   <servlet-name>EyeksMainServlet</servlet-name>
   <url-pattern>/</url-pattern>
  </servlet-mapping>

</web-app>
```

EyeksMainServlet captures all client requests and dispatches the request and response to EyeksServletSessionHandler object, which creates SessionHolder object and passes it to OperationManager object to start executing operation chain (in section 3.3.2) like in the HTTPComponent case.

In either way, StartUpService's *start* method must be called at the time of start up. In HTTPComponent based implementation, StartUpService is called in main method of SecurityLayer before any listener and worker threads are initialized and in servlet based implementation it must be called from *init* method of EyeksMainServlet. StartUpService is responsible for creating singleton instances of Manager objects within an order. StartUpService starts with initializing ConfigurationManager object which loads configuration files (which will be discusses in next section) and continues with EyeksDBManager, ContextResolver, OperationsManager and finally EyeksExceptionDispatcher in order. These managers load their configurations according to configuration files passing from ConfigurationManager's file streams to their *build* or *init methods*.

## 3.6 Managing EYEKS

The operation and behavior of Eyeks can be configured through five configuration files.

**context.properties:** As described in Context Mapping section 3.3.3, configures contexts that will be served by Eyeks. A number of context (N starts with 1) can be defined into Eyeks. This configuration file has the format:

*eyeks.contextN.name:* Name of the context, must match with path.properties and operation properties.

*eyeks.contextN.host:* IP address or DNS name of backhand server.

*eyeks.contextN.port:* In which port, does the backend application listen to.

*eyeks.contextN.protocol:* Which protocol does backend application uses (HTTP or HTTPS).

*eyeks.contextN.welcomepage:* To which path, Eyeks directs the login request after successful login operation.

A typical context configuration example is given in section 3.3.3.

**session.properties:** Configures the implementation method and storage information of user account and user session (which was described in section 3.3.4. The configuration file holds two configurations, one is user session and the other is user account. User session configuration starts with *usersession.classname* which refers the implementation method (LDAP or database), where it take full class name of the class that implements *IUserSessionDBManager* interface. Currently two different user session storage is implemented, database or LDAP. For database storage, the configuration file format is as follows:

*usersession.classname:* Full class name of user session handler

*usersession.database.driver:* Full database driver name.

*usersession.database.url:* Database URL.

*usersession.database.user:* User name to connect to database

*usersession.database.password:* Password of database user.

An example configuration for MySQL database is as follows;

```
usersession.classname = tr.com.eyeks.database.ImplUserSessionJDBCManager
usersession.database.driver =org.gjt.mm.mysql.Driver
usersession.database.url = jdbc:mysql://localhost:3306/csaas
usersession.database.user = csaas
usersession.database.password = csaas
```

User account configuration is the same as user session. The only change is that usersession tag is replaced with useraccount. So typical configuration for MySQL database becomes;

```
useraccount.classname = tr.com.eyeks.database.ImplUserAccountJDBCManager
useraccount.database.driver =org.gjt.mm.mysql.Driver
useraccount.database.url = jdbc:mysql://localhost:3306/csaas
useraccount.database.user = csaas
useraccount.database.password = csaas
```

The implementation methods of user account and user session does not have to be the same. User sessions can be stored into LDAP to improve performance, where user account can be stored in database.

**message.properties:** holds the error messages of Eyeks, where an exception is raised from any of the operation chain element to break the execution chain. This file consists of three comma separated values where the first value represents the class name of the exception, second name represents the short name of error message and the third value represents the full message to be presented to the user. The exception handling mechanism was discussed in section 3.3.2.4 and a typical example for authentication error messages is as follows:

```
tr.com.eyeks.exceptions.EyeksAuthenticationException,UNSUCCESS,Unsuccessful login try
tr.com.eyeks.exceptions.EyeksAuthenticationException,DISABLED,Account has been disabled
tr.com.eyeks.exceptions.EyeksAuthenticationException,JUST_LOCKED,Account has just locked
tr.com.eyeks.exceptions.EyeksAuthenticationException,LOCK_PASSWORD,Account lock
because of password
tr.com.eyeks.exceptions.EyeksAuthenticationException,UNKNOWNCONTEXT,Path is not
allowed!
```

**operation.properties:** is used for constructing operation chain of Eyeks, which was described in 3.3.2. It has 3 configuration lists; pre-operations list, post operations list, commands and their operations list depending on contexts. As mentioned in 3.3.2, pre and post operations will be executed regardless of the context where pre-operations are executed before any command operation and post-operations are executed after executing all command operations. The format of pre-operations in configuration file is;

*preOperations.N:* The operation class that implements IEyeksOperationChain Element interface. Where N starts with N and goes to a number of pre-operations in order. The format of post-operations is the same as pre-operations but the corresponding tag is postOperations.

After configuring pre and post operations, the commands and their operations must be configured. Commands can be configured using fallowing syntax.

*[NameForOperationList].contextN.name:* Context name for command class (Context names must be defined previously in context.properties)

*[NameForOperationList].contextN.commandClass:* Full name of the responsible command class that implements *IOperationCommand*.

And operations can be added to defined command as follows:

*[NameForOperationList].contextN.operationList.M:* Full name of the operation class that implements *IEyeksOperationChainElement*.

So from these constructs, a number of responsible commands can be defined for each context defined previously and a number of operations can be attached to each command. A typical example of operations.property file is given below.

*preOperations.1=tr.com.eyeks.operations.chain.pre.ContextResolveOperation*
*postOperations.1=tr.com.eyeks.operations.chain.post.HeaderReverseRedirectionOperation*

*loginOperations.context1.name=DynamicContext*
*loginOperations.context1.commandClass=tr.com.eyeks.operations.LoginCommand*
*loginOperations.context1.operationList.1=tr.com.eyeks.operations.chain.login.PreAuthenticatio nCheckOperation*
*loginOperations.context1.operationList.2=tr.com.eyeks.operations.chain.login.AuthenticationCh eckOperation*
*loginOperations.context1.operationList.3=tr.com.eyeks.operations.chain.login.UserLoginOperat ion*
*loginOperations.context1.operationList.4=tr.com.eyeks.operations.chain.login.FetchWelcomePa geOperation*
*loginOperations.context1.operationList.5=tr.com.eyeks.operations.chain.pagerequest.HttpComp onentsRedirectOperation*
*loginOperations.context1.operationList.6=tr.com.eyeks.operations.chain.pagerequest.HeaderMa nagedTokenPutOperation*

*pageRequestOperations.context1.name=DynamicContext*
*pageRequestOperations.context1.commandClass=tr.com.eyeks.operations.PageRequestComman d*
*pageRequestOperations.context1.operationList.1=tr.com.eyeks.operations.chain.pagerequest.Pa rameterListPrintOperation*
*pageRequestOperations.context1.operationList.2=tr.com.eyeks.operations.chain.pagerequest.He aderManagedTokenGetOperation*
*pageRequestOperations.context1.operationList.3=tr.com.eyeks.operations.chain.pagerequest.Ses sionCheckOperation*
*pageRequestOperations.context1.operationList.4=tr.com.eyeks.operations.chain.pagerequest.Htt pComponentsRedirectOperation*
*pageRequestOperations.context1.operationList.5=tr.com.eyeks.operations.chain.pagerequest.He aderManagedTokenPutOperation*

*logoutOperations.context1.name=DynamicContext*
*logoutOperations.context1.commandClass=tr.com.eyeks.operations.LogoutCommand*
*logoutOperations.context1.operationList.1=tr.com.eyeks.operations.chain.pagerequest.HeaderM anagedTokenGetOperation*
*logoutOperations.context1.operationList.2=tr.com.eyeks.operations.chain.pagerequest.SessionC heckOperation*
*logoutOperations.context1.operationList.3=tr.com.eyeks.operations.chain.logout.SessionDelete Operation*
*logoutOperations.context1.operationList.4=tr.com.eyeks.operations.chain.logout.RedirectLogout Operation*

Where three commands (Login, PageRequest and Logout) are defined for context, *DynamicContext,* and for login command six, for page request command five and for logout o command four operations were attached.

**path.properties:** As described in section 3.3.3, Context Mapping, this file holds mappings from real paths of the contexts to Eyeks paths that will be served. The entities in this file has format like *“EyeksPath” => “context name”,”realpath”*. The sample configuration was given in section 3.3.3 in details.

## 3.7 Verification of Solution

Verifying a security product is a hard and most probably an impossible job, because the verification method mainly depends on security testing. Security testing, by itself, isn't a particularly good measure of how secure an application is, because there are an infinite number of ways that an attacker might be able to make to break an application, and it isn't simply possible to test all of the possibilities. However, security testing has the unique power to absolutely show that there is a problem.

There are mainly three design considerations of Eyeks; encapsulating domain specific factors (enterprise rules) to decide on access decision, apply these rules organization-widely and transparently and secure application from web application attacks.

Encapsulating domain specific factors have been discussed in 2.2.6 and 2.5.1 sections in detail. An access control mechanism, depending on RAD specification, was shown to be one of the best ways to take access decision mechanism out of application. Considering this, CSAAS has been implemented as the access control mechanism with some improvements over RAD. Access decisions can easily be deployed on CSAAS as shown in section 3.2.

To guarantee enterprise rules to be executed organization-widely, Eyeks has been implemented as a separate layer which will be deployed in front of web application and control all access to backhand applications. Access decisions and security aspects are executed transparently on this layer. No direct connection is allowed from client to backhand applications so that Eyeks will be an application gateway, HTTP proxy for backhand applications.

Eyeks has also been designed to confront various web application attacks and also can be extended for future attacks. However, verifying this feature requires a well structured and organized security testing. OWASP [9] has released a security testing guide, which can be used for a base-line to construct security testing. In this section, security test sets and what countermeasures, what aspects have been considered to secure web applications will be presented.

In this thesis, OWASP Testing Guide 2007 V.2.0 release candidate 1 document was used to generate test sets and to verify the solution. The test sets of this document are given in *table 11*.

**Table 11 OWASP Testing List**

| **Category** | **Ref. Number** | **Name** |
|---|---|---|
| Information Gathering | OWASP-IG-001 | Application Fingerprint |

**Table 11 (continued)**

| | | |
|---|---|---|
| | OWASP-IG-002 | Application Discovery |
| | OWASP-IG-003 | Spidering and googling |
| | OWASP-IG-004 | Analysis of error code |
| | OWASP-IG-005 | SSL/TLS Testing |
| | OWASP-IG-006 | DB Listener Testing |
| | OWASP-IG-007 | File extensions handling |
| | OWASP-IG-008 | Old, backup and unrefered files |
| Business logic testing | OWASP-BL-001 | Testing for business logic |
| Authentication Testing | OWASP-AT-001 | Default or guessable account |
| | OWASP-AT-002 | Brute Force |
| | OWASP-AT-003 | Bypassing authentication schema |
| | OWASP-AT-004 | Directory traversal/file include |
| | OWASP-AT-005 | Vulnerable remember password and pwd reset |
| | OWASP-AT-006 | Logout and Browser Cache Management Testing |
| Session Management | OWASP-SM-001 | Session Management Schema |
| | OWASP-SM-002 | Session Token Manipulation |
| | OWASP-SM-003 | Exposed Session Variables |
| | OWASP-SM-004 | Session Riding |
| | OWASP-SM-005 | HTTP Exploit |
| | OWASP-DV-001 | Cross site scripting |
| | OWASP-DV-002 | HTTP Methods and XST |
| | OWASP-DV-003 | SQL Injection |
| | OWASP-DV-004 | Stored procedure injection |
| | OWASP-DV-005 | ORM Injection |
| | OWASP-DV-006 | LDAP Injection |
| | OWASP-DV-007 | XML Injection |
| | OWASP-DV-008 | SSI Injection |
| | OWASP-DV-009 | XPath Injection |
| | OWASP-DV-010 | IMAP/SMTP Injection |
| | OWASP-DV-011 | Code Injection |
| | OWASP-DV-012 | OS Commanding |
| | OWASP-DV-013 | Buffer overflow |
| | OWASP-DV-014 | Incubated vulnerability |
| Denial of Service Testing | OWASP-DS-004 | Writing User Provided Data to Disk |
| | OWASP-DS-005 | Failure to Release Resources |
| | OWASP-DS-006 | Storing too Much Data in Session |
| | OWASP-WS-001 | XML Structural Testing |
| | OWASP-WS-002 | XML content-level Testing |
| | OWASP-WS-003 | HTTP GET parameters/REST Testing |
| Web Services Testing | OWASP-WS-001 | XML Structural Testing |
| | OWASP-WS-002 | XML content-level Testing |
| | OWASP-WS-003 | HTTP GET parameters/REST Testing |

**Table 11 (continued)**

| | | |
|---|---|---|
| | OWASP-WS-004 | Naughty SOAP attachments |
| | OWASP-WS-005 | Replay Testing |
| AJAX Testing | OWASP-AJ-001 | Testing AJAX |

And counter measures to prevent these vulnerabilities are shown below.

**Application Fingerprint:** Eyeks has two different implementations, as a stand-alone server and as a deployment on any J2EE servers. In either case, adding HideApplicationHeaderOperation to the operation chain will remove backhand web application fingerprints from headers and shuffle headers. However if deployed on java application servers, fingerprints of web server (the application server that Eyeks deployed on.) are still one and can be revealed.

**Application Discovery:** Due to Eyeks's context mapping mechanism (section 3.3.3). Eyeks can serve more than one web application as if there is only one. So it hides backhand applications. If not intentionally deployed on front hand servers, there is no way to discover other applications.

**Spidering and Googling:** Eyeks's layered structure (3.3.1) and context mapping (3.3.3) are designed to hide backhand web application from spidering and googling. Google can only reveal virtual paths and names not real paths.

**Analysis of error code:** Exception handling mechanism (3.3.2.4) has been designed to confront information disclosure by error codes. Any kind of exceptions (due to backhand applications and as well as inner exception of Eyeks) have been caught and converted to generic error page.

**SSL/TLS Testing:** No countermeasure to confront attacks about SSL/TLS has been implemented.

**DB Listener Testing:** Eyeks has been designed only for web application layer attacks. DB Listener testing is out of scope.

**File Extensions Handling:** Although a virtual path concept has been implemented, no operation has been implemented to hide file extensions. However, a new operation that holds file extension mappings and hides them from client can easily be implemented and due to Eyeks operation chain mechanism (3.3.2), can easily be added to Eyeks.

**Old, Backup and Unreferenced Files:** Due to the layered structure (3.3.1), context mapping (3.3.3) and page request policies were executed using closed world assumption (3.4). There is no chance to guess and fetch unreferenced files without intentionally mapped to Eyeks.

**Testing for Business Logic:** There is no way to validate if business logic of an application has errors. However CSAAS enables to encapsulate business access logic from application code and provides more manageable and error-prone implementation. And Eyeks makes the request to validate enterprise rules organization-widely.

**Default or Guessable Account:** Eyeks does not hold user account or user passwords; however implements the authentication mechanism using CSAAS. CSAAS can use legacy system to access user account. Therefore, guessable user accounts are application responsibility.

**Brute Force:** As guessable accounts, the strength of user passwords is application responsibility. However, Eyeks provides an account locking mechanism to prevent brute force attacks (3.3.4).

**Bypassing authentication schema:** By default, Eyeks uses form based authentication to confront complex authentication mechanisms. All authentication requests are passed to CSAAS and it can be used as a bridge between application legacy authentication mechanism and Eyeks. Session tokens (3.3.4) and login policies (3.4) both target authentication bypassing attacks.

**Directory traversal/file include:** Eyeks tackles directory traversal attacks by organization-wide policies (page request policy) (3.4) and context mapping (3.3.3). All requests are catch by application security layer and passed to CSAAS to authorize. DirectoryCheckOperation and PageCheckOperation try to eliminate directory traversal attacks.

**Vulnerable remember password and pwd reset:** Browser caching is automatically turned off by Eyeks and on the other hand CSAAS has password reset and security questions mechanism however by default, there are not used.

**Logout and Browser Cache Management Testing:** Eyeks chain operations have logout command (3.3.2.1) which manages logout operations of whole system. Eyeks session and session tokens are become invalid after logout operation.

**Session Management Schema:** Eyeks session is handled through Eyeks session management schema (3.3.4). Three kinds of session management have been implemented; Header based, cookie based and within the content itself.

**Session Token Manipulation:** Session tokens hold user id, timestamp, sequence number of the request and a random variable in a serialized form with encryption. Non-predictable, non-generatable tokens are used.

**Exposed Session Variables:** Reusing session tokens is not allowed by Eyeks. Session token has a sequence number and a timestamp so that every new request invalidates the previous token.

**Session Riding:** Form based authentication with carrying session token within the content removes the risk of session riding. Eyeks's ContentPutTokenOperation and ContentGetTokenOperation handles carrying session token within the content.

**HTTP Exploit:** Eyeks's HeaderCheckOperation checks every header of the request and response for invalid header values. Also by using HeaderCheckOperation, it is also possible to define every allowed header that is checked for each request.

**Injection Attacks:** (Cross site scripting, XST, SQL, stored procedure, ORM, LDAP, XML, SSI, XPath, IMAP/SMTP, Code, Command injection) Mapping each resources (pages) to CSAAS (3.2), provides a common way to validate each parameter against injection attacks. Using the closed world assumption and page request policies (3.4), it is possible to write regular expressions to validate user inputs. On the other hand a common injection check policy has been written and if added to organization-wide resource-operations pairs (3.4,) every request is checked against injection vulnerabilities using black-list of known attack vectors. It is also possible to implement specialized policies against each injection attack and easily added to CSAAS as organization-wide policy.

**Locking Customer Accounts:** Eyeks's session mechanism has a temporary locking mechanism that can be controlled by session operation and policies (3.3.4).

**User Specified Object Allocation:** Eyeks has nothing to do with this vulnerability. It is each web application responsibility to manage object allocation.

**User Input as a Loop Counter:** Eyeks has nothing to do with this vulnerability. It is each web application responsibility to manage application logic.

**Writing User Provided Data to Disk:** Eyeks has nothing to do with this vulnerability. It is each web application responsibility to manage disk operations.

**Failure to Release Resources:** Eyeks has nothing to do with this vulnerability. It is each web application responsibility to manage releasing resources.

**Storing too Much Data in Session:** Eyeks has nothing to do with this vulnerability. It is each web application responsibility to manage session data.

**Web Services Testing:** Eyeks does not support web services.

**AJAX Testing:** Eyeks does not support AJAX.

# CHAPTER 4

# EXPERIMENTAL STUDY

EYEKS can be evaluated using four aspects; performance, capability of encapsulating access policies, capability of eliminating web attacks. In this thesis two experimental results will be presented. First results have been collected from a running real life system which has been using EYEKS as an application security layer. These results help us to evaluate performance of EYEKS on a highly loaded system and besides show how real life access problems can be solved by EYEKS. The second evaluation will be done on the test results taken from a test platform where EYEKS has been used as an application security layer to control access on a simple test web application, implemented using Java JSP and Struts technology. The main aim of this experiment is evaluating EYEKS against web application attacks. A vulnerability test set has been prepared using OWASP Testing Guide which was introduced in section 3.7 and applied to this test platform.

## 4.1 Case Study: Real Life System

EYEKS has been implemented and used for one of the biggest e-government projects of Turkey. The system became online on October 2004, and is being used for nearly 2.5 years. The work and the results were presented in an International Conference on Security of Information Networks (SIN 2007) [7]. This earlier version of EYEKS differs only for some concepts from the version presented in the thesis. The only changes are; Request/Response operation chain has been newly implemented to provide a more generic framework, previous version can only be deployed on Java application servers, however for this thesis, a stand-alone server was implemented.

Project was started with 13,466 registered users and by January 2007, 181,747 users have been registered. Because of the business domain of the application, the number of login and page requests is irregular and differs a lot from month to month. The application executes mainly one business transaction, which consists of five successful page requests and HTML form posts and corresponding database operations.

Project consists of four different web applications that use EYEKS as an application security layer. These four applications are defined to EYEKS as different contexts with proper mappings so that EYEKS can serve to all of them. EYEKS was installed to 3 servers. Two machines have four Solaris Ultra SPARC CPU with 8GB Ram and one machine have two Solaris Ultra SPARC CPU with 2GB Ram. On the other hand all of the backhand applications run on six Solaris Ultra SPARC CPU machine with 8GB Ram. As soon as the system was launched, the registered users and usage statistics are increased rapidly and still continue to increase.

Table 12 shows monthly statistics of the number of registered users, executed transactions, login and page requests.

**Table 12 Monthly Statistics of the Real Life System**

| MONTHS | USERS | TRANSACTIONS | LOGIN | PAGE REQUEST |
|---|---|---|---|---|
| OCTOBER [2004] | 13,446 | 34,534 | 131,448 | 975,481 |
| NOVEMBER [2004] | 15,811 | 91,465 | 237,809 | 2,648,826 |
| DECEMBER [2004] | 18,638 | 174,725 | 532,658 | 5,124,684 |
| JANUARY [2005] | 20,984 | 452,782 | 1,530,232 | 13,592,516 |
| FEBRUARY [2005] | 23,403 | 380,682 | 1,310,556 | 11,401,426 |
| MARCH [2005] | 41,519 | 955,901 | 3,058,883 | 27,606,421 |
| APRIL [2005] | 53,342 | 1,511,975 | 3,931,135 | 43,937,994 |
| MAY [2005] | 55,642 | 1,780,000 | 5,615,023 | 49,786,600 |
| JUNE [2005] | 56,543 | 1,004,445 | 3,022,113 | 23,735,035 |
| JULY [2005] | 57,250 | 1,728,969 | 5,390,383 | 39,576,100 |
| AUGUST [2005] | 58,037 | 2,053,585 | 4,115,414 | 48,649,429 |
| SEPTEMBER [2005] | 59,585 | 1,138,612 | 2,618,807 | 26,176,690 |
| OCTOBER [2005] | 60,271 | 2,019,727 | 4,440,552 | 50,735,542 |
| NOVEMBER [2005] | 60,903 | 2,218,907 | 5,000,212 | 57,602,826 |
| DEC [2005] | 61,607 | 1,243,317 | 2,565,332 | 28,745,489 |
| JANUARY [2006] | 62,555 | 2,184,729 | 5,001,502 | 58,157,486 |
| FEBRUARY [2006] | 64,549 | 2,518,218 | 5,113,201 | 69,830,185 |
| MARCH [2006] | 65,794 | 2,346,147 | 5,006,541 | 61,210,975 |
| APRIL [2006] | 65,950 | 2,717,194 | 5,911,656 | 76,054,260 |
| MAY [2006] | 66,069 | 2,865,000 | 6,411,211 | 84,889,950 |
| JUNE [2006] | 87,938 | 1,005,555 | 2,156,987 | 23,982,487 |
| JULY [2006] | 88,821 | 1,730,000 | 3,929,987 | 43,180,800 |
| AUGUST [2006] | 89,112 | 2,056,987 | 4,419,877 | 52,103,481 |
| SEPTEMBER [2006] | 96,004 | 1,100,562 | 2,409,652 | 25,312,926 |
| OCTOBER [2006] | 120,432 | 2,001,532 | 4,066,579 | 52,039,832 |
| NOVEMBER [2006] | 126,245 | 3,124,236 | 6,910,198 | 63,808,771 |
| DECEMBER [2006] | 150,324 | 1,245,330 | 2,776,630 | 26,837,801 |
| JANUARY [2007] | 181,747 | 1,010,336 | 2,062,146 | 25,591,811 |

Following figures show monthly distribution of executed transactions, login and page requests

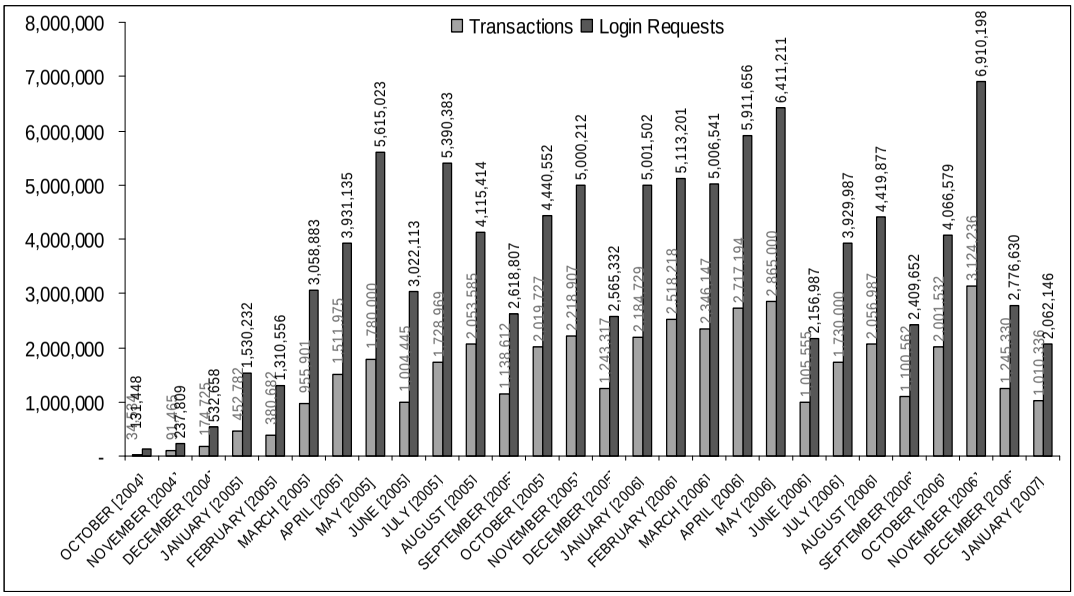


**Figure 28 Distribution of Transactions and Login Requests**

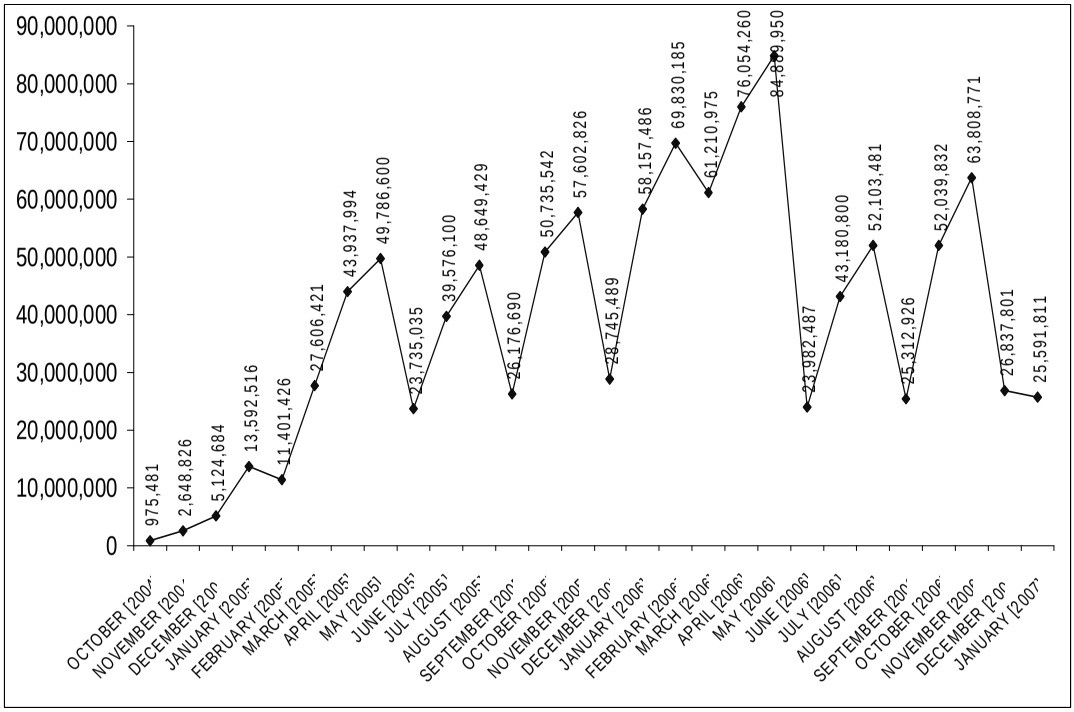


**Figure 29 Distributions of Page Requests**

As shown in the graphics, the distribution of page request and login numbers varies a lot and are totally irregular, some months like April, May and November takes two or three time more traffic than previous months like June, July and September. *Table 13* gives statistics about how many business transactions; login and page request has been done per month in last year respectively. Last column stands for the total request numbers on a peek day.

**Table 13 Average and Peek Statistics**

| | **Start** | **Average** | **Peek** | **Peek Day** |
|---|---|---|---|---|
| **Transactions** | 34,534 | 1,856,324 | 3,124,236 | 649,024 |
| **Login** | 131,448 | 3,559,883 | 6,910,198 | 833,670 |
| **Page Request** | 975,481 | 39,046,279 | 63,808,771 | 4,128,295 |

The irregularity of monthly distribution is also true for days in a months, next figure shows daily distribution of transaction numbers. In fact, the overall traffic is concentrated in the third week of the months where it takes nearly 70-80% of monthly traffic. This is because of the business of application. Business rules require deadliness for some business tractions in a month so that the traffic increases rapidly in the last week of deadlines.

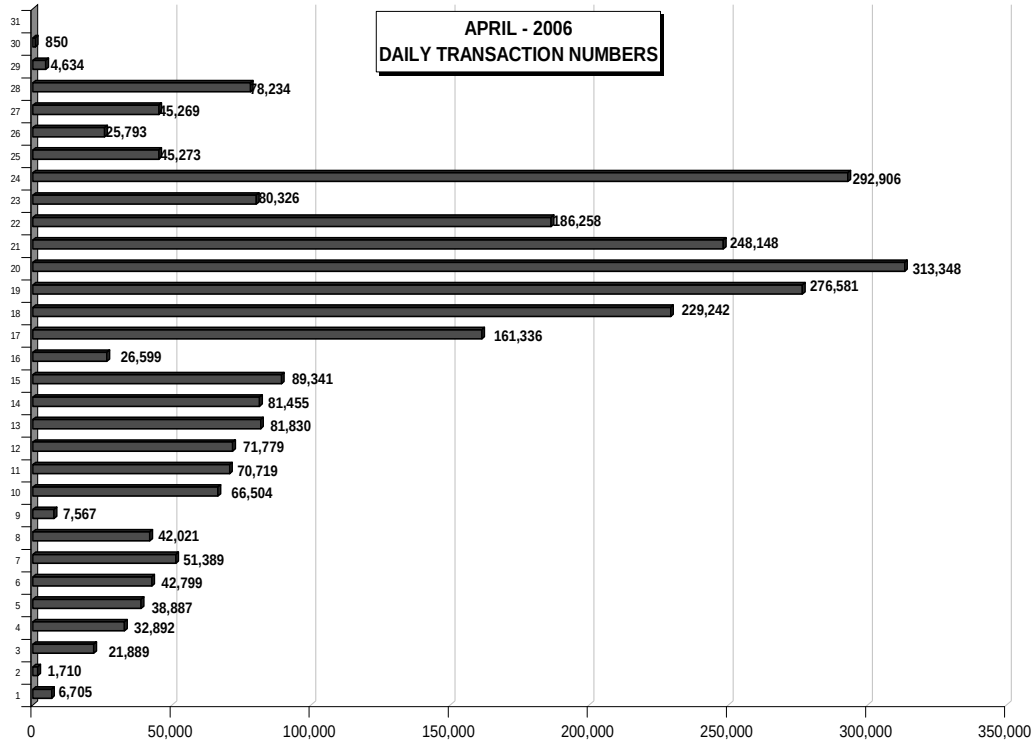


**Figure 30 Daily Transactions (April 2006)**

So the performance issue on high loads is critical for both the backhand application and EYEKS. The system must scale well to handle with high traffic in a responsible time and also must highly available. The users of the system also increase rapidly, within 2 year period, the number of users multiplied by ten and the increasing number of users still continues. So overall system must also handle with increasing number of users as well as traffic burst on some days of the month.

EYEKS scales and responses very well with this situation, the statistics show that on even peek days, average peek CPU usage has not been over 27 % where backend servers (8 servers) usage is 92 %.

The whole application (four different web applications) has been written in Java using J2EE technology. All four applications uses MVC pattern so that there are only one controller servlet for each application. The resource and operation mappings are done according to rule 4 which was described in section 3.2. Each application has been defined as resources of the system and so that 4 resources has been defined. The operations are mapped as allowed actions of each controller servlet, which consist of totally 67 operations. The system mainly uses RBAC policy evaluator for which 8 different roles are defined hierarchically according to business needs. Other from RBAC policy, enterprise security rules of project has been implemented using 11 user-defined policies. The policies

are mainly time based that defines deadlines of business transactions and also defines sequence of successfully executed user action to access more secure resource. These policies are mapped to 210 different resource-operation mappings as permissions that cover whole application.

EYEKS has also been used as authentication mechanism that provides single-sign-on for whole application. Form based authentication has been used to verify user passwords. One-time passwords has also been generated and verified for more critical operations. The authentication mechanism also provides authentication based on security questions that have been used for integration with call center application. EYEKS also can authenticate users coming from IVR (Interactive Voice Response) application so that call center operator can use the system in place of the client for assistance.

The session management of EYEKS depends on LDAP implementation in this case, where user session attributes are stored to LDAP after successful login operation. On each request, user session is fetched from LDAP using DN of the user that is stored in encrypted EYEKSTOKEN. Token's are stored within each web page and are checked for validity for each request as described in section 3.3.4.

There are no application level security policies to check for known security exploits are implemented however every parameter of each web page is well-defined and defined in CSAAS using the closed world assumption. The only place that checks for web application attacks are authorization mechanism where the login requests are checked for common injection attacks such as SQL and LDAP injection. However the system is secured for directory traversal, information disclosure, broken authentication and session management types of attacks by default. EYEKS logs also showed that in last 3 months (November, December 2006 and January 2007), total number of 865,327 requests than and as well as, 938,787 incorrect password tries per mount are found to be malicious and denied.

The experience with running real life system proves that EYEKS provides great benefits to enterprise applications. First of all frees whole application from embedding access decision rules in application source code that improves manageability of the system. The access decision rules can be added changed or removed dynamically without changing application code and removes the need of redeployment. On the other hand provides a secure authentication and authorization mechanism that covers whole application with no additional effort. The experiment also shows that EYEKS was very scalable and gives high performance under heavily loads.

## 4.2 Experiment 1: Artificial Load Tests

The real life system described in previous section has been also tested under artificial load tests. The testing tool has been chosen as Apache JMeter, which is a 100% pure Java desktop application designed to load test functional behavior and measure performance. Apache JMeter can be used to test performance both on static and dynamic resources (files, Servlets, Perl scripts, Java Objects). It can be

used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types.

These load tests consist of 12 successive system scenario operations to execute a business transaction starting from login request to logout request and tries to mimic typical user behavior. Thirteen different load tests are constructed for each targets number of 20 to 500 concurrent users. These tests run on the system with EYEKS and without EYEKS to compare the payload of EYEKS. The statistics are shown in table.

**Table 14 EYEKS Performance Statistic**

| Concurrent Users | Avg. Execution time with EYEKS (s) | Avg. Execution time without EYEKS (s) | Payload |
|---|---|---|---|
| 20 | 0.629 | 0.586 | 0.074246 |
| 50 | 1.577 | 1.468 | 0.074148 |
| 70 | 2.219 | 2.065 | 0.074319 |
| 100 | 3.202 | 2.980 | 0.074527 |
| 130 | 4.247 | 3.952 | 0.074765 |
| 160 | 5.438 | 5.058 | 0.075064 |
| 200 | 7.276 | 6.765 | 0.075590 |
| 250 | 18.205 | 16.913 | 0.076421 |
| 300 | 44.629 | 41.402 | 0.077950 |
| 350 | 108.014 | 99.986 | 0.080288 |
| 400 | 261.435 | 241.109 | 0.084302 |
| 450 | 638.744 | 585.894 | 0.090204 |
| 500 | 1582.747 | 1438.696 | 0.100126 |

The backhand application (without EYEKS) scales well until the number of 250 concurrent users. However after 250 users, the response time becomes increasing exponentially and when the system has 500 concurrent users the average execution time for a business transaction becomes 24 minutes. The corresponding graphics will show this behavior.

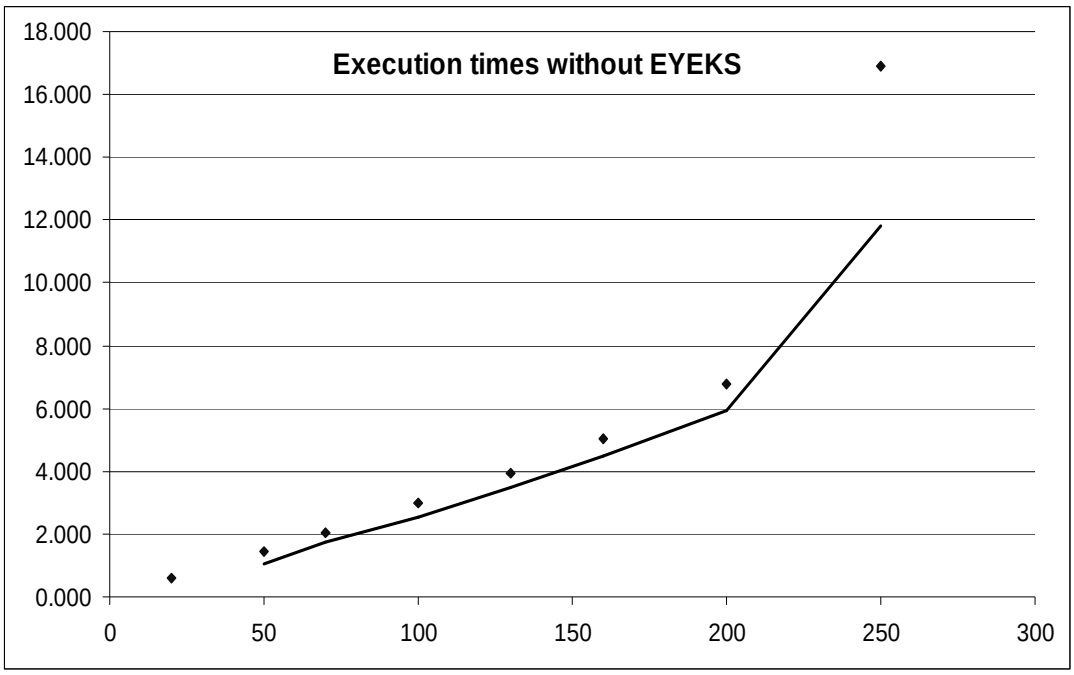


**Figure 31 Execution Times Without EYEKS (0-300)**

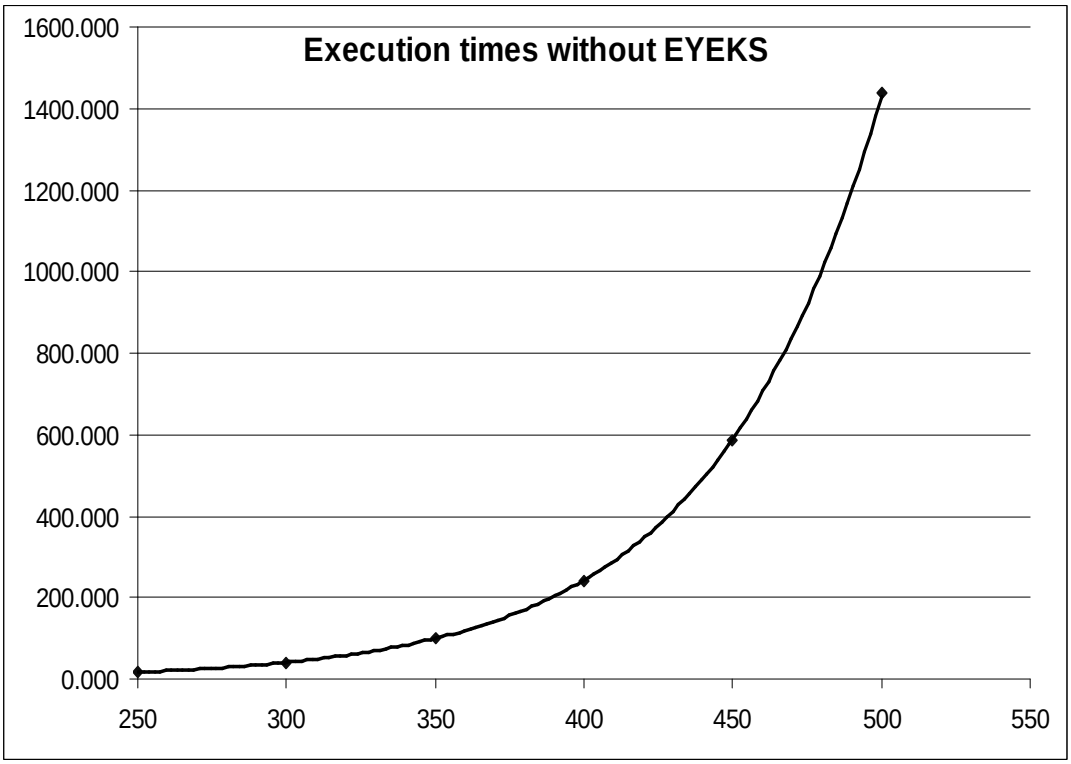


**Figure 32 Execution Times Without EYEKS (250-500)**

On the other hand, the average payload of EYEKS is very stable on increasing number of concurrent users. The execution times of business transaction suffer only 8 % if EYEKS was installed in front of web applications. EYEKS scales very well where at 20 concurrent users the payload was 7.4 % and at 500 concurrent users the payload only increases to 10%. The following graphic shows the payloads over number of concurrent users.

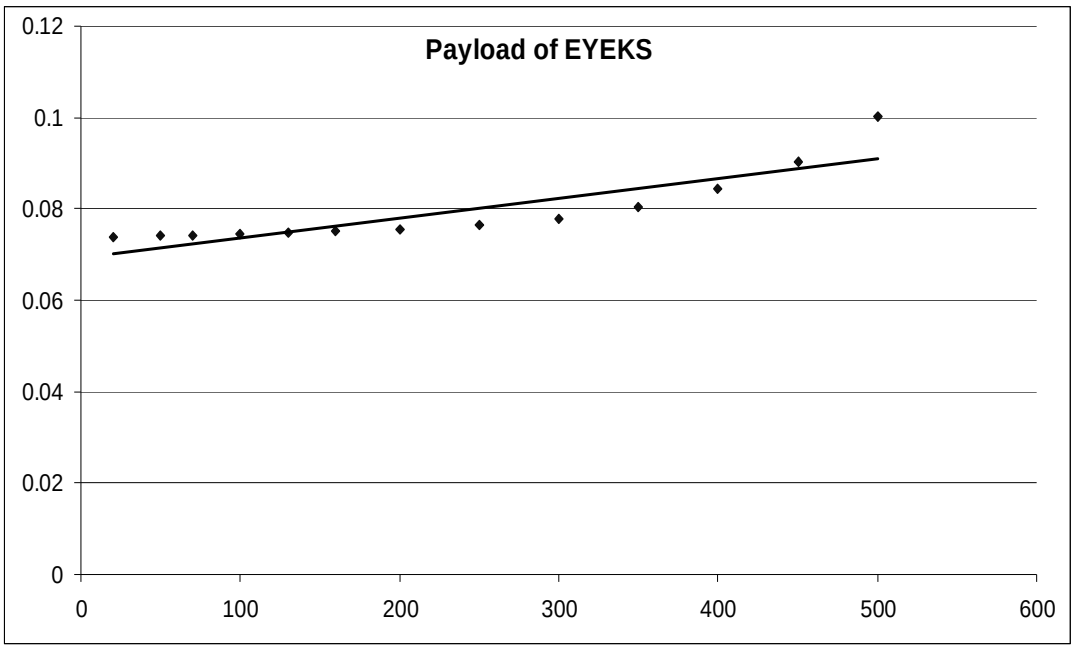


**Figure 33 Payload of EYEKS**

This high level of scalability is because of CSAAS's caching mechanism. After the first execution of business scenario, all necessary elements of RAD specification to decide on access decision like operations, resources and policies have all been fetched and cached. The only time consuming operation for CSAAS was retrieving user role (due to RBAC policy). The remaining evaluation is done in memory so that no other I/O operation has been needed. After each user's page request, the roles of these roles will be cached so that for the remaining operations, no access to database has been needed.

EYEKS has been using connection pooling with persistent HTTP connections that allow pipelining of client request. So that HTTP connection establishment occurs very less. The payload is due to I/O operations from these sockets and mostly RMI communication between EYEKS and CSAAS

## 4.3 Experiment 2: Testing Against Web Application Attacks

In this section, the solidity of EYEKS against web application attacks will be investigated. For this purpose, an open source web application that is available publicly has been chosen and EYEKS was installed in front of it as an application security layer.

The chosen web application was ADF Toy Store Demo application [75], which is realized by Oracle to demonstrate their newly build framework called Oracle Application Development Framework. This is a basic online shopping application for toy stores; it allows user login, listing of products under toy categories, searching for a specific product, online ordering and shipment. The reasons behind choosing ADF Toy Store application are; it reflects all functionalities of a typical online shopping application, considerably simple application so it is easy to configure, it is a public open source application and it is made up with latest technologies such as ADF and Java Server Faces (JSF).

ADF Toy Store application has been implemented using Java with Model/View/Controller (MVC) design pattern. It is implemented using two existing J2EE application frameworks: Apache Struts and Oracle Application Development Framework (ADF). Struts has been used as controller, ADF has been used to implement model. View layer has used standard Struts and JSTL tag libraries as well as JSF to simplify building the web UI

### 4.3.1 Test Environment and Setup

ADF Toy Store application has been ported to Oracle JDeveloper 10.1.3.3 and run on embedded OC4J application server with Java JRE 1.5.0_06. Application deployed on /ADFToyStore web context and run on port 8988.ADF Toy Store uses Oracle database by default so Oracle 10g Express Database has been installed and configured accordingly.

EYEKS has been installed as stand-alone server with Java JRE 1.5.0_06. For session management database implementation has been chosen and so MySQL Server 5.0 has been installed and configured accordingly.

ADF Toy Store can be deployed as a WAR or EAR file and from client of view, it has four directories; *faces* holds jsp pages, *images* holds image files, *templates* holds template files and *adf* holds java-script and configuration files need by JSF.

Two different Eyeks contexts have been defined and mapped as described in section 3.3.3. StaticContext is the default context and mapped to mytoystore web context, *faces* directory is mapped to /mytoystore/faces directory within ApplicationContext and *templates* directory is mapped to ScriptContext. Corresponding mappings are shown below.

/ADFToyStore → /mytoystore (StaticContext)
/staticfiles (StaticContext)
/images → /images (StaticContext)
/templates → /templates (ScriptContext)
/adf → /scripts (StaticContext)
/faces → /faces (ApplicationContext)

For StaticContext, HttpComponentsRedirectOperation was the only operation that has been added to operation chain. Since this context only holds static context like images or templates, no other security checks like token and content filtering, authorization, authentication or session management was required.

ScriptContext can be under more security risks like cross-site scripting, session riding attacks so before HttpComponentsRedirectOperation, PageCheckOperation has been added to operation chain with closed world assumption. As mentioned in section 3.4. PageCheckOperation ask CSAAS for permission on < [Directory Name], [Page Name]>. Since there is only one directory and three script files in this context. adf has been added to CSAAS as a resource and necessary scripts and cascading style sheets are added as operation as can be seen in *table 15*.

ApplicationContext's operation chain has been more complicated since it needs authentication, authorization and session management. Whole chain of operations for pre/post operations and commands are given in the table below.

**Table 15 Example Operation Chain**

| | |
|---|---|
| **Pre Operations** | ContextResolveOperation<br>HeaderCheckOperation |
| **Login Command** | PreAuthenticationCheckOperation<br>AuthenticationCheckOperation<br>UserLoginOperation<br>FetchWelcomePageOperation<br>HttpComponentsRedirectOperation<br>HeaderManagedTokenPutOperation |
| **Page Request Command** | HeaderManagedTokenGetOperation<br>SessionCheckOperation<br>HttpComponentsRedirectOperation<br>HeaderManagedTokenPutOperation |
| **Logout Command** | LogoutOperation |
| **Post Operations** | HeaderReverseRedirectionOperation |

ADF Toy Store application consists of 17 JSP pages and these pages must be mapped to CSAAS in order to be used for enterprise access control and application security. The best mapping strategy can be fourth strategy which suggests context name as resource and each page name as possible operations on that resource must be defined to CSAAS. On the other hand some other resource and operation pairs, as well as, policy mappings must be defined as described in section 3.4. A full set of resources, operations and attached policies will be given in *table 16*.

**Table 16 Example Resource- Operation and Policy Mappings**

| **Resource** | **Operation** | **Policy** |
|---|---|---|
| Adf | /styles/oracle-desktop-en-gecko.css<br>/jslib/CommonFormat.js<br>/jslib/CoreFormat.js<br>/jslib/DataFormat.js<br>/jslib/DataFiels.js<br>/jslib/CharSets.js | |
| ApplicationContext | LOGIN_REQUEST | StatusPolicy,<br>ReLoginPeriodPolicy<br>MultipleLoginPolicy |
| | LOGIN | LoginPolicy<br>RBAC policy |
| | PAGE_REQUEST | SECURITY_PARAM_REG_EX_POLICY<br>SECURITY_PARAM_REG_EX_POLICY<br>SECURITY_INJECTION_POLICY |
| | accountcreated.jsp<br>accountupdated.jsp<br>confirmshoppinginfo.jsp<br>editaccount.jsp<br>help.jsp<br>home.jsp<br>index.jsp<br>register.jsp<br>search.jsp<br>showcategory.jsp<br>showproduct.jsp<br>showproductdetails.jsp<br>signin.jsp<br>thankyou.jsp<br>yourcart.jsp | |

### 4.3.2 Test Tools

**CAL9000:** All security testing activity is handled by CAL9000 tool [76]. CAL9000 is one of the OWASP projects that are a collection of web application security testing tools that complements the feature set of automated scanners. CAL9000 is written in Javascript and provides flexibility and functionality for more effective manual testing efforts. It mainly targets XSS attacks which collects XSS attack signatures from RSnake [77] and also provides character encoder/decoder, manually crafting and sending HTTP requests to servers (GET, POST, HEAD, TRACE, TRACK, OPTIONS, CONNECT, PUT, DELETE, COPY, LOCK, MKCOL, MOVE, PROPFIND, PROPPATCH, SEARCH and UNLOCK methods supported), sending single requests or launch automated attacks with more than one request at a time, viewing the status codes, response headers and body, isolating the script, form and cookie information in the response, IP Encoder/Decoder and string generator.

**Httprint:** is another testing tool to test web server fingerprints. It relies on web server characteristics to accurately identify web servers [78].

**Wget:** is used for web content crawling [79].

**DirBuster:** is a multi threaded java application designed to brute force directories and files names on web/application servers [80].

**NMap:** Nmap ("Network Mapper") is a free and open source utility for network exploration or security auditing. It is also useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime [81].

**WebScarab:** WebScarab is a framework for analyzing web applications. It is written in Java, WebScarab has several modes of operation, implemented by a number of plug-ins. Some usable features are extracting Scripts and HTML comments from HTML pages, observing traffic between the browser and the web server, allowing HTTP and HTTPS requests and responses modification on the fly, revealing hidden fields, allowing editing and replay of previous requests, collecting and analyzing session ID’s and performing automated substitution of parameters. In its most common usage, WebScarab operates as an intercepting proxy, allowing the operator to review and modify requests created by the browser before they are sent to the server, and to review and modify responses returned from the server before they are received by the browser. WebScarab is able to intercept both HTTP and HTTPS communication. The operator can also review the conversations (requests and responses) that have been passed through WebScarab [82].

### 4.3.3 Test Results

The test sets are generated using OWASP testing guide, mentioned in section 3.7.

### 4.3.3.1 Information Gathering

**Application Fingerprint:** Httprint tool has been used to evaluate for application signature. However it fails to reveal EYEKS application layer signature. The results are given below.

(EYEKS running on port 8070, backhand application server which is embedded OC4J running on 8988, a test apache server running on 8080)

**Table 17 Application Fingerprint Test**

| Host | Port | Banner Reported | Banner Reduced |
|---|---|---|---|
| Localhost | 8080 | Apache-Coyote/1.1 | Apache-Tomcat/4.1.29 |
| localhost | 8988 | Oracle Containers for J2EE | TUX/2.0 (Linux) |
| localhost | 8070 | Unspecified Error… | |

So EYEKS (if deployed on stand-alone server) does not reveal backhand application fingerprints.

**Application Discovery:**

Web application discovery aims to identify web applications on a given infrastructure. The offered test sets consist of trying different base URLs, ports and virtual hosts. Other than /mystore context which is explicitly mapped, no other context can be accessed. Port scan is done using NMap tool, it finds that three web servers are running on target computer however this is because all of the applications (including EYEKS) are deployed on the same machine for testing purposes. Targeting virtual hosts is related with DNS configuration, so no test can be done on this testing environment.

As a result, EYEKS hinders backhand web applications however port scanning and virtual host tracking is related with deployment of the whole application so EYEKS can not handle improper configuration and deployment.

**Spidering and Googling:** Spidering a web site means creating a map of the application with all points of access to the application. For this purpose *wget* tool has been used with the option: *wget –r –x –S http://localhost:8070/mystore* which searches though the targeted web site and downloads the structure recursively. It fetched images through */mytoystore/staticfiles /images* directory but failed on */mystore/application* and */mystore/templates* directories. The only successful page that is downloaded was */mystore/application/home.jsp* which is a welcome page of the application, for the other page links, it downloaded *error.html* page which indicates login was needed. Googling could not be tested.

**Analysis of error code:** The purposes of error code analysis is generating unexpected error problems on web applications, then analyze the response error page to reveal web application technology like the database or application server information. Various malformed HTTP requests have been generated using CAL9000, but no sensible information could be fetched. However when backhand web application inserted error codes within a successful HTTP response, EYEKS could not sense that error code has been revealed. So it can be said that EYEKS is partially successful on this test set, additional content filtering operation must be added to operation chain to hide sensible error codes generated by backhand web applications.

**SSL/TLS testing:** EYEKS does not handle improper configuration of SSL/TSL, no countermeasure to confront attacks about SSL/TLS has been implemented. So this test set has been skipped.

**DB Listener testing:** is out of scope and has been skipped.

**File extension handling:** EYEKS fails to obscure file extensions. By inspecting file extensions, it is possible to infer underlying technologies. For example */mystore/application/home.jsp* is a welcome page and reveals that J2EE technology has been used for web application.

**Old, backup and unrefered files:** It depends on the context and mappings. This test set is handled using DirBuster tool. Any files on StaticContext can be retrieved, however on ScriptContext and ApplicationContext contexts, test files could not be retrieved. So with this test configuration, EYEKS do not reveal sensitive information though unrefered files.

#### 4.3.3.2 Business Logic Testing

**Testing for business logic:** OWASP states that if a web application is an e-commerce application, most probable places of business logic errors are product ordering, checkout business scenarios so additional business logic test has been done on *yourcart.jsp* and *confirmshoppinginfo.jsp*. Various parameter manipulations have been tested using CAL9000 but no dangerous business errors can be generated like ordering a product by mimicking another registered user

#### 4.3.3.3 Authentication Testing

**Default or Guessable Account:** Guessable user accounts are application responsibility. So no testing has been done.

**Brute Force:** EYEKS successfully locked user accounts after 3 unsuccessful login tries and opens a locked account after 20 minutes so brute forcing user account would not be possible.

**Bypassing authentication schema:** Authentication schema can be bypassed by a direct page request, parameter manipulation, session ID prediction and SQL injection. Direct page request has been tested while spidering testing using *wget*. The results were satisfiable; none of the pages under

ApplicationContext which requires authentication could be fetched. For to analyze session ID prediction, WebScarab tool has been used for capturing and analyzing EYEKSTOKEN values. 250 tokens has been captured and analyzed. The following snapshot has been taken from WebScarab and shows the result. The EYEKSTOKEN seems to form a pattern so that it can be predictable; however the range of edit distance values is infinity. In fact minimum edit distance has been found to be 3.56 E+37, so it is easy to say that Eyeks session ID's can be impossible to predict. However if backhand application has implemented unsafe handling of their own session ID, the system would still vulnerably to session ID prediction. Bypassing authentication schema using SQL injection has been inspecting using WebScrab. WebScrab founds that *signing.jsp* can be vulnerably to possible injection but this is a false alarm because although it is possible to inject SQL statements in parameter values at client side, EYEKS will refuse this kind of attacks and response a proper error message. Further testing using WebScrab and CAL9000 show that authentication mechanism is safe from SQL injection attacks.

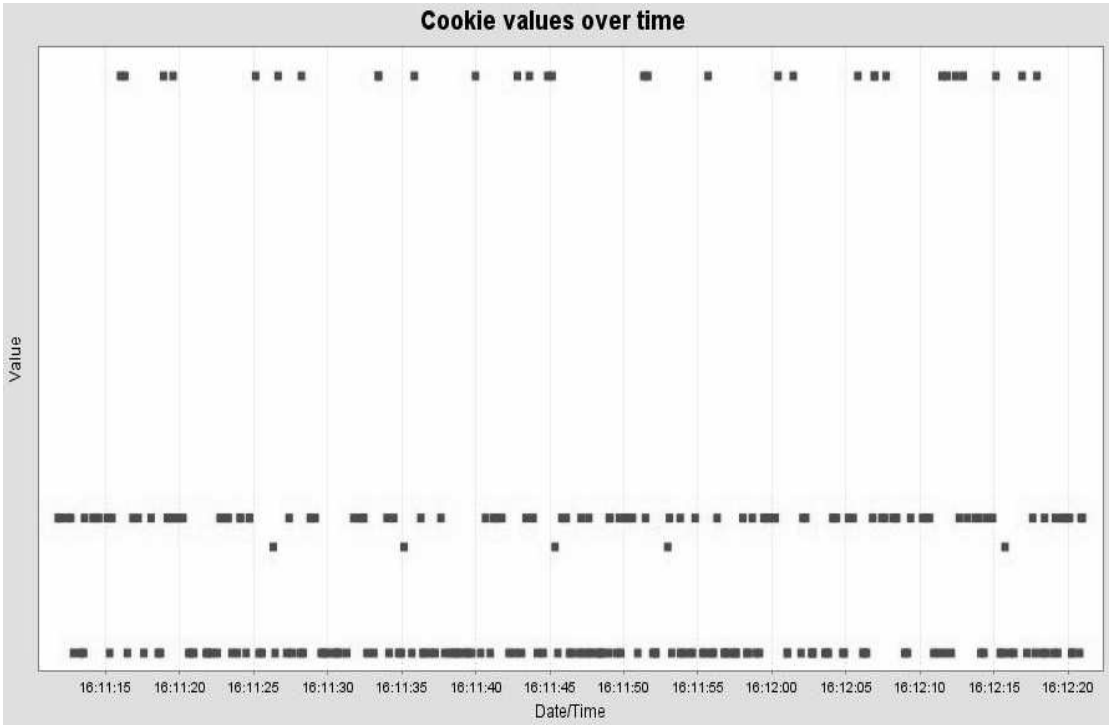


**Figure 34 Cookie Distribution over Time**

**Directory traversal/file include:** Some part of this test set has been executed during spidering test successfully. For testing these kinds of attacks in detail, a test set has been prepared including directory traversal though different encoding techniques like hexadecimal encoding (%XX), Unicode encoding (%uUUUU), named encoding (&lt) and with double encodings like URL over Hex, Unicode over Hex. The results were successful; EYEKS would not allow directory traversal even if different kinds of encodings have been used. The tested web application does not contain file operations, so file inclusion through input vector enumeration could not be tested.

**Logout and Browser Cache Management Testing:** Logout operation testing consists of testing logout function if session remains after logout and after logout can any cached pages be accessible or not. Testing logout functionality is handled by backing the previous pages and trying to continue operation, checking if session token (EYEKSTOKEN) expires and checking response pages headers whether Cache Control: no-cache header is included. Test results show that although session token remains valid, no further operations are allowed after logout and every page under ApplicationContext contains no-cache header.

#### 4.3.3.4 Session Management Testing

**Session Management Schema:** This test set contains analyzing methods to identify session management technique of the web application. Response headers, cookies and content have been checked to identify the mechanism. WebScarab has been used for this purpose and it has been founded that the application uses cookie based session management named EYEKSTOKEN.

**Session Token Manipulation:** In this test set, session ID's of application were tested against predictability and randomness. As described previously, the session ID's has been analyzed using WebScarab and found to be secure from any types of brute force and reverse engineering attacks. Minimum edit distance is found to be 3.56 E+37 and session ID contains 457 characters with in character set *[a-z] U [A-Z] U {%}.*

**Exposed Session Variables:** Session variables can easily be exposed, since it does not transfer session variables or cookies using SSL. However session tokens can not be reused every new request invalidate previous token.

**Session Riding:** Session riding is very hard to test since it needs a number of different attack vectors to be executed, however form based authentication with carrying session token within the content removes risk of session riding. Current configuration of test web application uses cookie based session management however can be configured to use content based session management to remove the risk.

**HTTP Exploit and Injection Attacks:** (Cross site scripting, XST, SQL, stored procedure, ORM, LDAP, XML, SSI, XPath, IMAP/SMTP, Code, Command injection) HTTP exploit and injection attacks have been tested automatically using WebScarab and manually using CAL9000. WebScarab

reports that 12 operations with 4 different pages can possibly be vulnerably to injection attacks as can be seen in following snapshot.

| ID | Date | Method | Host | Path | P... | Status | Origin | Possible Injection |
|---|---|---|---|---|---|---|---|---|
| 3 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/faces/showcategory.jsp | | 200 OK | Proxy | ☐ |
| 4 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/styles/cache/oracle-desktop-10_1_3... | | 200 OK | Proxy | ☐ |
| 5 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/jsLibs/Common10_1_3_3_0.js | | 200 OK | Proxy | ☐ |
| 6 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/images/branding.gif | | 200 OK | Proxy | ☐ |
| 7 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/t.gif | | 200 OK | Proxy | ☐ |
| 8 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/cache/cghes.gif | | 200 OK | Proxy | ☐ |
| 9 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/cache/cghee.gif | | 200 OK | Proxy | ☐ |
| 10 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/cache/cghec.gif | | 200 OK | Proxy | ☐ |
| 11 | 2007/0... | GET | http://localhost:8988 | /favicon.ico | | 404 Not Fou... | Proxy | ☐ |
| 12 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/cache/cseparator.gif | | 200 OK | Proxy | ☐ |
| 13 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/showcategory.jsp | | 200 OK | Proxy | ☑ |
| 14 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/oracle/tnavpd.gif | | 200 OK | Proxy | ☐ |
| 15 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/oracle/tnavn.gif | | 200 OK | Proxy | ☐ |
| 16 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/showcategory.jsp | | 200 OK | Proxy | ☑ |
| 17 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/showproduct.jsp | | 200 OK | Proxy | ☑ |
| 19 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/showcategory.jsp | | 200 OK | Proxy | ☑ |
| 20 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/showproduct.jsp | | 200 OK | Proxy | ☑ |
| 21 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/cache/en/bUpdaBloK.gif | | 200 OK | Proxy | ☐ |
| 22 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/cache/en/bChecXP9g.gif | | 200 OK | Proxy | ☐ |
| 23 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/yourcart.jsp | | 200 OK | Proxy | ☑ |
| 24 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/yourcart.jsp | | 200 OK | Proxy | ☑ |
| 25 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/cache/en/bConfBJ8V.gif | | 200 OK | Proxy | ☐ |
| 26 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/reviewcheckout.jsp | | 302 Moved ... | Proxy | ☐ |
| 27 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | ?... | 200 OK | Proxy | ☑ |
| 28 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/errorl.gif | | 200 OK | Proxy | ☐ |
| 29 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/cache/en/bLogio9S-.gif | | 200 OK | Proxy | ☐ |
| 30 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/adf/images/cache/cmbts.gif | | 200 OK | Proxy | ☐ |
| 31 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | | 302 Moved ... | Proxy | ☐ |
| 32 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | ?... | 200 OK | Proxy | ☑ |
| 33 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | | 200 OK | Proxy | ☑ |
| 34 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/showcategory.jsp | | 302 Moved ... | Proxy | ☐ |
| 35 | 2007/0... | GET | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | ?... | 200 OK | Proxy | ☑ |
| 36 | 2007/0... | POST | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | | 200 OK | Proxy | ☑ |

**Figure 35 WebScarab Testing Report**

The test results of web application without EYEKS are shown in *figure 36*. Without installing EYEKS, test web application is found to be vulnerable to XSS attacks and possibly more kinds of injection attacks.

| ID | Date | Method | Host | Path | Status | Origin | XSS | CRLF |
|---|---|---|---|---|---|---|---|---|
| 13 | 2007/08/04... | POST | http://localhost:8988 | /ADFToyStore/faces/showcategory.jsp | 200 OK | Proxy | ☑ | ☐ |
| 16 | 2007/08/04... | POST | http://localhost:8988 | /ADFToyStore/faces/showcategory.jsp | 200 OK | Proxy | ☑ | ☐ |
| 17 | 2007/08/04... | POST | http://localhost:8988 | /ADFToyStore/faces/showproduct.jsp | 200 OK | Proxy | ☑ | ☐ |
| 19 | 2007/08/04... | POST | http://localhost:8988 | /ADFToyStore/faces/showcategory.jsp | 200 OK | Proxy | ☑ | ☐ |
| 20 | 2007/08/04... | POST | http://localhost:8988 | /ADFToyStore/faces/showproduct.jsp | 200 OK | Proxy | ☑ | ☐ |
| 23 | 2007/08/04... | POST | http://localhost:8988 | /ADFToyStore/faces/yourcart.jsp | 200 OK | Proxy | ☑ | ☐ |
| 24 | 2007/08/04... | POST | http://localhost:8988 | /ADFToyStore/faces/yourcart.jsp | 200 OK | Proxy | ☑ | ☐ |
| 27 | 2007/08/04... | GET | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | 200 OK | Proxy | ☑ | ☐ |
| 32 | 2007/08/04... | GET | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | 200 OK | Proxy | ☑ | ☑ |
| 33 | 2007/08/04... | POST | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | 200 OK | Proxy | ☑ | ☐ |
| 35 | 2007/08/04... | GET | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | 200 OK | Proxy | ☑ | ☐ |
| 36 | 2007/08/04... | POST | http://localhost:8988 | /ADFToyStore/faces/signin.jsp | 200 OK | Proxy | ☑ | ☑ |

**Figure 36 Reported Vulnerabilities without EYEKS**

However, after installing EYEKS as an application security layer, No XSS attacks have been reported. On the other hand, EYEKS fails against Carriage Return/Line Feed type of attack that can be result in HTTP Exploits such as HTTP Response Splitting.

| ID | Date | Method | Host | Path | Status | Origin | XSS | CRLF |
|---|---|---|---|---|---|---|---|---|
| 13 | 2007/08/04 1... | POST | http://localhost:8070 | /mystore/faces/showcategory.jsp | 200 OK | Proxy | ☐ | ☐ |
| 16 | 2007/08/04 1... | POST | http://localhost:8070 | /mystore/faces/showcategory.jsp | 200 OK | Proxy | ☐ | ☐ |
| 17 | 2007/08/04 1... | POST | http://localhost:8070 | /mystore/faces/showproduct.jsp | 200 OK | Proxy | ☐ | ☐ |
| 19 | 2007/08/04 1... | POST | http://localhost:8070 | /mystore/faces/showcategory.jsp | 200 OK | Proxy | ☐ | ☐ |
| 20 | 2007/08/04 1... | POST | http://localhost:8070 | /mystore/faces/showproduct.jsp | 200 OK | Proxy | ☐ | ☐ |
| 23 | 2007/08/04 1... | POST | http://localhost:8070 | /mystore/faces/yourcart.jsp | 200 OK | Proxy | ☐ | ☐ |
| 24 | 2007/08/04 1... | POST | http://localhost:8070 | /mystore/faces/yourcart.jsp | 200 OK | Proxy | ☐ | ☐ |
| 27 | 2007/08/04 1... | GET | http://localhost:8070 | /mystore/faces/signin.jsp | 200 OK | Proxy | ☐ | ☐ |
| 32 | 2007/08/04 1... | GET | http://localhost:8070 | /mystore/faces/signin.jsp | 200 OK | Proxy | ☐ | ☑ |
| 33 | 2007/08/04 1... | POST | http://localhost:8070 | /mystore/faces/signin.jsp | 200 OK | Proxy | ☐ | ☐ |
| 35 | 2007/08/04 1... | GET | http://localhost:8070 | /mystore/faces/signin.jsp | 200 OK | Proxy | ☐ | ☐ |
| 36 | 2007/08/04 1... | POST | http://localhost:8070 | /mystore/faces/signin.jsp | 200 OK | Proxy | ☐ | ☑ |

**Figure 37 Reported Vulnerabilities with EYEKS**

In fact, further investigations about XSS attacks show that EYEKS is still vulnerable to XSS attacks if an attack also includes character encoding attacks. Some examples of the injection are;

**Unicoded XSS attack:** <DIV STYLE="background-image:\0075\0072\006C\0028'\006a\0061\0076\0061\0073\0063\0072\0069\0070\0074\003a\0061\006c\0065\0072\0074\0028.1027\0058.1053\0053\0027\0029'\0029">
**Hex Encoding:** <IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>
**Broken up javascript:** @im\port'\ja\vasc\ript:alert("XSS")';

Test results show that EYEKS is still safe from SQL, LDAP and code injection. EYEKS rejects these kinds of attacks with proper error messages.

Denial of service testing, web services testing and AJAX testing was not executed since denial of service testing depends on backhand web application and EYEKS does not support web service, AJAX security. A summary of web application security test results is given in *table 18*.

**Table 18 OWASP Testing Results**

| **Category** | Test Name | Result |
|---|---|---|
| **Information Gathering** | Application Fingerprint | Successful |
| | Application Discovery | Successful |
| | Spidering and googling | Successful |
| | Analysis of error code | Partially successful, could no filter backhand application errors. |
| | SSL/TLS Testing | Not responsible |
| | DB Listener Testing | Not responsible |
| | File extensions handling | Fails |
| | Old, backup and unrefered files | Successful |
| **Business logic testing** | Testing for business logic | No errors were found, but improper configuration could result vulnerabilities. |
| **Authentication Testing** | Default or guessable account | Not responsible |
| | Brute Force | Successful |
| | Bypassing authentication schema | Successful |
| | Directory traversal/file include | Successful |
| | Vulnerable remember password and pwd reset | Not responsible |
| | Logout and Browser Cache Management Testing | Successful |
| **Session Management** | Session Management Schema | Successful |
| | Session Token Manipulation | Successful |
| | Exposed Session Variables | Fails however does not cause security risk. |
| | Session Riding | Successful if content managed session tokens have been used. |
| | HTTP Exploit | Fails because of CRLF vulnerability |
| | Cross site scripting | Fails if different kinds of encoding have been used. |
| | HTTP Methods and XST | Successful |
| | SQL Injection | Successful |
| | Stored procedure injection | Not tested |
| | ORM Injection | Successful |
| | LDAP Injection | Successful |
| | XML Injection | Not tested |
| | SSI Injection | Successful |
| | XPath Injection | Not tested |

**Table 18 (continued)**

| | | |
|---|---|---|
| | IMAP/SMTP Injection | Not tested |
| | Code Injection | Successful |
| | OS Commanding | Successful |
| | Buffer overflow | Not tested |
| | Incubated vulnerability | |
| | Writing User Provided Data to Disk | Successful |
| **Denial of Service Testing** | Failure to Release Resources | Not responsible |
| | Storing too Much Data in Session | Not responsible |
| | XML Structural Testing | Not responsible |
| | XML content-level Testing | Not responsible |
| | HTTP GET parameters/REST Testing | Not responsible |
| | XML Structural Testing | Not responsible |
| **Web Services Testing** | XML content-level Testing | Not supported |
| | HTTP GET parameters/REST Testing | Not supported |
| | Naughty SOAP attachments | Not supported |
| | Replay Testing | Not supported |
| **AJAX Testing** | Testing AJAX | Not supported |

0

# CHAPTER 5

# CONCLUSION AND FUTURE WORK

In this thesis, a fully implemented solution EYEKS that secures web application is presented. Adding a new layer to web application that deals with all security aspects makes application developers free from thinking about security issues of the application, and also leads to more functional, structured and scalable system.

The most important access control problem of enterprise applications is encapsulating domain specific factors in access decisions. Middleware infrastructures are incapably of providing enough abstraction to evaluate enterprise-level security policies. Enterprise application developers tackle this problem by embedding access control rules within an application code that handles domain-specific factors. However enterprise access control rules aims to implement enterprise security policies that are mostly stated by legislations, regulations or company's business processes so that they are subject to frequently changes and modifications. It is very hard to tackle these frequent modifications with embedding access control rules into application code since every change requires a new software cycle of deployment and testing processes. As a result, it reduces reusability and manageability of whole system. EYEKS handles this problem by introducing transparent access control evaluation using RAD service. RAD specification has been shown to be one of the best authorization mechanisms to encapsulate domain specific factors in access control. Enterprise web applications can be mapped to RAD domain by defining resources and operations of the system and any access to those secured resources can be controlled by defining enterprise-level security policies to RAD that are deduced from complex access control rules of the application. This enables access control logic of the application to be managed outside of the application and directly by RAD implementation. Security officer, who is ideally non-developer person, can manage access control policies and verify that they satisfy security requirements of the system without bothering with application code. EYEKS reflects access control changes simultaneously without requiring redeployment of the applications that improves manageability and reusability of the system significantly.

EYEKS not only verifies and enforces “enterprise-level security (access) policies” but also can provide a common evaluation and enforcement environment for application level policies. The mapping from web application structure to RAD domain also provides a well structured and efficient positive security model by naming all allowed resources and operations of the system. This is the main contribution and aspect of this thesis. Although using positive security model is an essential to secure web application and prevents most of the dangerous types of attacks, a negative security model that targets specific attack types may also be needed. EYEKS also allows attack signatures to be defined as security policies that can be added to the system to build a negative security model.

EYEKS has been designed as reverse proxy that works inline mode, installed in front of the web applications and introduces a specific layer, so called application security layer to control the traffic and enforce security policies to be satisfied. The layered operations structure enables full control over request-response chain of HTTP protocol where any kind of verifications and manipulations can be done by adding suitable operations to operation chain. All aspects of web application security like session management, authentication, authorization and content checking have been implemented as atomic operations and can be added according to web applications security needs. Although EYEKS is a self-sufficient tool that can be used with any kind of web applications, still can be extended by implementing new operations. EYEKS operations has simple interface that must be implemented to adapt currently unsupported operations.

EYEKS has been used for two e-government projects in Turkey and Azerbaijan since October 2004. The experience with these real life systems has shown that EYEKS offers great benefits to enterprise web applications. First of all, the powerful authorization and authentication mechanism has been freed developers from considering access control issues. Separating enterprise-level policies from application code gives flexibility to developers since access and security requirement changes no longer effects whole application code but only a revision is needed for access policies. On the other hand this approach leads us to continued security where access control and security requirements are handled independently during analysis, design and implementation phases and updated within each phases, besides it also simplify overall analysis, design and implementation efforts. The performance results are also very promising. Although the overall traffic is concentrated in the third week of the months where it takes nearly 70-80% of monthly traffic, the average peek CPU usage has not been over 27 % where backend servers (8 servers) usage is 92 % on even peek days. The artificial stress tests are also strengthen these results; although after number of 250 concurrent users, the backhand applications' response time becomes increasing exponentially and the payload of EYEKS is still stable and have an average of 8%. However the performance related test sets are generated by only considering the number of user as parameter, the tests can be extended to cover the results of increasing the number of policies. The caching mechanism of CSAAS tries to target performance drawback for increasing number of users and policies. Although the expected result is that EYEKS will response steadily, more test can be done to inspect system performance for increasing number of policies as a future work.

Another test set targets EYEKS defense to web application attacks, OWASP web application testing guide has been used as a guideline and various security tests has been performed. For this purpose, a public available, open source web application has been chosen and EYEKS was installed in front of it as an application security layer. Positive security model has been constructed by mapping the application to RAD domain and a basic negative security model has been implemented by policies. The testing results shows that EYEKS very well confronts information gathering and session stealing types of attacks. However there are also security breaches with some type of injection attacks that

shows that injection detection policy is not strength enough to detect all kind of injections. This is not a surprising result since injection detection policy has been using a set of black list (holding injection attack signatures) and attacker can bypass black list check by hiding attack with different type's encodings.

EYEKS introduces some improvements as well as some limitations over related works. As access control mechanism, it can compete with other policy based access control mechanisms. Its expressional power to encapsulate domain specific factors is not behind its competitors. As web application firewall, EYEKS can not be regarded as a full product. It can gain web application firewall characteristics by adding strong attack detection policies. Although can be extended, current detection policies of EYEKS are surpassed by broad range of attack signatures supplied by on-market web application firewalls. The other limitation is EYEKS can only run as reverse proxy mode, but other products can also run on bridge and router modes. Although reverse proxy has some advantages like information hiding and give full power to manipulate the traffic, it lacks of performance while handling dual sessions of both party. One of the important advantages of EYEKS over web application firewall products is secure handling of user authentication and session management. Web application firewalls do not interfere authentication and session management mechanisms of backhand applications. If they have security flaws with these mechanisms, they would be still vulnerably to these kinds of attacks even if they are behind web application firewalls. On the other hand, EYEKS can mandate web application's authentication and session management mechanisms by using strong authentication and session handling methods

The real strength of EYEKS comes out when we consider both aspects together. A centralized view of security aspects enables web application to be more manageable. It also improves the traceability of the system, the security and access control requirements can be directly mapped to enterprise-level and application level security policies and traced through EYEKS. Business depended enterprise-level security policies and protection mechanisms (application-level security policies) can be added together to form a full security policy chain that can be managed on RAD specification. RAD implementations offer high available, fine-grain, extensible and dynamic access control mechanism which suits well for web application authorization needs. As a result, adding an application security layer that controls organization-wide security policies, could give great benefits such as reusability, manageability, scalability to all kind of web application

## 5.1 Future Work

EYEKS must be regarded as a security framework rather than a full product. The core of EYEKS only provides a common evaluation and enforcement environment for both enterprise-level and application level policies, so its strength depends directly on policies that are defined. From this point of view, for the most efficient usage, the design and implementation of backhand enterprise applications must be

considered according to EYEKS so that EYEKS will lead the process of continued security. Although EYEKS succeeds to fulfill its claims, there are still some points that can be improved.

First of all, defining resource and operations requires manual process and can be quite cumbersome. Most of the web application firewalls have automated process to construct positive security model. EYEKS can be extended to reveal resources, operations and also security attributes (parameters) automatically. This can be achieved by integrated web crawler or by learning process that tracks normal execution of enterprise web applications.

Client-side security can also be improved using EYEKS; form and parameter sealing, validation of Javascripts and support for XML based technologies like AJAX can be added to the system as a future work. Although EYEKS can be deployed on multiple instances, the load-balancing and fault tolerance features does not supported and must be accomplished though load-balancing switches so one of the feature work can be adding load-balancing and fault-tolerance features to EYEKS.

The other improvement can be done on RAD specification rather than EYEKS. RAD limits resources to a flat structure; however a hierarchical view of resource will improve the manageability of policies significantly. This will also improve EYEKS manageability.

EYEKS has lack of SSL support with is in fact essential for any web security products. Supporting SSL and digital signatures for authentication and authorization is another planned future work

# REFERENCES


**[1]** NSF, “Information Technology Research Program Requirements,” National Science Foundation, 1999.

**[2]** Blakley B., “CORBA Security: an Introduction to Safe Computing with Objects”, First ed. Reading: Addison-Wesley, 1999.

**[3]** Microsoft, “DCOM Architecture”, Microsoft, 1998.

**[4]** Sun, Enterprise, “JavaBeans Specification Documentation 3.0 Final Release”, 2006.

**[5]** Beznosov K., “Object Security Attributes: Enabling Application-Specific Access Control in Middleware”. In DOA'02, pages 693-710, London, UK, October 2002.

**[6]** Göğebakan Y., “Cok Katmanlı Internet Uygulamalarında Yetkilendirme Problemi”, Akademik Bilişim Konferansı 2005.

**[7]** Metin M.Ö., Şener C., Göğebakan Y., “Creating Application Security Layer Based on Resource Access Decision Service”, International Conference on Security of Information and Networks, SIN 2007

**[8]** SANS Institute, “SANS Top-20 Internet Security Attack Targets”, http://www.sans.org/top20/, (last accessed August 28 2007)

**[9]** Open Web Application Security Project (OWASP), “The OWASP Testing Guide version 2”, http://www.owasp.org/index.php/OWASP_Testing_Project, (last accessed August 28 2007)

**[10]** Shin S., “Web Application Security Threats and Counter Measures”, http://www.javapassion.com /j2ee/WebSecurityThreats.pdf, (last accessed August 28 2007)

**[11]** Common Vulnerabilities and Explosures (CVE), “All Exposures List”, http://cve.mitre.org/cve/, (last accessed August 28 2007)

**[12]** Privacy Rights ClearingHouse, “A Chronology of Data Breaches”, http://www.privacyrights.org /ar/ChronDataBreaches.htm, (last accessed August 28 2007)

**[13]** Icove D., Seger K. And VonStorch W., “Computer Crime: A Crimefighter's Handbook”, O'Reilly & Associates, Inc., Sebastopol, CA, 1995.

**[14]** Cheswick W.R., Bellovin S.M., “Firewalls and Internet Security: Repelling the Wily Hacker”, Addison-Wesley Publishing Company, Reading, MA, 1994.

**[15]** Lough D. L., “A Taxonomy of Computer Attacks with Applications to Wireless Networks”,

**[16]** Cohen F.B., “Protection and Security on the Information Superhighway”, John Wiley & Sons, New York, 1995.

**[17]** Stallings W., “Network and Internetwork Security Principles and Practice”, Prentice Hall, Englewood Cliffs, NJ, 1995.

**[18]** Web Application Security Consortium (WASC), "Threat Classification", http://www.webappsec.org/projects/threat/ , (last accessed August 28 2007).

**[19]** Open Web Application Security Project (OWASP), "The Ten Most Critical Web Application Security Vulnerabilities", http://www.owasp.org/index.php/Top_10_2004 , (last accessed August 28 2007)

**[20]** Open Web Application Security Project (OWASP), "The Ten Most Critical Web Application Security Vulnerabilities", http://www.owasp.org/index.php/Top_10_2007 , (last accessed August 28 2007)

**[21]** Krügel C., Vigna G., "Anomaly Detection of Web-. Based Attacks", in Proc. 10th ACM Conference on Computer and Communications Security.

**[22]** Zhang L., White G.B., "Analysis of Payload Based Application Level Network" Anomaly Detection, 40th Annual Hawaii International Conference on System Sciences (HICSS'07).

**[23]** Web Application Security Consortium (WASC), "Web Security Glossary", http://www.webappsec.org/projects/glossary/ , (last accessed August 28 2007).

**[24]** Payment Card Industry (PCI), "Data Security Standard (DSS) version 1.1", https://www.pcisecuritystandards.org/tech/index.htm, (last accessed August 28 2007).

**[25]** OMG Security Specifications, "Resource Access Decision (RAD) Version 1.0", http://www.omg.org/technology/documents/formal/resource_access_decision.htm , (last accessed August 28 2007).

**[26]** Ferraiolo D.F, Sandhu R., Gavrila S., Kuhn D.R., Chanramouli R., "Proposed NIST Standard for Role-Based Access Control", National Institute of Standards and Technology (NIST).

**[27]** Lucas J., Moeller B., "The Effective Incident Response Team", Addison-Wesley Professional; 1st edition (September 26, 2003).

**[28]** RFC 3067, "Incident Object Description and Exchange Format Requirements", The Trans-European Research and Education Networking Association (TERENA).

**[29]** Howard J. D., Longstaff T.A., "A Common Language for Computer Security Incidents", Sandia National Laboratories (1998).

**[30]** Grance T., Kent K., Kim B., "Computer Security Incident Handling Guide", NIST Publication SP800-61.

**[31]** Van Wyk K.R., Forno R.. "Incident Response", O'Reilly Press, ISBN # 0-59600-130-4.

**[32]** IEEE, "The IEEE Standard Dictionary of Electrical and Electronics Terms", Sixth Edition, Institute of Electrical and Electronics Engineers, Inc., New York, NY, 1996.

**[33]** Amoroso E.G., "Fundamentals of Computer Security Technology", Prentice-Hall PTR, Upper Saddle River, NJ, 1994.

**[34]** Krsul I., "Software Vulnerability Analysis", PhD Thesis, Purdue University.

**[35]** Lindqvist U., Jonsson E., "How to Systematically Classify Computer Security Intrusions," Proceedings of the 1997 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos, CA, May, 1997, pp. 154-163.

**[36]** Cohen F.B., "Protection and Security on the Information Superhighway", John Wiley & Sons, New York, 1995.

**[37]** Cohen F.B, "Information System Attacks: A Preliminary Classification Scheme," Computers and Security, Vol. 16, No. 1, 1997, pp. 29-46.

**[38]** Aslam T., "A Taxonomy of Security Faults in the UNIX Operating System," Master of Science Thesis, Purdue University (1995).

**[39]** Bishop M., Bailey D., "A Critical Analysis of Vulnerability Taxonomies". Tech. Rep. Department of Computer Science at the University of California, September 1996.

**[40]** Russell D., Gangemi G.T., "Computer Security Basics", O'Reilly & Associates; 1 edition (January 1991).

**[41]** Neumann P., Parker D., "A Summary of Computer Misuse Techniques" Proceedings of the 12th National Computer Security Conference, 1989.
**[42]** Power R., "Current And Future Danger: A CSI Primer of Computer Crime & Information Warfare", CSI Bulletin.

**[43]** Beznosov K., "Engineering Access Control for Distributed Enterprise Applications", PhD Thesis, Florida International University, July, 2000.

**[44]** United States Department of Defense, Trusted Computer System Evaluation Criteria, DoD Standard 5200.28-STD (1985).

**[45]** United States Department of Defense, Understanding Discretionary Access Control in Trusted Systems.

**[46]** Boebert, W.E., and Ferguson, C.T., "A Partial Solution to the Discretionary Trojan Horse Problem", 9th Security Conference, DoD/NBS, September 1985, pp 141-144.

**[47]** Ferraiolo D., Kuhn D. R., "Role-based access control", in 15th National Computer Security Conference. NIST/NSA, 1992.

**[48]** Sandhu R., Coyne E. J. Feinstein H. L., and Youman C. E. "Role-based access control models". IEEE Computer, 29(2), February 1996.

**[49]** Ferraiolo D., Cugini J., and Kuhn D. R... "Role-based access control: Features and motivations", in Annual Computer Security Applications Conference. IEEE Computer Society Press, 1995.

**[50]** Ossher H., Tarr P., "Using multidimensional separation of concerns to (re)shape evolving software", ACM, 44(10):43-50, 2001.

**[51]** Verhanneman T., Piessens F., De Win B., Truyen E., Joosen W., "A Modular Access Control Service for Supporting Application-Specific Policies", IEEE Computer Society, June 2006 (vol. 7, no. 6), art. no. 0606-o6001 1541-4922.

**[52]** Open Web Application Security Project (OWASP), Home site, http://www.owasp.org/, (last accessed August 28 2007)

**[53]** MITRE Corparation, Home site, http://www.mitre.org, (last accessed August 28 2007)

**[54]** Open Systems Interconnection (OSI), "Information Technology -- Open Systems Interconnection -- Security frameworks in open systems -- Part 3: Access control", ISO/IEC JTC1 10181-3, 1994.

**[55]** Microsoft, "Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication", Microsoft Press, 2002. (at MS web site).

**[56]** Lai C., Gong L., Koved L., Nadalin A., and Schemers R., "User Authentication And Authorization In The Java Platform", in Proceedings of Annual Computer Security Applications Conference, Phoenix, Arizona, USA, 1999, pp. 285-290.

**[57]** Beznosov K., "Access Control Mechanisms in Commercial Middleware", Tutorial JavaPolis, Antwerpen, Belgium, 16 December, 2004.

**[58]** Beznosov K., "Middleware and Web Services Security Mechanisms", in lecture, Katholieke Universiteit Leuven, Brussels, Belgium 2 March, 2005, pp.65.

**[59]** Jajodia S., Samarati P., Sapino M. L., and Subrahmanian V. S., "Flexible support for multiple access control policies", ACM Trans. Database Syst., 26(2):214–260, 2001.

**[60]** Ryutov T., Neuman C., "Access Control Framework for Distributed Applications", IETF, Internet Draft draft-ietf-cat-acc-cntrl-frmw-03, March 9 2000.

**[61]** Ryutov T., Neuman C., "Generic Authorization and Access control Application Program Interface: C-bindings", IETF, draft-ietf-cat-gaa-bind-03, March 9 2000.

**[62]** OASIS Core Specification, "eXtensible Access Control Markup Language (XACML) Version 2.0".

**[63]** Damianou N., Dulay N., Lupu E., and Sloman M.. "The Ponder Policy Specification Language", LNCS, 1995:18-28,2001.

**[64]** Verhanneman T., Piessens F., De Win B., Truyen E., Joosen W., "Implementing a Modular Access Control Service to Support Applications Specific Policies in CaesarJ", 6th International Middleware Conference November 28th 2005, Grenoble, France.

**[65]** BEA, "BEA Web Logic Enterprise Security version 4.2", http://edocs.bea.com/wles/docs42 /index.html (last accessed August 28 2007).

**[66]** Oracle Access Manager, http://www.oracle.com/technology/products/id_mgmt/coreid_acc /index.html, (last accessed August 28 2007).

**[67]** IBM WebSphere, http://www-306.ibm.com/software/websphere/, (last accessed August 28 2007).

**[68]** Entegrity AssureAccess Product, Home Site, http://www.entegrity.com/products/aa/aa.shtml, (last accessed August 28 2007).

**[69]** Zhao C., Chen Y., Xu D., Heilili N., Lin Z., "Integrative Security Management for Web-Based Enterprise Applications", Department of Information Science, Peking University.

**[70]** Scott D., Sharp R., "Abstracting Application-Level Web Security", The 11th International World Wide Web Conference (WWW2002), May 2002.

**[71]** ModSecurity, "Open Source Web Application Firewall", http://www.modsecurity.org/, (last accessed August 28 2007).

**[72]** TrafficShield® Application Firewall, f5 Home Site, http://www.f5.com/products/TrafficShield/, (last accessed August 28 2007).

**[73]** Imperva SecureSphere®, Imperva Home Site, http://www.imperva.com/products/securesphere/, (last accessed August 28 2007).

**[74]** Beznosov K., Deng Y., Blakley B., Burt C. and Barkley J., “A Resource Access Decision Service for CORBA-based Distributed Systems.”, in proceedings of the Annual Computer Security Applications Conference, Phoenix, Arizona, U.S.A, December 6-10, 1999.

**[75]** Muench S., ADF Toystore Demo Application, Oracle, http://www.oracle.com/technology/products/jdev/collateral/papers/10g/adftoystore.html, (last accessed August 28 2007).

**[76]** Open Web Application Security Project, “OWASP CAL9000 Project”, http://www.owasp.org/index.php/Category:OWASP_CAL9000_Project, (last accessed August 28 2007).

**[77]** RSnake's Security Stuffs/Hacks, http://ha.ckers.org/, (last accessed August 28 2007).

**[78]** “Httprint web server fingerprinting tool version is build 301 (beta)”, http://net-square.com/httprint/, (last accessed August 28 2007).

**[79]** GNU Wget, http://www.gnu.org/software/wget/, (last accessed August 28 2007).

**[80]** Open Web Application Security Project, “OWASP DirBuster Project”, http://www.owasp.org/index.php/Category:OWASP_DirBuster_Project, (last accessed August 28 2007).

**[81]** Nmap, “Free Security Scanner for Network Exploration & Security Audits”, http://insecure.org/nmap/, (last accessed August 28 2007).

**[82]** Open Web Application Security Project, OWASP WebScarab Project, http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project, (last accessed August 28 2007).

**[83]** Perry T., Wallich P., "Can Computer Crime Be Stopped?," IEEE Spectrum, Vol. 21, No. 5.

**[84]** Landwehr C.E., Bull A.R., McDermott J.P, and Choi W.S, "A Taxonomy of Computer Security Flaws," ACM Computing Surveys, Vol. 26, No. 3, September, 1994, pp. 211-254.

**[85]** Preliminary List Of Vulnerability Examples for Researchers (PLOVER), http://cve.mitre.org/docs/plover/plover.html, (last accessed August 28 2007).

# APPENDIX A

# LIST OF WEB APPLICATION SECURITY VULNERABILITIES:

## 1.1 Path Traversal Attacks:

This attack technique involves providing relative or absolute path information as a part of request information. Such attacks try to access files that are normally not accessible by anyone and if this kind of request has come, it must be denied. This attack threatens information disclosure of systems. Although it does not directly threatens integrity of the system, the attacker can reveal sensitive data such as password and configuration files and by using it, he can do more dangerous attacks to the system. Path traversal attacks divided into two categories;

### 1.1.1 Relative Path Traversal:

This is a subcategory of path traversal attacks; the attacker constructs a path that contains relative traversal sequences such as "...". Examples are stated below.

- In the form of '**../filedir**' path traversal.
- In the form of '**/../filedir**' path traversal.
- In the form of '**/directory/../filename**' path traversal.
- In the form of '**directory/../../filename**' path traversal. CAN-2002-0298
- In the form of '**..\filename**' path traversal CAN-2002-0661, CVE-2002-0946, CAN-2002-1042, CAN-2002-1209, CVE-2002-1178
- In the form of '\..\filename' path traversal. CAN-2002-1987, CAN-2005-2142
- In the form of '\directory\..\filename' path traversal. CVE-2002-1987
- In the form of 'directory\..\..\filename' path traversal. CVE-2002-0160
- In the form of '...' path traversal.

CVE-2001-0615 - "..." or "...." in chat server

CVE-2001-0963 - "..." in cd command in FTP server

CVE-2001-1193 - "..." in cd command in FTP server

CAN-2001-1131 - "..." in cd command in FTP server

CAN-2001-0480 - "..." in GET or CD command in FTP server

CAN-2002-0288 - "..." in web server

CAN-2002-0784 - HTTP server protects against ".." but allows "..."

CAN-2003-0313 - Directory listing of web server using "..."

CAN-2005-1658 - Triple dot

- In the form of '....'(Multiple dots) path traversal.

CVE-2000-0240 - read files via "/........../" in URL

CVE-2000-0773 - read files via "...." in web server

CAN-1999-1082 - read files via "......" in web server (doubled triple dot?)

CAN-2004-2121 - read files via "......" in web server (doubled triple dot?)

CAN-2001-0491 - multiple attacks using "..", "...", and "...." in different commands

CVE-2001-0615 - "..." or "...." in chat server

- In the form of '....//' path traversal.

- In the form of '.../...//' path traversal. CAN-2005-2169, CAN-2005-0202

## 1.2 Absolute Path Traversal

This is a subcategory of path traversal attacks; the attacker constructs an absolute path as input and tries to access arbitrary file. Examples are;

- In the form of '**/absolute/pathname/target'** path traversal.

CAN-2002-1345 - Multiple FTP clients write arbitrary files via absolute paths in server responses

CAN-2001-1269 - ZIP file extractor allows full path

CAN-2002-1818 - Path traversal using absolute pathname

CAN-2002-1913 - Path traversal using absolute pathname

CAN-2005-2147 - Path traversal using absolute pathname

- In the form of ‘\absolute\pathname\here’ path traversal. CVE-1999-1263, CAN-2003-0753, CAN-2002-1344, CAN-2002-1525, CAN-2000-0614

## 1.3 Path Equivalence Attacks

This attack technique involves adding special characters in file and directory names. These manipulations are indented to generate multiple names and so multiple access points for the same object. Just like path traversal attacks, path equivalence attacks also threaten disclosure of information. If any application restricts directory access programmatically, these restrictions can be bypassed by adding special characters in requested file or directory so application might fail to parse requested URL and misinterpret the request. Path equivalence attacks can also used for bypassing security restrictions depends on black list. Consider an example of an application that allows uploading and a black list to eliminate malicious file formats such as symbolic links. An attacker can bypass this black list check by adding trailing dots to extension of a file. So he can traverse to target file or directory. When an attacker collects enough information about the application using path traversal and path equivalence attacks then he could plan new attacks to break into the application. So eliminating these kinds of attacks are extremely important for security. Examples are;

- In the form of 'filedir.' path equivalence.

CAN-2002-1114 - Source code disclosure using trailing dot.

CAN-2002-1986 - Source code disclosure using trailing dot.

CAN-2004-2213 - Source code disclosure using trailing dot.

CVE-2005-3293 - Source code disclosure using trailing dot.

CAN-2004-0061 - Bypass directory access restrictions using trailing dot in URL.

CAN-2000-1133 - Bypass directory access restrictions using trailing dot in URL.

CVE-2001-1386 – Bypass check for “.Ink” extension using “.Ink.”

- In the form of ‘filedir…’ (Multiple dots) path equivalence.

BUGTRAQ: 20040205 – Apache, Resin Reveals JSP Source Code.

CAN-2004-0281 - Multiple trailing dots allows directory listing.

- In the form of ‘file.ordir’ (Internal dot) path equivalence.

- In the form of ‘file…ordir’ (Multiple internal dot) path equivalence.

- In the form of 'filedir ' (Trailing space) path equivalence.

CAN-2001-0693 - Source disclosure via trailing encoded space "%20"

CAN-2001-0778 - Source disclosure via trailing encoded space "%20"

CAN-2001-1248 - Source disclosure via trailing encoded space "%20"

CAN-2004-0280 - Source disclosure via trailing encoded space "%20"

CAN-2004-2213 - Source disclosure via trailing encoded space "%20"

CAN-2005-0622 - Source disclosure via trailing encoded space "%20"

CAN-2005-1656 - Source disclosure via trailing encoded space "%20"

CAN-2002-1603 - Source disclosure via trailing encoded space "%20"

CVE-2001-0054 - Multi-Factor Vulnerability (MVF). Directory traversal and other issues in FTP server using Web encodings such as "%20"; certain manipulations have unusual side effects.

CAN-2002-1451 - Trailing space ("+" in query string) leads to source code disclosure.

- In the form of ' filedir' (Leading space) path equivalence.
- In the form of ‘file (space) name’ (Internal space) path equivalence.

CAN-2000-0293 - Filenames with spaces allow arbitrary file deletion when the product does not properly quote them; some overlap with path traversal.

CVE-2001-1567 - "+" characters in query string converted to spaces before sensitive file/extension (internal space), leading to bypass of access restrictions to the file.

- In the form of ‘/./’ path equivalence. CVE-2000-0004, CAN-2002-0304, BID:6042, CAN-2002-0112, CAN-1999-1083, CAN-2004-0815 - "/.////etc" cleansed to ".///etc" then "/etc
- In the form of ‘filedir*’ path equivalence.

CAN-2004-0696 - List directories using desired path and "*"

CAN-2002-0433 - List files in web server using "*.ext"

## 1.4 Path Manipulation Attack

Path Manipulation attack might occur in web application if an attacker can manipulate the request parameter which specifies a path used in some operation on file system and by manipulating the parameter, if web application run with enough privileges, an attacker would gain a capability to change, or rewrite the specified resource. The example code snippet shows this kind of weakness.

*String fileName = request.getParameter("fileName");*

*File file = new File("/usr/local/workingfiles/"+fileName);*

…

*FileOutputStream fileStream = new FileOutputStream("file);*

In this code snippet, the developer assumes that it gets a parameter *"fileName"* specifying a valid file and does some modification on this file according to business rule and did not consider whether this parameter can be changed my malicious user or not. When an attacker finds this weakness, he could send a malicious HTTP request with *"fileName"* with value *"../../tomcat/conf/server.xml"* and since this code has not any checks about the parameter, application server configuration file can be overwritten and web application would break down. Path manipulation attack's likelihood is high to very high according to CWE List and as shown in the example it would cause extremely severe results.

## 1.5 Special Element Injection

This category deals with various problems that involve special elements such as reserved word and special characters. The main problem area of this category is parsing errors that comes with using special characters and reserved words with in request parameters. Most of these vulnerabilities are because of poor coding practices. Although SQL injection and cross site scripting are also a kind of special element injection, since they are technology specific, they will be described in their own categories.

The attacker tries to break the code by inserting various special characters or reserved words in valid request parameters. The most harmless impact is information leakage by breaking the execution for example getting the error code or stack trace. After collecting the system information, the attacker could try to bypass authentication and authorization or insert malicious code into the web application considering the various methods that can be used for special element insertion attacks; CWE declares the likehood of exploit of special element insertion attacks as high to very high and dangerousness of impacts as high.

Some commonly used attacks techniques and reported attacks are listed as follows;

- By inserting ‘Parameter Delimiter’. The attacker inserts field separator into input parameter.

CAN-2003-0307 - attacker inserts field separator into input to specify admin privileges.

- By inserting ‘Value Delimiter’. The attacker inserts delimiters between values.

CAN-2000-0293 - multiple internal space, insufficient quoting - program does not use proper delimiter between values

- By inserting ‘Record Delimiter’. The attacker inserts carriage returns and ‘|’ fields separator to insert more and malicious records to the system.

CAN-2004-1982 - carriage returns in subject field allow adding new records to data file

CVE-2001-0527 - attacker inserts carriage returns and "|" field separator characters to add new user/privileges.

- By inserting ‘Line Delimiter’. The attacker inserts line breaks to insert malicious input to the system.

CVE-2002-0267 - linebreak in field of PHP script allows admin privileges when written to data file.

- By inserting ‘Section Delimiter’. One example of a section delimiter is the boundary string in a multipart MIME message.

- By inserting ‘Input Terminator’. If an application parses input using special input delimiters, an attacker could break down the code by inserting false input delimiters. CVE-2000-0319, CVE-2000-0320 - MFV. mail server does not properly identify terminator string to signify end of message, causing corruption, possibly in conjunction with off-by-one error.

CAN-2001-0996 - mail server does not quote end-of-input terminator if it appears in the middle of a message.

CAN-2002-0001 - improperly terminated comment or phrase allows commands..

- By inserting ‘Input Leader’. If an application uses special input leader characters representing start of the input, an attacker could break down the code by inserting false input leaders.

- By inserting ‘Quoting Element’. If an application allows quoting elements, an attacker could try to break down the code by inserting duplicate quotes or missing leading/trailing quotes.

CAN-2003-1016 - MIE. MFV tool bypass AV/security with fields that should not be quoted, duplicate quotes, missing leading/trailing quotes.

- By inserting ‘Escape, Meta or Control Sequence’ If an application uses special escape, meta or control sequence characters, an attacker could break down the code, change execution by inserting malicious characters or commands.

CVE-2002-0542 - mail program handles special "~" escape sequence even when not in interactive mode.

CVE-2000-0703 - setuid program does not filter escape sequences before calling mail program.

CVE-2002-0986 - mail function does not filter control characters from arguments, allowing mail message content to be modified.

CVE-2003-0020, CAN-2003-0083 - Terminal escape sequences not filtered from log files.

CVE-2003-0021, CVE-2003-0022, CVE-2003-0023, CVE-2003-0063, CAN-2000-0476 - terminal escape sequences not filtered by terminals when displaying files.

CAN-2001-1556 - MFV. (multi-channel). Injection of control characters into log files that allow information hiding when using raw Unix programs to read the files.

- By inserting ‘Comment Element’. The attacker could attack the application by inserting duplicate comment elements or missing leading/trailing comment elements. Mostly used for cross site scripting.

CAN-2002-0001 - mail client command execution due to improperly terminated comment in address list

CAN-2004-0162 - MIE. RFC822 comment fields may be processed as other fields by clients.

CAN-2004-1686 - well-placed comment bypasses security warning

CAN-2005-1909, CAN-2005-1969 - information hiding using a manipulation involving injection of comment code into product.

- By inserting ‘Variable Name Delimiter’. The attacker could insert special characters such as ‘$’, ‘%’ to bypass the black list of available commands.

CAN-2005-0129 - "%" variable is expanded by wildcard function into disallowed commands.

CAN-2002-0770 - server trusts client to expand macros, allows macro characters to be expanded to trigger resultant infoleak.

- By inserting ‘Wildcard or Matching Element’. The attacker could insert wildcard or matching element, which could result in unexpected behaviors. Most used for SQL injection.

CAN-2002-0433, CAN-2002-1010 - bypass file restrictions using wildcard character

CVE-2001-0334 - wildcards generate long string on expansion

CAN-2004-1962 - SQL injection involving "/**/" sequences

- By inserting ‘White Space Elements’. The attacker could insert white space characters into the input and these characters could overlap separator characters or delimiters.

CAN-2002-0637 - Virus protection bypass with RFC violations involving extra whitespace, or missing whitespace.

CAN-2004-0942 - CPU consumption with MIME headers containing lines with many space characters, probably due to algorithmic complexity (RESOURCE.AMP.ALG).

CAN-2003-1015 - Whitespace interpreted differently by mail clients.

- By inserting ‘Grouping Element / Paired Delimiter’. If an application does not properly handle the characters that are used to mark the beginning and ending of a group of entities, such as parentheses, brackets, and braces, the attacker can break down the code by inserting or deleting these characters from input causing crashes and buffer overflows.

CAN-2004-0956 - crash via missing paired delimiter (open double-quote but no closing double-quote)

CVE-2000-1165 - crash via message without closing ">"

CVE-2005-2933 - buffer overflow via mailbox name with an opening double quote but missing a closing double quote, causing a larger copy than expected

- By inserting ‘Null Character / Null Byte’. Inserting null characters can result in various interpretation errors. The application could parse failing parsing the input.

CAN-2005-2008, CVE-2005-3293 - source code disclosure using trailing null

CAN-2005-2061 - trailing null allows file include

CAN-2002-1774 - null character in MIME header allows detection bypass

CVE-2004-0189 - decoding function in proxy allows regular expression bypass in ACLs via URLs with null characters

CVE-2005-3153, CVE-2005-4155 - null byte bypasses PHP regexp check

## 1.6 Command Injection

Command Injection attacks are subset of injection attacks, in which the attacker manipulates the request parameters to control the calling external processes. Dynamically generating operating system

commands that include user input as parameters can lead to command injection attacks. An attacker can insert operating system commands or modifies the command that will be executed. CWE declares the likehood of exploit of command injection attacks as high to very high and dangerousness of impacts as very high. Following example shows how a command injection occurs;

*String operationType = request.getParameter("opttype");*

*String cmdToExecute = new String ("bash /usr/local/somebatchjop.sh " +opttype;*

*System.Runtime.getRuntime().exec(cmd);*

In this example, the developer wants to get some command parameter from the request and tries to execute some batch process. (For example, to do backup operation of an administrative web application.) However an attacker can manipulate *opttype* parameter and adds "& rm -rf /usr/local/JBoss", and tries to delete the application server folder. Even *Runtime.exec()* command executes only one command per call. Creating bash shell enables executing multiple commands. With this vulnerably code, an attacker can execute whatever system commands, as he wants.

Command injection vulnerabilities occur when these three conditions are satisfied: 1. Input of the application enters from an untrusted source. 2. The data is part of a string that is executed as a command by the application. 3. By executing the command, the application gives an attacker a privilege that the attacker would not otherwise have.

## 1.7 Argument Injection or Modification

Argument injection or modification vulnerability is not a vulnerability that is used for web application attacks. This vulnerability is in fact affects standalone applications that has interaction with OS commands and which takes arguments from the OS. The attacker tries to inject or modify the arguments of application so that he can gain more privileges or execute malicious code. For example, if an application is configured to take init file URL from the command line and read an init file to load some dynamic link libraries, the attacker could able to execute malicious DLL's by changing the init file location to malicious file from altering command line arguments.

However, from web application view, there is a unique but serious security flaw in Java Web Start, client-side deployment technology for java applications. Java Web Start handles java virtual machine properties defined in JNLP files. A malicious user can modify these JNLP files and pass malicious command line arguments to the Java virtual machine. They can be used to disable the Java "sandbox" and compromise the system. The attack can be carried out when the victim user views a web page crafted by the attacker.

A few system properties are considered "secure" and if defined in a JNLP file, they are passed to the Java executable (javaw.exe) via the -Dproperty=value command line argument. However, a malicious user can use this feature to inject extra command line arguments to the Java executable.

For instance, a JNLP file can contain this property tag:

<property name="sun.java2d.noddraw" value="true HELLO" />

The property "sun.java2d.noddraw" is considered secure by Web Start, so it is accepted and the startup command for the application is something like this:

javaw.exe -Dsun.java2d.noddraw=true HELLO (other args) your.application

This would produce a Web Start error message saying the main class can't be found, as javaw.exe interprets "HELLO" as the main class name instead of "your.application". The problem is that Web Start fails to use quote symbols around the property argument.

To exploit the flaw, an attacker can pass command line arguments affecting the Java security policies. Normally an unsigned, untrusted Java applet operates inside a "sandbox" and can't e.g. access local files. By exploiting this flaw, the default "sandbox" security policy can be overridden with an arbitrary policy file hosted on the attacker's web server. The new policy can grant full permissions to the application, which could then e.g. read or write arbitrary files on the victim system, or download and launch viruses, keyloggers or other malware. The attacker may set up a JNLP file on a web server so that it will be launched without further user interaction when the victim visits the site, e.g. with the IFRAME tag.

Although this attack is in fact web browser attack vector, the attacker could replace an existing JNLP file on a web site with a malicious one. So that any web application can be source of this vulnerability.

## 1.8 Resource Injection

This vulnerability enables an attacker to access or modify otherwise protected system resources. Resource injection attacks resemble path manipulation attacks so that it covers path manipulation attack, which is related to file system resources but also considers all kind of system resources such as data sources, system ports.

An application is vulnerable to resource injection attacks when these two condition occurs; 1. An attacker can specify the identifier used to access a system resource. For example, an attacker might be able to specify part of the name of a file to be opened or a port number to be used. 2. By specifying the resource, the attacker gains a capability that would not otherwise be permitted. CWE declares the likehood of exploit of resource insertion attacks as high and dangerousness of impacts as very high.

## 1.9 Code Injection

Many of code injection attacks are under-studied, and terminology is not sufficiently precise according to CWE. However CWE describes tree main categories under code injection vulnerability;

### 1.9.1 Direct Dynamic Code Evaluation:

If a web application uses an interpreter and allows inputs to be fed directly into a function (e.g. "eval") that is dynamically evaluated and executed the input as code. Perl, Python and PHP technologies are among these that use an interpreter so that they are vulnerably to this kind of attacks. Some attacks are listed as follows;

CAN-2002-1750, CAN-2002-1751, CAN-2002-1752, CAN-2002-1753, CAN-2005-1527, CAN-2005-2837 are examples of direct code injection into Perl ‘eval’ function.

CAN-2005-2498 and CAN-2005-1921 are examples of MFV. code injection into PHP ‘eval’ statement using nested constructs that should not be nested.

CAN-2001-1471 is example of MFV. invalid value prevents initialization of variables, which can be modified by attacker and later injected into PHP ‘eval’ statement.

### 1.9.2 Direct Static Code Injection:

The product allows inputs to be fed directly into an output file that is later processed as code. Different from XSS or HTML injection techniques which is executed on the client side, direct static code injection vulnerability enables malicious codes to be executed at server side but this can be resultant from XSS or HTML injection because the same special characters can be involved. One example of direct static code injection is Server-Side Includes (SSI) injection.

SSI Injection (Server-side Include) is a server-side exploit technique that allows an attacker to send code into a web application, which will later be executed locally by the web server. SSI Injection exploits a web application's failure to sanitize user-supplied data before they are inserted into a server-side interpreted HTML file. Before serving an HTML web page, a web server may parse and execute Server-side Include statements before providing it to the user. In some cases (e.g. message boards, guest books, or content management systems), a web application will insert user-supplied data into the source of a web page. If an attacker submits a Server-side Include statement, he may have the ability to execute arbitrary operating system commands, or include a restricted file's contents the next time the page is served.

The following SSI tag can allow an attacker to get the root directory listing on a UNIX based system.

< !--#exec cmd="/bin/ls /" -- >

The following SSI tag can allow an attacker to obtain database connection strings, or other sensitive data contained within a .NET configuration file.

<!--#INCLUDE VIRTUAL="/web.config"-->

Some direct static code injection attacks are listed as follows;

CVE-2002-0495 - Perl code directly injected into CGI library file from parameters to another CGI program

CAN-2005-1876 - direct PHP code injection into supporting template file

CAN-2005-1894 - direct code injection into PHP script that can be accessed by attacker

CAN-2003-0395 - PHP code from User-Agent HTTP header directly inserted into log file implemented as PHP script.

### 1.9.3 PHP File Inclusion Attack:

This vulnerability is specific to PHP technology, however since it is likehood of exploit is considered as very high according to CVE, it is treated as an sub category of code injection attack techniques. When a PHP product uses "require" or "include" statements, or equivalent statements, that use attacker-controlled data to identify code or HTML to be directly processed by the PHP interpreter before inclusion in the script this vulnerability could be occur. Some examples of this vulnerability are;

CAN-2004-0285, CAN-2004-0030, CVE-2004-0068, CAN-2005-2157, CAN-2005-2162, CAN-2005-2198, CVE-2004-0128 are examples of modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.

CAN-2005-1864, CAN-2005-1869, CAN-2005-1870, CAN-2005-2154, CAN-2002-1704, CAN-2002-1707, CAN-2005-1964, CAN-2005-1681, CAN-2005-2086 are examples of PHP file inclusion.

CAN-2004-0127 and CAN-2005-1971 are examples of Directory traversal vulnerability in PHP include statement.

CVE-2005-3335 is example of PHP file inclusion issue, both remote and local; local include uses ".." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector.

## 1.10 LDAP Injection

Lightweight Directory Access Protocol (LDAP) is a widely used protocol for accessing information directories. LDAP injection vulnerability enables an attacker to reveal sensitive and secret information

from the system and generating authentication and authorization errors that result in more dangerous situations. If a web application does not properly filter or quote special characters or reserved words that are used in LDAP queries or responses and allows attackers to modify the syntax, contents, or commands of the LDAP query before it is executed, LDAP injection could be occurred.

LDAP injection techniques have very similar like SQL injections. Although there can several LDAP injection techniques, some example injection is stated as follows;

Consider a web application that has page called *ldap-search.jsp* that takes a parameter *userid* and return back user information. Such a code can be vulnerably to these kinds of attacks.

- Insertion of special characters like $,@ can destroy query structure and reveals technical information as error page. An attacker can gain type of LDAP implementation, line of query code. (Example http://some.site/ldap-search.jsp?userid=((##$!!) )

- Insertion of (|(cn=*) can reveal cn value of specified user. (Example http://some.site/ldap-search?userid=someuser(|(cn=*) )

- Insertion of (|(objectclass=*) can reveal list of available object classes. (Example http://some.site/ldap-search?userid=someuser(|(objectclass=*) )

- Insertion of (|(homedirectory=*) can reveal home directory of specified user. (Example http://some.site/ldap-search?userid=someuser(|(homedirectory=*) )

- Insertion of (*) can reveal home directory of specified user. (Example http://some.site/ldap-search?userid=someuser=* )

Main purpose of most of the LDAP injection attacks is revealing sensitive information. However LDAP injection can be solely used to bypassing authentication and authorization of web application and result in more dangerous consequences. CWE states that besides there are a few reported LDAP injection attacks, this vulnerability is found very frequently by third party codes.

## 1.11 SQL Injection

SQL injection attacks are one of the most dangerous instantiation of injection attacks. In this attack technique malicious SQL commands are injected into request parameters in order to effect the execution of predefined SQL commands. SQL injection attacks threats most of the subjects computer security.

**Confidentiality:** Most common consequence of SQL injection attacks is loss of confidentiality. Since SQL databases hold sensitive data, unauthorized access to these data could generate more dangerous consequences.

**Authentication:** Most of the applications use SQL databases for storing authentication data. If a SQL injection occurs in authentication part of the system, all authentication mechanism can be bypassed by the attacker.

**Authorization:** Authorization modules that use SQL database are another critical part of the web application. If they are vulnerably to SQL injection attacks, it would be possible to change authorization information and a security breach can be opened for an application.

**Integrity:** By SQL injection, it is also possible to make changes or deletions that threats integrity of whole database.

There are various SQL insertion techniques, however we can categories these techniques into five; insertion using multiple SQL statements, authorization bypass, using SELECT command, using INSERT command, using stored procedures.

**Insertion using multiple SQL statements:** Although not all database servers is vulnerably to insertion using multiple SQL statements, some important ones such as Microsoft® SQL Server 2000 allows multiple SQL statements separated by semicolons to be executed at once, as a result becomes vulnerably. This type of attack allows the attacker to execute arbitrary commands against the database. A typical example of this attack is shown below;

*String userId=Request.getParameter("userid");*

*String itemNo=Request.getParameter("itemno");*

*String sqlQuery = "SELECT * FROM items WHERE owner= '"+userId+" ' AND itemno= "+itemNo;*

*Statement.executeQuery(sqlQuery);*

Assume that the regular execution of this code from a web application is http://somesite.com/searchitems.jsp?userid=someuser&itemno=5 which selects *someuser's* items with item no 5. However if an attacker generates this request as http://somesite.com/seachitems.jsp ?userid=someuser&itemno=5;DELETE FROM items;-- Then the query to be executed becomes SELECT * FROM items WHERE owner='someuser' AND itemno=5; DELETE FROM items; now there are two distinct sql statement to executed by database server sequentially, which result in deleting all items from database.

**Authorization bypass:** The simplest SQL injection technique type is bypassing logon form. The code in the following example shows vulnerably code.

*String sqlQuery = "SELECT username FROM users WHERE username = '" & strUsername & "' AND password = '" & strPassword & "'"*

*Statement.executeQuery(sqlQuery);*

*// if statement has return some rows*

*If rowCount >0 boolAuthenticated = False;*

*else*

*boolAuthenticated = True;*

A valid request for this code is http://somesite.com/logon.jsp?userid=someuser&password=somepassword. However if an attacker supplies userid field with ‘ OR 1=1 and password field with ‘ OR 1=1 then this will give sqlQuery the following values;

SELECT username FROM users WHERE username= ‘’ OR 1=1 AND password=’’ OR 1=1

This is a valid SQL statement and returns all usernames from users table and the code only checks that if that user exists, this request would bypass this authentication check. If an attacker knows a valid username, he can supply userid field with that user and password field with ‘OR 1=1 so that he can login to the system as that user without knowing his password, so that an attacker now has all privileges to do any operation of a valid user.

**Using SELECT command:** The most dangerous injections that threaten confidentiality of the web application are result from select command attacks. There are various techniques that depend on the coding structure of web application; however underlying attack manner is the same for all kinds of attacks. Firstly an attacker tries to reveal the SQL query structure of the web page. This can be done trying lots of quotes, parenthesis, WHERE, OR statement combinations. For example if the related SQL query is as follows;

*SQLString = "SELECT FirstName, LastName, Title FROM Employees WHERE Employee = " & intEmployeeID*

The injection will be simple adding ‘OR 1=1 will enough to allow injection, however if the SQL query is like this:

*SQLString = "SELECT FirstName, LastName, Title FROM Employees WHERE EmployeeID = ‘" & strCity & "‘"*

The injection can be done by adding ‘ OR ‘1’=’1 to get rid of syntax errors. An attacker tries different combination to determine the structure. When he receives a blank page, or a valid page, the injection is successful. The next step of attack is inserting UNION statement to that query. For example if some part of the web application uses the following code;

*mySQL="SELECT LastName, FirstName, Title, Notes, Extension FROM Employees WHERE (City = ‘" & strCity & "‘)"*

so when an attacker injects this value;

*“‘) UNION SELECT OtherField FROM OtherTable WHERE (‘‘=‘”,*

now the following query is send to server, which is a valid SQL query;

*SELECT LastName, FirstName, Title, Notes, Extension FROM Employees WHERE (City = ‘‘) UNION SELECT OtherField From OtherTable WHERE (‘‘=‘‘)*

It is valid request and database server only complains about a bad table name, so the next step is choosing a valid system table name for example for MS SQL server these are sysobjects syscolumns or for Oracle SYS.USER_OBJECTS SYS.TAB SYS.USER_TABLES SYS.USER_VIEWS SYS.ALL_TABLES SYS.USER_TAB_COLUMNS SYS.USER_CONSTRAINTS SYS.USER_ TRIGGERS SYS.USER_CATALOG. After that all an attacker must do is finding the exact column number and type, he can do this by trying various injections like;

‘UNION ALL SELECT 9,9 FROM SysObjects WHERE ‘=‘

‘UNION ALL SELECT 9,9,9 FROM SysObjects WHERE ‘=‘

‘UNION ALL SELECT 9,9,9,9 FROM SysObjects WHERE ‘=‘

Now the injection is successfully done, and he can reveal all table names from SYS.USER_TABLES and find corresponding columns from SYS.USER_TAB_COLUMNS so that he can query any table in that web application.

**Using the INSERT command:** The insert command can also be used for revealing sensitive information. Common uses of INSERT in web application are user registrations, bulleting boards, adding items to shopping carts, etc. To take advantage of an INSERT vulnerability, an attacker must be able to view the information that he has submitted. Consider an example user registration form with following SQL string;

SQLString = "INSERT INTO userregistration VALUES (‘" & strUserName & "‘, ‘" & strUserMail & "‘, ‘" & strUserPhone & "‘)"

If an attacker fill out the form like this;

Name: ‘ + (SELECT TOP 1 FieldName FROM TableName) + ‘

Email: blah@blah.com

Phone: 333-333-3333

Then the insert query becomes;

INSERT INTO userregistration VALUES (‘‘ + (SELECT TOP 1 AnyFieldName FROM AnyTableName) + ‘‘, ‘blah@blah.com’, ‘333-333-3333’)

So that when the user list his registration information, he can now see his selection query (SELECT TOP 1 AnyFieldName FROM AnyTableName) result in the name value so that he can reveal any information from the web application.

### Using stored procedures:

SQL injection can be used for accessing stored procedures and lead to more dangerous consequences. All database servers have already built-in stored procedures which can be used for reporting, management, monitoring activities and these procedures can be used using SQL injection depending on the permissions of the web application’s database user. Two most dangerous stored procedures of MS SQL server that can be used are **xp_cmdshell** and **sp_makewebtask.** xp_cmdshell takes a single argument which is the command to be executed in SQL server’s user shell. Using xp_cmdshell, an attacker can executed any command such as deletion of sensitive data or broken down the whole database server. If an attacker makes a request to a JSP page which has SQL injection vulnerability as shown below; he can delete entire disk.

http://somesite.com/search.jsp?userid=someuser’ ;EXEC master.dbo.xp_cmdshell ‘cmd.exe delete c:*

While xp_cmdshell threatens integrity, sp_makewebtask threatens confidentiality. sp_makewebtask takes first argument as output file and second argument as SQL query to be executed. So if an attacker manage to execute sp_makewebtask procedure, he can report any SQL query to a file which has public access. Afterwards he can request previously created file as HTTP request. As an example using query below, he can generate a web page that list all customer information.

http://somesite.com/seach.jsp?userid=someuser’;EXECmaster.dbo.sp_makewebtask‘\public\output.html; ‘SELECT * from Customers’

SQL injection attacks are one of the most common and most easy to generate security exploits of web applications. Not only application that directly access database servers but also some commonly used technologies for accessing database servers like Hibernate or HibersonicSQL are also vulnerable to SQL injection attacks.

## 1.12 Cross Site Scripting (XSS) Attacks

Nowadays, all web application depends on dynamic page generation which requires user input to change behavior of a web page. Without proper input validation, web applications are easy to vulnerable from XSS attacks. A web application is vulnerable to XSS attacks when they allow

injection of malicious scripts as inputs of user and as a result of generating dynamic pages from this infected input, these malicious scripts could be executed from client browsers and could affect all web site clients. Although secure execution of JavaScript code is based on a *sandboxing mechanism*, which allows the code to perform a restricted set of operations only and JavaScript programs downloaded from different sites are protected from each other using a compartmentalizing mechanism, called the *same-origin policy*, scripts may be confined by the sand-boxing mechanisms and conform to the same-origin policy, but still violate the security of a system. This can be achieved when a user is lured into downloading malicious JavaScript code (previously created by an attacker) from a trusted web site.

Two main classes of XSS attacks exist: stored attacks and reflected attacks. In a stored XSS attack, the malicious JavaScript code is permanently stored on the target server (e.g., in a database, in a message forum, in a guestbook, etc.). In a reflected XSS attack, on the other hand, the injected code is "reflected" off the web server such as in an error message or a search result that may include some or all of the input sent to the server as part of the request. Reflected XSS attacks are delivered to the victims via e-mail messages or links embedded on other web pages. When a user clicks on a malicious link or submits a specially crafted form, the injected code travels to the vulnerable web application and is reflected back to the victim's browser. A typical reflected cross site scripting scenario is shown in following figure.

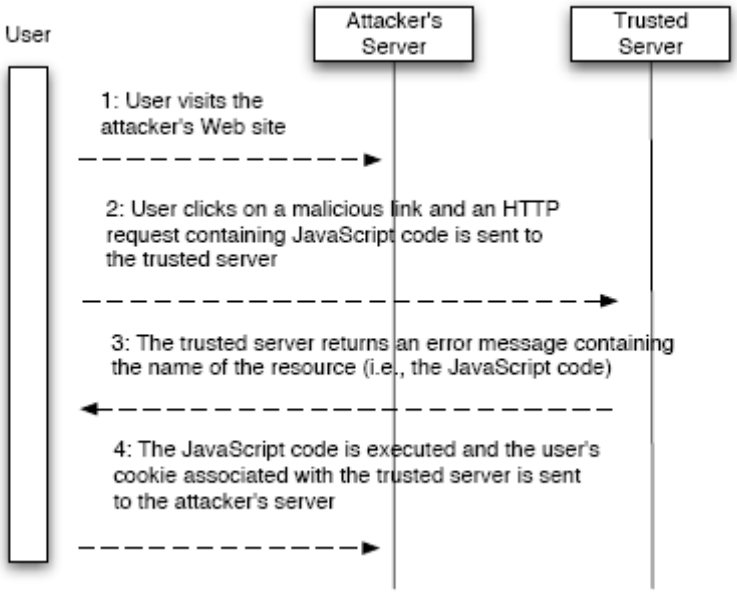


**Figure 38 XSS Attack**

Some XSS attack techniques are;

**Basic XSS:** Basic XSS involves, web applications that involves lack of filtering of any special characters, such as “<” “>” and “&”. This exploit is very common and it is the easiest technique of XSS attack. Some observed example of basic XSS attacks are;

CVE-2002-0938 - XSS attack using a parameter in a link.

CAN-2002-1495 - XSS attack via attachment filenames in web-based email product.

CAN-2003-1136 - HTML injection in posted message.

CAN-2004-2171 - XSS attack result from not quoted in error page.

**XSS in Error Pages:** This Weakness occurs when a web developer displays input on an error page (e.g. a customized 403 Forbidden page). If an attacker can influence a victim to view/request a web page that causes an error, then the attack may be successful. Some observed examples are;

CVE-2002-0840 - XSS attack in default error page from Host: header.

CVE-2002-1053 - XSS attack in error message.

CAN-2002-1700 - XSS attack in error page from targeted parameter.

**Script in IMG Tags:** An attacker could attack web application in the form of HTML IMG tags. Attackers can embed XSS exploits into the values for IMG attributes (e.g. SRC) that is streamed and then executed in a victim's browser. Some observed examples are;

CAN-2002-1649, CAN-2002-1803, CAN-2002-1804, CAN-2002-1805, CAN-2002-1806, CAN-2002-1807, CAN-2002-1808.

**XSS Using Script in Attributes:** XSS attacks can be inserted in a web page using dangerous attributes within tags such as “onmouseover”, “onload”, “onerror”, or “style”. Some observed examples are;

CAN-2001-0520 – XSS attack by bypassing filtering of SCRIPT tags using onload in BODY, href in A, BUTTON, INPUT.

CVE-2002-1493 – XSS attack on guestbook in STYLE or IMG SRC attributes.

CAN-2002-1965 – XSS attack using Javascript in onerror attribute of IMG tag.

CAN-2002-1495 - XSS attack in web-based email product via onmouseover event.

CAN-2002-1681 - XSS attack via script in <P> tag.

CAN-2003-1136 - XSS attack using Javascript in onmouseover attribute.

CAN-2004-1935 - XSS attack using onload, onmouseover, and other events in an e-mail attachment.

CAN-2005-0945 - XSS attack using Onmouseover and onload events in img, link, and mail tags.

CAN-2003-1136 - XSS attack using Onmouseover attribute in e-mail address or URL.

**XSS using script via encoded URI schemes:** Although web application uses filtering of malicious scripts, an attacker could cloak the script using URI encodings. Some observed examples are;

CAN-2005-0563 - Cross-site scripting (XSS) vulnerability in Microsoft Outlook Web Access (OWA) component in Exchange Server 5.5 allows remote attackers to inject arbitrary web script or HTML via an email message with an encoded javascript: URL ("jav&#X41sc
ript:") in an IMG tag.

CAN-2005-2276 - Cross-site scripting (XSS) vulnerability in Novell Groupwise WebAccess 6.5 before July 11, 2005 allows remote attackers to inject arbitrary web script or HTML via an e-mail message with an encoded javascript URI (e.g. "j&#X41vascript" in an IMG tag).

CAN-2005-0692 – XSS attack by bypassing the script filter using encoded script within BBcode IMG tag.

CVE-2002-0117 - XSS attack by bypassing the script filter using Encoded "javascript" in IMG tag

CAN-2002-0118 - XSS attack by bypassing the script filter using Encoded "javascript" in IMG tag.

**Doubled character XSS manipulations:** An attacker could disguise injected script tag using doubling of the “<<” character. It is a very basic kind of XSS attack, but some examples exist in CVE database such as;

CAN-2002-2086 - XSS using "<script".

CAN-2001-1157 – XSS attack using extra "<" in front of SCRIPT tag.

**Invalid characters in identifiers:** Some whitespace characters such as CLRF, null can be discarded by some web browsers so that insertion of these characters in malicious scripts could bypass script filtering of web application but since some web browsers discard these characters, the injected script would be executed correctly.

CAN-2004-0595 – On this attack XSS filter doesn't filter null characters before looking for dangerous tags, which are ignored by web browsers.

**Alternate XSS syntax:** An attacker could try to bypass script filter by alternating XSS syntax, an if a web application's script filter could not detect insertion of alternate script syntax, then an attacker could inject malicious script to success the attack. One example of successful XSS attack is CVE-2002-0738 where the attacker inject the script in the form of &={script}.

## 1.13 XML Injection

XML injection is similar to SQL injection vulnerability. It occurs when web site uses user supplied information to query XML data. By sending malformed information into the web site, an attacker can find out how the XML data is structured or access data that they may not normally have access to just like SQL injection. Although XML injection uses for loss of confidentially, it can be used for bypassing authentication and authorization if these modules depends on XML data.

Querying XML is done with XPath, a type of simple descriptive statement that allows the xml query to locate a piece of information. When using XML for a web site it is common to accept some form of input on the query string to identify the content to locate and display on the page.

Although all of the insertion techniques that is described in SQL injection section can be also used, a simple example can be given as a web application that authenticate users depends on XML document that has xml snippet as given below;

```
<?xml version="1.0" encoding="utf-8"?>
<Employees>
  <Employee ID="1">
    <FirstName>First User Name </FirstName>
    <LastName>First User Last Name</LastName>
    <UserName>FirstUser</UserName>
    <Password>somepassword1</Password>
    <Role>Admin</Role>
  </Employee>
  <Employee ID="2">
    <FirstName> Second User Name </FirstName>
    <LastName> Second User Last Name </LastName>
    <UserName>SecondUser</UserName>
    <Password>somepassword2</Password>
    <Role>User</Role>
  </Employee>
</Employees>
```

Consider this web application has a login form that post username and password fields as request parameters and web application tries to match this username, password data to match record of XML document.

```
String username = Request.getParameter("username");

String password = Request.getParameter("password");

String findUserXPath = "//Employee[UserName/text()='" + username + "' And
```

Password/text()='" + password + "']";

This works fine, if users enter valid username and password, then it would fetch corresponding record, if not it would return nothing. But if a malicious user inject username field with *someuser' or 1=1 or 'a'='a*, then XPath query becomes *//Employee[UserName/text()='someuser' or 1=1 or 'a'='a' And Password/text()='']* and this is logically equivalent to *//Employee[(UserName/text()='someuser' or 1=1) or ('a'='a' And Password/text()='')]*. In this case, only the first part of the XPath needs to be true. The password part becomes irrelevant, and the UserName part will match ALL employees because of the "1=1" part. So it will allow an attacker to login as any user in the system without supplying a valid password.

Although there are very few publicly reported examples in CVE database, XML injection vulnerability is as serious and dangerous vulnerability as SQL injection that threatens mostly confidentiality of web applications.

## 1.14 Missing XML Validation

XML validation is an important concept for web application. Since nearly all web application requires XML form of data for processing or integrating without proper validation web application becomes vulnerably to be affected by code or data injection. To be vulnerably a web application is not necessary to accept XML as a user supplied input, most of the times XML form of data is dynamically generated and passes to other parts of the web application to be directly parsed and executed. So if a web application uses XML form of data, it might be vulnerably to malicious code injection even if it does not expect XML input from user.

So all XML data must be validated by DTD or XML schema, by accepting an XML document without validating it against a DTD or XML schema, it is possible to provide unexpected, unreasonable, or malicious input.

## 1.15 HTTP Response Splitting

HTTP response splitting is one of the most dangerous attacks that lead to many different type of application attacks and if not properly handled threaten nearly all concepts of computer security. By HTPP response splitting, an attacker made application server to generate two or more HTTP response for one valid user request and one or more of these responses are malicious and gives an attacker to full control the response as if it is a valid response that comes from a legimate site.

If an application allows writing unvalidated data into an HTTP header, this would be result for an attacker to specify the entirety of the HTTP response rendered by the browser. HTTP response splitting vulnerabilities occur when:

1. Data enters a web application through an untrusted source, most frequently an HTTP request.

2. The data is included in an HTTP response header sent to a web user without being validated for malicious characters. For example the application must allow input that contains CR (carriage return, also given by %0d or \r) and LF (line feed, also given by %0a or \n)characters into the header.

As a result of injecting these characters in the header of a valid response it will not only give attackers control of the remaining headers and body of the response the application intends to send, but also allows them to create additional responses entirely under their control.

A typical HTTP response splitting example can be as follows; consider a web application that receives an input from the request and tries to set it to a cookie header of an HTPP response.

String userSelection = request.getParameter(“selectedvalue”);

...

Cookie cookie = new Cookie(“selection”, userSelection);

cookie.setMaxAge(cookieExpiration);

response.addCookie(cookie);

Here, the developer assumes that *“selectedvalue”* is under his control and it will consist of standard alpha-numeric characters, such as "BlueTheme" (assume that this user selection represents a site theme selection), is submitted in the request the HTTP response including this cookie might take the following form:

HTTP/1.1 200 OK ... Set-Cookie: selectedvalue=BlueTheme ...

If an attacker submits a malicious string, such as "BlueTheme\r\nHTTP/1.1 200 OK\r\n...", then the HTTP response would be split into two responses of the following form:

HTTP/1.1 200 OK ... Set-Cookie: author= selectedvalue=BlueTheme

HTTP/1.1 200 OK ... –Malicious Response-

The second response is completely controlled by the attacker and can be constructed with any header and body content desired. The ability of attacker to construct arbitrary HTTP responses permits a variety of resulting attacks, including: cross-user defacement, web and browser cache poisoning, cross-site scripting and page hijacking.

**Cross-User Defacement:** An attacker can make a single request to a vulnerable server that will cause the sever to create two responses, the second of which may be misinterpreted as a response to a different request, possibly one made by another user sharing the same TCP connection with the server. This can be accomplished by convincing the user to submit the malicious request themselves, or

remotely in situations where the attacker and the user share a common TCP connection to the server, such as a shared proxy server. In the best case, an attacker can leverage this ability to convince users that the application has been hacked, causing users to lose confidence in the security of the application. In the worst case, an attacker may provide specially crafted content designed to mimic the behavior of the application but redirect private information, such as account numbers and passwords, back to the attacker.

**Cache Poisoning:** The impact of a maliciously constructed response can be magnified if it is cached either by a web cache used by multiple users or even the browser cache of a single user. If a response is cached in a shared web cache, such as those commonly found in proxy servers, then all users of that cache will continue receive the malicious content until the cache entry is purged. Similarly, if the response is cached in the browser of an individual user, then that user will continue to receive the malicious content until the cache entry is purged, although the user of the local browser instance will be affected.

**Cross-Site Scripting:** Once attackers have control of the responses sent by an application, they have a choice of a variety of malicious content to provide users. Cross-site scripting is common form of attack where malicious JavaScript or other code included in a response is executed in the user's browser. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site. The most common and dangerous attack vector against users of a vulnerable application uses JavaScript to transmit session and authentication information back to the attacker who can then take complete control of the victim's account.

**Page Hijacking:** In addition to using a vulnerable application to send malicious content to a user, the same root vulnerability can also be leveraged to redirect sensitive content generated by the server and intended for the user to the attacker instead. By submitting a request that results in two responses, the intended response from the server and the response generated by the attacker, an attacker can cause an intermediate node, such as a shared proxy server, to misdirect a response generated by the server for the user to the attacker. Because the request made by the attacker generates two responses, the first is interpreted as a response to the attacker's request, while the second remains in limbo. When the user makes a legitimate request through the same TCP connection, the attacker's request is already waiting and is interpreted as a response to the victim's request. The attacker then sends a second request to the server, to which the proxy server responds with the server generated request intended for the victim, thereby compromising any sensitive information in the headers or body of the response intended for the victim.

## 1.16 Process Control

Process control vulnerabilities take two forms;

1. An attacker can change the command that the program executes: the attacker explicitly controls what the command is. If data enters the application from an untrusted source and if the data is used as or as part of a string representing a command that is executed by the application, by manipulating the input string it is possible that the application can call untrusted malicious code and gives an attacker a privilege or capability that the attacker would not otherwise have.

2. An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means. An example of this is if an application loads a native library from the environment, it is possible that an attacker can replace the library or insert a malicious with the same name of the indented library and take control of execution. For example if an application loads a library using System.loadlibrary(library.dll) which takes only library name not absolute path, the mapping from a library name to a specific filename is done in a system-specific manner. If an attacker is able to place a malicious copy of library.dll higher in the search order than file the application intends to load, then the application will load the malicious copy instead of the intended file.

## 1.17 Log Forging

Log forging attack does not threaten the integrity or confidentially of a web application, in fact it does not directly aim web application, instead it forge log entries or inject malicious content into logs. Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. Depending on the nature of the application, the task of reviewing log files may be performed manually on an as-needed basis or automated with a tool that automatically culls logs for important events or trending information. By inserting malicious, most of the time garbage entries, an attacker can misdirect reviews of logs and makes them useless.

Log forging vulnerabilities occur when: 1. Data enters an application from an untrusted source. 2. The data is directly written to an application or system log file.

As an example of Log forging, assume that a web application tries to log all authentication requests as in following code.

```
String userid = request.getParamater(“userid”);
String password = request.getParamater(“password”);
…
Boolean isAuthenticate = authenticate(userid,password);
Logger.info(“LOGIN USERID= “+userid”+”,”+isAuthentice);
…
And when a user logout logs as;
Logger.infor(LOGOUT USERID=”+userid);
In a normal case, the authentication log will be like following;
…
```

LOGIN USERID=e112901,true
LOGIN USERID=maliciousUser,false
LOGIN USERID=e112089,false
LOGOUT USERID=e112901
…

However, if an attacker post userid parameter as

maliciousUser%0aLOGIN%20USERID=e112901%2cfalse%0aLOGIN%20USERID=e112901%2ctrue%0aLOGOUT%20USERID=e112901

then the log becomes; (The injected code is shown in bold)

…
LOGIN USERID=e112901,true
**LOGIN USERID=maliciousUser,**
**LOGIN USERID=e112901,true**
**LOGOUT USERID=e112901,false**
LOGIN USERID=e112089,false
LOGOUT USERID=e112901

So by log forging it is possible to ruin the log files. In the most case, an attacker may be able to insert false entries into the log file by providing the application with input that includes appropriate characters. If the log file is processed automatically, the attacker can render the file unusable by corrupting the format of the file or injecting unexpected characters. A more subtle attack might involve skewing the log file statistics. Forged or otherwise, corrupted log files can be used to cover an attacker's tracks or even to implicate another party in the commission of a malicious act. In the worst case, an attacker may inject code or other commands into the log file and take advantage of vulnerability in the log processing utility.

## 1.18 Buffer and Numeric Errors

Buffer overflow attacks and numeric errors are not directly threaten web applications itself, but threatens web servers or application server products. An attacker use buffer overflows to corrupt execution stack of web application. By sending malicious input to a web application, an attacker can inject malicious codes and cause the web application to execute them. Buffer overflows and numeric errors found widely in server products and can pose significant risk to users of these products. Although it is unlikely to find buffer overflows in web application itself, it is still possible especially for web applications that use third party dynamic link libraries or shared objects such as graphics library to generate images or reporting tools.

The reason behind buffer overflow and numeric errors threaten especially application servers not directly web application is that application servers that have buffer overflow vulnerabilities are publicly known and even if there is no report about buffer overflow attack, an attacker can easily try buffer overflow attack techniques on application server since errors that is generated in application

server code is directly reflected to attacker which gives an attacker important clues, on the other hand for a web application, if properly configured, the ability to exploit the flaw is significantly reduced by the fact that the source code and detailed error messages for the application are not visible to the attacker.

Some common buffer overflow flow and numeric error attack techniques are; stack overflow attack, heap overflow attack, buffer underwrite, buffer overwrite, unchecked array indexing, length parameter inconsistency, format string vulnerability, improper string length checking, integer overflow, integer underflow, integer coercion error, sign extension error, signed to unsigned conversion error, unsigned to signed conversion error, numeric truncation error numeric byte ordering error. [CWE ]

Although all known web servers, application servers are vulnerably to buffer overflows, Java and J2EE environments, web application that use these technologies, are immune to these attacks.

## 1.19 Cleansing, Canonicalization and Comparison Errors

Web applications, does cleansing and filtering for validation of data and also canonicalize the names of resources. However inappropriate combination of these steps might lead to overlooking of possible malicious attempts. Malicious codes or unacceptable input may disguise from the filtering and black listing mechanisms of web applications. Misinterpreting the input when they are differently encoded is also studied under this category.

**Encoding Errors:** If a web application does not properly handle input when an input has been modified to use encoding, this can result in overlooking of malicious attacks. An attacker could try to hide malicious data by trying different kind of encodings, mixing or doubling encodings. In CWE, encoding errors are discussed in five categories; Alternate Encoding, Double Encoding, Mixed Encoding, Unicode Encoding, URL Encoding.

**Case Sensitivity Errors:** If a web application fails to handle case sensitive data, this can lead to several possible consequences; case-insensitive passwords will reduce the size of the key space and makes brute force attacks easier, an attacker can bypass filters or access controls using alternate names with lowercase, uppercase, mixed case, multiple interpretation errors using alternate names.

**Early Validation Errors:** If validation of data step is done before cleansed or canonicalized, validation would be susceptible to various manipulations that result in dangerous inputs that are produced by canonicalization and cleansing.

**Collapse of Data into Unsafe Value:** If a web application cleanses or filters data in a way that causes the data to "collapse" into an unsafe value, it might lead to various vulnerabilities. Some examples are;

CAN-2004-0815 - "/.////" in pathname collapses to absolute path.

CVE-2005-3123 - "/.//../////././" is collapsed into "/.././" after ".." and "//" sequences are removed.

CAN-2002-0325 - ".../...//" collapsed to "..." due to removal of "./" in web server.

CAN-2002-0784 - "///../.../" claimed to work - "./" removal would produce "///..."

CAN-2005-2169 - Regular expression intended to protect against directory traversal reduces ".../...//" to "../".

**Partial Comparison Errors:** If a web application evaluate user input as only partially compared to the desired input before a match is determined. An attacker can find a way to bypass security checks. For example, an attacker might succeed in authentication by providing a small password that matches the associated portion of the larger, correct password.

## 1.20 Information Leak

A system information leak occurs when system data or debugging information leaves the program through an output stream or logging function. An attacker can cause errors to occur by submitting unusual requests to the web application. The response to these errors can reveal detailed system information, deny service, cause security mechanisms to fail, or crash the server. There are various sources of information leak. Some of the important ones are;

**Information Leak through Error Messages:** An attacker can use error messages that reveal technologies, operating systems, and product versions to tune the attack against known vulnerabilities in these technologies. The application uses diagnostic methods that provide significant implementation details such as stack traces as part of its error handling mechanism. For example, following code snippet reveals path environment variable in error message. An attacker could generate various attacks, including process control (inserting malicious dll's in path folders.).

```
String path = System.getenv("PATH");
…
System.err.println("Can not find "+filename+" on path="+path);
```

Or in following example, system exception is reflected as output, which could reveal system information, which is the most common information source for generating SQL injection attacks.

```
try {
Connection conn=getConnection
…
 }
catch(Exception ex){
ex.printStackTrace();
}
```

**Information Leak through Sent Data:** The accidental leaking of sensitive information through sent data refers to the transmission of data which are either sensitive in and of itself or useful in the further exploitation of the system through standard data channels. Most common reason of this kind of leakage is unexpected errors generated by the web application such as product error codes, especially from database vendors, such as shown below;

*Warning: mysql_pconnect(): Access denied for user: 'root@localhost' (Using password: N1nj4) in /usr/local/www/wi-data/includes/database.inc on line 4*

**Information Leak through File and Directory:** Without proper administration of web application servers, files or directories that hold sensitive information might leak through the system. Especially front hand servers are under higher risks. An attacker could scan the web servers using directory listing attack techniques and if properly not restricted, he might gain sensitive information through these files. Some sources of file and directory information leaks are backup files, core dump files, source files, log files and through CVS repository.

**Information Leak through Data Queries:** An attacker could gain information not only directly accessing but also inferring information using statistics. If a web application supplies some statistics about the system to malicious users, there is a change that an attacker could infer sensitive data from user data statistics for example by gaining online user information; an attacker could generate session-fixation or hijacking attacks aiming that users.

**Information Leak through Debug Information:** If debug information is not totally cleaned from the final product, an attacker could use these debug information to reveal sensitive information. Although developers try to hide debug information, most of the time in hidden fields of a web page, there is always high risk that it will be revealed.

**Information Leak through Caching:** If a web application does not use a restrictive caching policy for forms and web pages that potentially contain sensitive information, there is a risk that this information could be stored in a client-side cache (with most browsers) and left behind for other users to find. Malicious user could use this cached information to generate various attacks.

## 1.21 Information Loss or Omission

Information loss vulnerabilities does not directly lead to any attack, but threatens security of a web application by loosing security-relevant information that used for monitoring and auditing. CWE gives three categories under this subject;

**Truncation of Security-Relevant Information:** The application truncates the display, recording, or processing of security-relevant information in a way that can obscure the source or nature of an attack. Some observed examples of this category are;

CAN-2005-0585 - Firefox before 1.0.1 and Mozilla before 1.7.6 truncates long sub-domains or paths, facilitating phishing.

CAN-2004-2032 – Netgear RP114 bypass URL filter via a long URL with a large number of trailing hex-encoded space characters.

CAN-2003-0412 - Sun ONE Application Server 7.0 does not log complete URI of a long request (truncation).

**Omission of Security-Relevant Information:** The application does not record or display information that would be important for identifying the source or nature of an attack. Some observed examples of this category are;

CAN-1999-1029 – A web application does not record login attempts if user disconnects before maximum number of tries.

CAN-2002-1839 - Sender's IP address not recorded in outgoing e-mail.

CVE-2000-0542 - Failed authentication attempt not recorded if later attempt succeeds.

**Obscured Security-relevant Information by Alternate Name:** The software records security-relevant information according to an alternate name of the affected entity, instead of the canonical name which lead to omission of security-relevant information.

CAN-2002-0725 - Attacker performs malicious actions on a hard link to a file, obscuring the real target file.

## 1.22 Credentials Management Errors

Credential management is one of the fundamental concepts for securing web applications. Commonly, web applications handles credentials based on user id password pairs; however stronger methods of credentials management techniques such as hardware tokens are also used but such mechanisms are cost prohibitive for web applications. This kind of weaknesses occurs when a web application transmits or stores authentication credentials and uses an insecure method that is susceptible to unauthorized interception or retrieval. CWE reports these common weaknesses about credential management of web applications;

- *Storing passwords in plaintext storage* or *in configuration files* makes them open to any kind of attacks. An attacker could retrieve these password files easily and could login into system with any user's credential.

- *Storing passwords in a recoverable format* is another common weakness for web application and is no different from storing password in plaintext storage. The use of recoverable passwords significantly increases the chance that passwords will be used maliciously.

- *Unprotected transport of credentials* is also an important weakness for web applications since user credentials can also be captured during transmission from client side to server side. Without encrypting HTTP messages using SSL user passwords are vulnerable from eavesdropping or altering message contents

- *Using Weak passwords* increases the chance that passwords will be guessed. If a web application does not force strong passwords, application would be vulnerably from brute force attacks.

- *Using Hard-Coded Passwords* is common developer mistake and seriously weaken web application security. Embedding a super user password (enables developers to login as any user for debugging) or writing passwords in a source code (especially for creating database connection) are some examples of using hard-coded passwords. If attackers have access to the byte codes for application, they can use decompiler to access the disassembled code, which will contain the values of the passwords used.

- *Missing Password Field Masking* during login process will increase the potential for attackers to observe and capture passwords.

- *Weak Cryptography for Passwords* lowers the security of web application. Storing passwords with weak cryptographic methods such as Base 64 encoding would enable attackers to reconvert the passwords.

- *Not allowing password aging* is a weakness for a web application because the users will have no incentive to update passwords in a timely manner and as passwords age, the probability that they are compromised grows.

## 1.23 Permission, Privilege, and Access Control Errors

Nearly all web applications have user management module, handling a number of user groups, user roles and user permission. A successfully design of permission and access control mechanism is a must for securing a web application. Accidentally assigning an incorrect privilege to a malicious user would threaten whole security of the system. CWE reports some common weaknesses of web applications while handling permissions, privileges and access controls, these are;

- Incorrect Privilege Assignment occurs when a web application incorrectly assigns a privilege to a particular user group or role. Some observed examples are;

CVE-2005-2741 - Product allows users to grant themselves certain rights that can be used to escalate privileges.

CAN-2005-2496 - Product uses group ID of a user instead of the group, causing it to run with different privileges. This is resultant from some other unknown issue.

CVE-2004-0274 - Product mistakenly assigns a particular status to an entity, leading to increased privileges.

- Unsafe Privilege is a weakness occurs when a privilege or a role can be used to perform an operation that was not intended. Some observed examples are;

CAN-2004-2204 - Gain privileges using functions/tags that should be restricted (Accessible entities).

CAN-2004-0380 - Bypass domain restrictions using a particular file that references unsafe URI schemes (Accessible entities).

CAN-2005-1742 - Inappropriate actions allowed by a particular role(Unsafe privileged actions).

CAN-2005-2173 - Users can change certain properties of objects to perform otherwise unauthorized actions (Unsafe privileged actions).

- Privilege Chaining occurs when two or more distinct privileges or roles combined or chained together in a way that the resulting chain allows operations that would not be allowed.

- Privilege Management Error in a product is a serious error and with a buggy implementation web application becomes unable to properly track, modify, record or reset privileges.

- Privilege Context Switching Error occurs when a web application could not manage cross privilege boundaries. Examples are;

CAN-2003-1026 - Web browser cross domain problem when user hits "back" button.

CAN-2002-1770 - Cross-domain issue - third party product passes code to web browser, which executes it in unsafe zone.

- Insecure default permissions occurs when an overlooked permission is assigned default value of user roles. This vulnerability might result in various side-effects.

- Insecure inherited permissions during assignment of a user role, an overlooked permission could be gained unintentionally.

- Insecure execution-assigned permissions occurs when a web application changes or reassign permission in a insecure way that can result in side-effects.

- Access Control Bypass can be occur by using SQL attack techniques described before, An attacker could bypass access control module and could do unauthorized operations.

## 1.24 Authentication Attacks

Authentication is the key issue of web application security. Without proper authentication mechanism, all works to secure a web application becomes meaningless. A significant percentage of web application attacks are targeting to break down authentication mechanisms. Although it is known that authentication is critical and various kinds of attack can be done to bypass authentication mechanisms, still a significant percentage of web applications suffer from having a secure authentication.

Some common authentication attacks techniques and common design errors are described below;

**Authentication Before Parsing and Canonicalization:** Authentication must be done after parsing and canonicalization, if not there is a change that web application might fail to require authentication for protected zones. An attacker could try path traversal attack techniques to bypass authentication for protected zones.

**Authentication Bypass by Alternate Name:** If a web application performs authentication based on the name of resources such as pages, but there are alternate names referring the same resource. Authentication mechanism might be bypassed by supplying alternate name of the resource such using different encoding for requesting the same web page. An attacker tries one or combination of equivalent encodings, canonicalization, multiple trailing slash, trailing space, mixed case, and other equivalence attack techniques described above.

**Authentication Bypass by Alternate Path:** If a web application has protected zones that requires authentication however it also has an alternate path or channel that does not require authentication. Malicious user could use alternate path to reach protected zones.

**Authentication Bypass by Assumed-Immutable Data:** If authentication mechanism of a web application depends on assumed-immutable data but that can be controlled or modified by the attacker such as cookies. An attacker could bypass authentication by setting certain cookies. Some observed examples are;

CAN-2002-1730, CAN-2002-1734 - Authentication bypass by setting certain cookies to "true".

CAN-2002-2064 - Admin access by setting a cookie.

CAN-2002-2054 - Attacker could gain privileges by setting cookie.

CAN-2004-1611 - Product trusts authentication information in cookie.

CAN-2005-1708 - Authentication bypass by setting admin-testing variable to true.

CAN-2005-1787 - Attacker could bypass authentication and gain privileges by setting a variable.

**Replay Attack:** Authentication mechanism of web applications can be easily broken by replay attack if it is possible for a malicious user to sniff network traffic and replay it the server giving same effect as the original message. By using replay attack, an attacker could login the system as owner of the captured message and has all privileges as him.

**Authentication Bypass by Spoofing:** If a web application does authentication depends on self-reported IP address, self-reported DNS name or referrer field in HTTP requests, it could be vulnerably to spoofing attacks. Malicious users can fake authentication information, claim any IP address, DNS cache could be vulnerably to cache poisoning so DNS names are easy to be spoofed and also the

referrer field in HTTP requests can be easily modified and, as such, is not a valid means of message integrity checking.

**Man-in-the-Middle Attack:** A web application's authentication mechanism is susceptible to man-in-the-middle attacks when it fails to adequately and consistently authenticate the identity of both ends of a communication channel. An attacker can place himself/herself in the middle of two communicating parties and impersonate each.

**Reflection Attack:** If a web application's authentication mechanism depends on shared secret key authentication and not properly designed. It could be vulnerably to reflection attack. An attacker could use reflection attack techniques and bypass the authentication.

**Account lockout attack:** In an account lockout attack, the attacker attempts to lockout all user accounts, typically by failing login more times than the threshold defined by the authentication system. For example, if users are locked out of their accounts after three failed login attempts, an attacker can lock out their account for them simply by failing login three times. This attack can result in a large scale denial of service attack if all user accounts are locked out.

Some common errors that can lead to authentication error so lowers the application security strength are using single-factor authentication; using password based authentication if passwords are not encrypted or non-reversible or if password aging is not considered or password strength is not enforced; having a missing step in authentication design; multiple failed authentication attempts are not prevented and no authentication for critical function.

## 1.25 Sniffing Application Traffic Attack

Sniffing application traffic simply means that the attacker is able to view network traffic and will try to steal credentials, confidential information, or other sensitive data. Anyone with physical access to the network is able to sniff the traffic. Also, anyone with access to intermediate routers, firewalls, proxies, servers, or other networking gear may be able to see the traffic as well. By sniffing application traffic, an attacker gain sensitive information about the web site. If this communication is not protected, the attacker can reveal user cookies, session id, user id and password that can be used to generate other attacks later.

## 1.26 Cross-Site Request Forgery (Session Riding)

Cross-Site Request Forgery is about forcing an unknowing user to execute unwanted actions on a web application in which he is currently authenticated. CSRF is an attack that tricks the victim into loading a page that contains a malicious request. It is malicious in the sense that it inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf, like change the

victim's e-mail address, home address, or password, or purchase something. CSRF attacks target functions that cause a state change on the server.

CSRF works like XSS attack: An attacker identifies a URL on a Website that initiates typical Web functions such as making a purchase, changing an email address or transferring funds and takes that URL and loads it to a web page he controls with malicious code injected to be executed later. The following example has an attack embedded in the img request below:

<img src="http://www.mybank.com/transferFunds.do?acctId=123456" />

The actual attack occurs when the user visits the attacker-controlled web page via a legit link, which forces the browser -- using legitimate, authenticated cookies -- to make malicious requests. In this example it will issue a request to www.mybank.com to the transferFunds.do page with the specified parameters. The browser will think the link is to get an image, even though it actually is a funds transfer function. For most sites, such a request will normally automatically include any credentials associated with the site, such as the user's session cookie, basic auth credentials, IP address, Windows domain credentials, etc. Therefore, if the user has authenticated to the site, the site will have no way to distinguish this from a legitimate user request.

In this way, the attacker can make the victim perform actions that they didn't intend to, such as logout, purchase item, change account information, or any other function provided by the vulnerable website.

## 1.27 Session Fixation Attack

Session fixation attack is one of the most complex attack techniques that must combine cross-site scripting or DNS cache poisoning or network based attack techniques in order to succeed. But the impacts of session fixation is extremely dangerous, with a successful attack, the attacker would login in to the system as victim and gain all privileges of the victim. In session fixation attack, the attacker tries to fixes the user’s session ID before the user logs into the target server so that he can then generate malicious requests with that fixed session ID. Although the session fixation attack techniques are complex, the reason of vulnerability for web application authenticating a user without first invalidating the existing session, thereby continuing to use the session already associated with the user. For example, J2EE web application where the application authenticates users with *LoginContext.login()* without first calling *HttpSession.invalidate()* makes whole application to be vulnerably to session fixation attack.

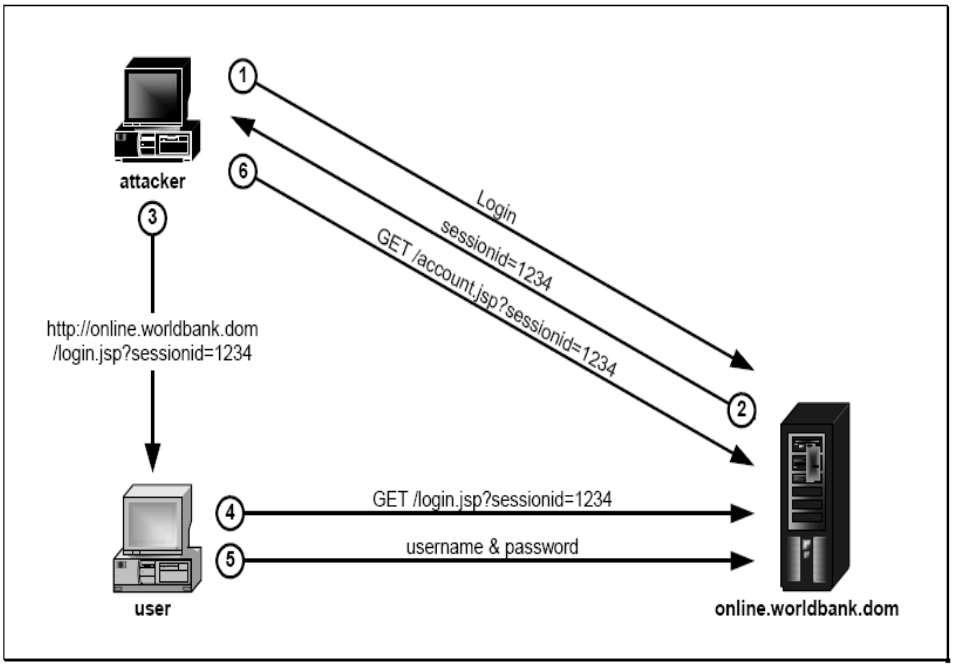


**Figure 39 Session Fixation Attack**

Session fixation attack has three phases; session setup phase, session fixation phase and session entrance phase.

**Session Setup Phase:** Firstly, the attacker either sets up trap session on the target server and obtains that session's ID or selects an arbitrary session ID to be used in the attack. The attack technique depends on the session management mechanism on web servers and can be classified into two categories; *permissive* those that accept arbitrary session IDs, *strict* those that only accept known session IDs, which have been locally generated previously. For a permissive mechanism, the attacker only needs to make up a random trap session ID and store it to use at session fixation phase. For strict mechanism the session setup phase becomes more complicated. Now the attacker will have to actually establish a trap session with the target server possibly from different account, extract the trap session ID from response and store it to user at next phase. If a time out mechanism is set for user sessions, the attacker also needs to keep alive the session by sending arbitrary requests periodically to web server.

**Session Fixation Phase:** Next, the attacker needs to introduce trap session ID to the user's browser to fix victim session. The attack technique used in this phase is chosen according to session ID transport mechanism of a web application;

- If the web application depends on session IDs stored in an URL argument, the attacker needs to trick the victim into logging in to the target web server through the malicious link including the trap session ID obtained from the previous step provided. This malicious link can be at attacker own web site or can be send to the victim by email. While it is the only method that can be done, it is quite impractical and risky for detection.

- If the web application store session IDs in a hidden form, the attacker needs to trick the victim into logging in to the target web server through a look-like login form that comes from attacker web server. In order to do this, an attacker must exploit a cross-site scripting vulnerability with page-hijacking to construct a malicious login form with the trap session ID obtained from the previous step. However, if victim is affected by this kind of cross-site vulnerability, there is no need to continue session fixation attack since a malicious login form could just direct the user's login credentials to the attacker's web server.

- If the web application's session mechanism depends on cookies. There are various and more effective techniques to fix trap session ID to victim's browser. Some of these are exploiting cross-site script or meta tag injection for issuing a cookie, break into a host in domain and install a cookie-issuing web server, by DNS poisoning add a cookie-issuing server to the domain on user's DNS server, modify the response from any server to issue a cookie.

**Session Entrance Phase:** After the attacker successfully completed previous phases, all he has to do is wait for the victim to login to the system, then the attacker can enter the trap session and assume the user's identity. If web application does not dedicate users session to user login IP, there is no way to differentiate attacker's requests from victim requests so he can do any operation that the victim has been permitted.

## 1.28 Session Hijacking Attack

Using session hijacking attack, the attacker tries to take control of a user session by obtaining or generating an authentication session ID. Session hijacking involves an attacker using captured, brute forced or reverse-engineered session IDs to seize control of a legitimate user's session while that session is still in progress. In most applications, after successfully hijacking a session, the attacker gains complete access to all of the user's data, and is permitted to perform operations instead of the user whose session was hijacked.

There are several problems with session ID's. If encryption is not used (typically SSL), Session IDs are transmitted in the clear and are susceptible to eavesdropping.

There are three primary techniques for hijacking sessions;

**Brute force attack:** The attacker tries multiple IDs until successful. There are lots of brute force attack tools, that generates HTTP requests with possible session ID's. If the attacker finds a valid ID, he could continue using the vulnerably site as if he were a valid user.

**Reverse Engineering:** In many cases, IDs are generated in a non-random manner and can be calculated. Many of the popular websites use algorithms based on easily predictable variables, such as time or IP address, in order to generate the Session IDs, causing their session IDs to be predictable.

**Stealing:** - Stealing session ID's from valid users is the last but effective case that can be used Using different types of techniques like sniffing network traffic, using trojans on client PCs, using the HTTP referrer header where the ID is stored in the query string parameters, and using cross-site scripting attacks, the attacker can acquire the Session ID.

In a "referrer" attack, the attacker entices a user to click on a link to another site (a hostile link, say www.hostile.com):

GET /index.html HTTP/1.0

Host: www.attackersite.com

Referrer: www.targetside.com/viewmsg.asp?msgid=438933&SID=2343X32VA92

The browser sends the referrer URL containing the session ID to the attacker's site - www.hostile.com, and the attacker now has the session ID of the user.

Session IDs can also be stolen using script injections, such as cross-site scripting. The user executes a malicious script that redirects the private user's information to the attacker. Sniffing network traffic can be used if the transportation of session ID's is done on open channel that can be eavesdropping, Inserting Trojan on victim PC's can steal cookies and send them back to the attacker site.

# APPENDIX B

## FULL LIST OF COMMON WEB APPLICATION ATTACKS

Below, common web application attacks are given in a table. The attacks are categorized in **Source Taxonomies** column depending on PLOVER taxonomy and also OWASP Top Ten Most Critical Web Application Security Vulnerabilities. As stated in Security Incidents section, all vulnerabilities that are explained in this section are categorized under Location->Code->Source Code node. **Parent Category** column is used to combine several related attack into a parent category, for example relative path traversal and absolute path traversal attack have similar character and combined under path traversal attacks category. So the real category of an attack that has a parent category is combination of source taxonomy and parent category. So relative path traversal is in fact belongs to Data Validation->Input Validation->Pathname traversal and equivalence errors->Path Traversal Attacks. **Causal Nature** describes if an attack depends other attacks to be occur. Likelihood of exploit shows the rank of attack likeness. **Impacts** column describes the consequence of an attack category and can be disclosure of information, unauthorized modification, unauthorized access and disruption of service. This distinction is depended on National Vulnerabilities Database web site.

**Table 19 Full List of Common Web Application Attacks**

| Web Application Attack | Source Taxonomies | Parent Category | Causal Nature | Likelihood of Exploit | Impacts |
|---|---|---|---|---|---|
| Relative Path Traversal | Data Validation->Input Validation->Pathname Traversal and Equivalence Errors<br>A2 – Broken Access Control | Path Traversal Attacks | Independent | Very High | Disclosure of Information |
| Absolute Path Traversal | Data Validation->Input Validation->Pathname Traversal and Equivalence Errors<br>A2 – Broken Access Control | Path Traversal Attacks | Independent | Very High | Disclosure of Information |
| Path Equivalence Attacks | Data Validation->Input Validation->Pathname Traversal and Equivalence Errors<br>A2 – Broken Access Control | | Independent | Very High | Disclosure of Information |
| Path Manipulation Attack | Data Validation->Input Validation->Pathname Traversal and Equivalence Errors<br>A2 – Broken Access Control | | Independent | Very High | Disclosure of Information |
| Special Element Injection | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| Command Injection | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | | Independent | Rare | Unauthorized Access |
| Argument Injection or Modification | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | | Independent | Rare | Unauthorized Modification<br>Unauthorized Access |
| Resource Injection | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | | Independent | Medium | Unauthorized Access |
| Direct Dynamic Code Evaluation | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | Code Injection | Independent | Medium | Disclosure of Information<br>Unauthorized Modification |

**Table 19 (continued)**

| | | | | | |
|---|---|---|---|---|---|
| Direct Static Code Injection | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | Code Injection | Independent | Medium | Disclosure of Information<br>Unauthorized Modification |
| PHP File Inclusion Attack | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | Code Injection | Independent | High | Disclosure of Information<br>Unauthorized Modification |
| LDAP Injection | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | | Independent | Very Few | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| SQL Injection | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| Basic XSS | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-4 XSS Flaws | Cross Site Scripting (XSS) Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| XSS in Error Pages | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-4 XSS Flaws | Cross Site Scripting (XSS) Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| Script in IMG Tags | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-4 XSS Flaws | Cross Site Scripting (XSS) Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| XSS Using Script in Attributes | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-4 XSS Flaws | Cross Site Scripting (XSS) Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| XSS using script via encoded URI schemes | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-4 XSS Flaws | Cross Site Scripting (XSS) Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| Doubled character XSS manipulations | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-4 XSS Flaws | Cross Site Scripting (XSS) Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |

**Table 19 (continued)**

| | | | | | |
|---|---|---|---|---|---|
| Invalid characters in identifiers | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-4 XSS Flaws | Cross Site Scripting (XSS) Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| Alternate XSS syntax | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-4 XSS Flaws | Cross Site Scripting (XSS) Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| XML Injection | Data Validation->Input Validation->Injection<br>A1- Invalidated Input, A-6 Injection | | Missing XML Validation | Medium | Unauthorized Modification |
| Missing XML Validation | Data Validation->Input Validation<br>A1- Invalidated Input | | Independent | Medium | Unauthorized Modification |
| Cross-User Defacement | Data Validation->Input Validation->HTTP Response Splitting<br>A1- Invalidated Input, A-6 Injection | HTTP Response Splitting | XSS Attacks | High | Disclosure of Information<br>Unauthorized Access |
| Cache Poisoning | Data Validation->Input Validation->HTTP Response Splitting<br>A1- Invalidated Input, A-6 Injection | HTTP Response Splitting | XSS Attacks | High | Unauthorized Access |
| Page Hijacking | Data Validation->Input Validation->HTTP Response Splitting<br>A1- Invalidated Input, A-6 Injection | HTTP Response Splitting | XSS Attacks | High | Unauthorized Access |
| Process Control | Data Validation->Input Validation<br>A1- Invalidated Input | | Independent | Rare | Unauthorized Modification<br>Disruption of Service |
| Log Forging | Data Validation->Output Validation<br>A1- Invalidated Input, A-6 Injection | | Independent | Medium | Disruption of Service |
| Buffer and Numeric Errors | Data Validation->Range Errors & Numeric Errors<br>A1 – Invalidated Input, A5 Buffer Overflows | | Independent | Rare | Disclosure of Information<br>Disruption of Service |
| Encoding Errors | Data Validation->Representation Errors<br>A2 – Broken Access Control | Cleansing, Canonicalization and Comparison Errors | Independent | Medium | Unauthorized Access |

**Table 19 (continued)**

| | | | | | |
|---|---|---|---|---|---|
| Case Sensitivity Errors | Data Validation->Representation Errors<br>A2 – Broken Access Control | Cleansing, Canonicalization and Comparison Errors | Independent | High | Unauthorized Access |
| Early Validation Errors | Data Validation->Representation Errors<br>A2 – Broken Access Control | Cleansing, Canonicalization and Comparison Errors | Independent | High | Unauthorized Access |
| Collapse of Data into Unsafe Value | Data Validation->Representation Errors<br>A2 – Broken Access Control | Cleansing, Canonicalization and Comparison Errors | Independent | Rare | Unauthorized Access |
| Partial Comparison Errors | Data Validation->Representation Errors<br>A2 – Broken Access Control | Cleansing, Canonicalization and Comparison Errors | Independent | High | Unauthorized Access |
| Information Leak through Error Messages | Data Validation->Information Management Errors<br>A7 – Improper Error Handling | Information Leak | Independent | High | Disclosure of Information |
| Information Leak through Sent Data | Data Validation->Information Management Errors<br>A8 – Insecure Storage | Information Leak | Independent | High | Disclosure of Information |
| Information Leak through File and Directory | Data Validation->Information Management Errors<br>A8 – Insecure Storage | Information Leak | Path Traversal Attacks | Very High | Disclosure of Information |
| Information Leak through Data Queries | Data Validation->Information Management Errors<br>A8 – Insecure Storage | Information Leak | Independent | Medium | Disclosure of Information |
| Information Leak through Debug Information | Data Validation->Information Management Errors<br>A7 – Improper Error Handling | Information Leak | Independent | High | Disclosure of Information |

**Table 19 (continued)**

| | | | | | |
|---|---|---|---|---|---|
| Information Leak through Caching | Data Validation->Information Management Errors<br>A8 – Insecure Storage | Information Leak | Independent | Medium | Disclosure of Information |
| Truncation of Security-Relevant Information | Data Validation->Information Management Errors<br>A7 – Improper Error Handling | Information Loss or Omission | Independent | Medium | Disruption of Service |
| Omission of Security-Relevant Information | Data Validation->Information Management Errors<br>A7 – Improper Error Handling | Information Loss or Omission | Independent | Medium | Disruption of Service |
| Obscured Security-Relevant Information by Alternate Name | Data Validation->Information Management Errors<br>A7 – Improper Error Handling | Information Loss or Omission | Independent | Medium | Disruption of Service |
| Credentials Management Errors | Security Features<br>A3 – Broken Authentication and Session Management,<br>A8 – Insecure Storage | | Independent | Very High | Disclosure of Information<br>Unauthorized Access |
| Permission, Privilege, and Access Control Errors | Security Features<br>A2 – Broken Access Control, A3 – Broken Authentication and Session Management | | Independent Injection Attacks | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| Authentication Before Parsing and Canonicalization | Security Features->Authentication Attacks<br>A3 – Broken Authentication and Session Management | Authentication Attacks | Cleansing, Canonicalization and Comparison Errors | Very High | Disclosure of Information<br>Unauthorized Access |
| Authentication Bypass by Alternate Name | Security Features->Authentication Attacks<br>A3 – Broken Authentication and Session Management | Authentication Attacks | Special Element Injection | Very High | Disclosure of Information<br>Unauthorized Access |

**Table 19 (continued)**

| | | | | | |
|---|---|---|---|---|---|
| Authentication Bypass by Alternate Path | Security Features->Authentication Attacks<br>A3 – Broken Authentication and Session Management | Authentication Attacks | Special Element Injection | Very High | Disclosure of Information<br>Unauthorized Access |
| Authentication Bypass by Assumed-Immutable Data | Security Features->Authentication Attacks<br>A3 – Broken Authentication and Session Management | Authentication Attacks | XSS Attack, All Cookie based attacks | Very High | Disclosure of Information<br>Unauthorized Access |
| Replay Attack | Security Features->Authentication Attacks<br>A3 – Broken Authentication and Session Management | Authentication Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Access |
| Authentication Bypass by Spoofing | Security Features->Authentication Attacks<br>A3 – Broken Authentication and Session Management | Authentication Attacks | Independent | Very High | Disclosure of Information<br>Unauthorized Access |
| Man-in-the-Middle Attack | Security Features->Authentication Attacks<br>A3 – Broken Authentication and Session Management | Authentication Attacks | Independent | Medium | Disclosure of Information<br>Unauthorized Access |
| Reflection Attack | Security Features->Authentication Attacks<br>A3 – Broken Authentication and Session Management | Authentication Attacks | Independent | Medium | Disclosure of Information<br>Unauthorized Access |
| Account lockout attack | Security Features->Authentication Attacks<br>A3 – Broken Authentication and Session Management,<br>A9 – Denial of Service | Authentication Attacks | Independent | Very High | Disruption of Service |
| Sniffing Application Traffic Attack | A8 – Insecure Storage | | Independent | Medium | Disclosure of Information |
| Cross-Site Request Forgery (Session Riding) | Time and State<br>A3 – Broken Authentication and Session Management | | XSS Attack, All Cookie based attacks | Very High | Unauthorized Modification<br>Unauthorized Access |
| Session Fixation Attack | Time and State<br>A3 – Broken Authentication and Session Management | | XSS Attack, All Cookie based attacks | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |
| Session Hijacking Attack | Time and State<br>A3 – Broken Authentication and Session Management | | Sniffing Application Traffic Attack | Very High | Disclosure of Information<br>Unauthorized Modification<br>Unauthorized Access |