Automated Generation of Application Models Through Access Traffic Analysis

By

Syed MishalMurtaza

2007-NUST-MS-PhD-IT-34

Supervisor

Dr. Hafiz Farooq Ahmad

A thesis submitted in partial fulfilment of the requirements for the degree of Masters in Information Technology (MSIT)

In

School of Electrical Engineering and Computer Science (SEECS)

National University of Sciences and Technology (NUST), Islamabad, Pakistan

APPROVAL

It is certified that the contents and form of thesis entitled “Automated Generation of Application Models Through Access Traffic Analysis” submitted by Syed MishalMurtaza has been found satisfactory for the requirement of the degree.

Advisor: _Dr. Hafiz Farooq Ahmad_____

Signature: _____

Date: _____

Committee Member 1: _Dr. Khalid Latif___

Signature_____

Date:_____

Committee Member 2: _ Ms. Sana Khalique

Signature _____

Date: _____

Committee Member 3: _ Mr. QasimRajpoot

Signature _____

Date: _____

IN THE NAME OF ALMIGHTY ALLAH

THE MOST BENEFICENT AND THE MOST MERCIFUL

TO MY PARENTS & SISTERS

CERTIFICATE OF ORIGINALITY

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: Syed MishalMurtaza

Signature: _____

ACKNOWLEDGEMENTS

First of all I am thankful to Almighty Allah for giving me courage and strength to complete this challenging task and to compete with international research community. I am also grateful to my family, especially my parents who have supported and encouraged me through their prayers that have always been with me.

I am extremely thankful to Dr. H. Farooq Ahmad, Dr Khalid Latif, Mr. QasimRajpoot and Ms. Sana Khalique for their help and guidance. I am thankful to Dr. H. Farooq Ahmad and Dr. Khalid Latif for his valuable suggestions and continuous guidance throughout my course work and research work. There foresightedness and critical analysis of things taught me a lot about research which will be more helpful to me in my practical life.

I am highly thankful to all of my teachers who have been guiding me throughout my course work and have contributed to my knowledge. Their knowledge, guidance and training helped me a lot to carry out this research work. I would like to offer my gratitude to all the members of the research group and my close colleagues who have been encouraging me throughout my research work especially Mr. Ali Hur, Mr. Zaffar Gachhal ,Mr. Rana Faisal Munair, Mr. Malik Nabeel Ahmad, Mr. Mahmood Ahmad. In the end I am thankful to my Mother and Father for their unconditional love, patience, and continuous guidance.

Syed MishalMurtaza

Table of Contents

Table of Contents

List of Figures

Table of Contents

Abstract

Web applications security has become critically vibrant. Traditionally the "default allow" model has been used for securing web applications, but this approach has exposed web applications to a plethora of attacks. Default deny model, on the other hand provides more restricted security to the web applications. This approach depends on building a model for the application and then allowing only those requests that comply with model and ignoring everything else. An innovative and effective methodology being adopted which lead to the analysis of valid application requests and as a result semi-structured XML cases for the web application being generated. Moreover, learning techniques are being adopted resulting to more mature and strong generated XML cases. This positive security model namely Web Application Model Generator (WAMG) consists of three components namely 1.Automatic white list cases generation Module, 2. Resource Tree Generator and 3.Case Based Reasoning. AMG needs to be described using a standardized XML language. The format should be able to describe all the three components of the positive security model accurately. We build this model through analysis of valid traffic logs in offline mode. The model is represented in the form of XML based cases. This system will be evaluated on the basis of fact that the XML file containing cases is being generating correctly according to the XML format. Moreover, it is ensured that splitting of malicious and non malicious traffic is carried out successfully. Results prove its effectiveness of rule generation using access traffic log of cross site scripting (XSS), SQL injection, JS Charcode, HTTP Request Splitting, HTTP response splitting and Buffer overflow attacks.

Chapter 1: Introduction

Web Applications security has become progressively more important these days. Enormous numbers of attacks are being deployed on the web application layer. Due to dramatic increase in Web applications, security gets vulnerable to variety of threats. Most of these attacks are targeted towards the web application layer and network firewall alone cannot prevent these kinds of attacks. The basic reason behind success of these attacks is the ignorance of application developers while writing the web applications and the vulnerabilities in the existing technologies. There are different technologies from various vendors for implementing same standards, e.g. Common Gateway Interface (CGI) is the standard mechanism for specifying the work of dynamic web application. Different technologies like ASP and ASP.NET, JSP and PHP to name a few exist for implementing the same technology in different ways and hence results in increasing complexities entailing in added security concerns. Figure 1 shows various technologies with respect to vulnerabilities found in their implementations.

The rapid development in Web 2.0 and evolution of social networks became centric to the hackers. Considering above fact, web applications are the most vulnerable. 75% of attacks are being deployed on web application layer [1, 2, 3, 4]. 81% of these attacks are targeted on payment card industry. The organizations which uses shared and default credentials give 51 % of the data to the hackers [5]. According to site security monitor that in every 90 breaches there are 285 million records exposed and that is greater the 230 millions exposed records in previous 5 years [6]. 30% of each 57 attacks are carried out using SQL injection attack [5] that's why the web application security is the most important these.

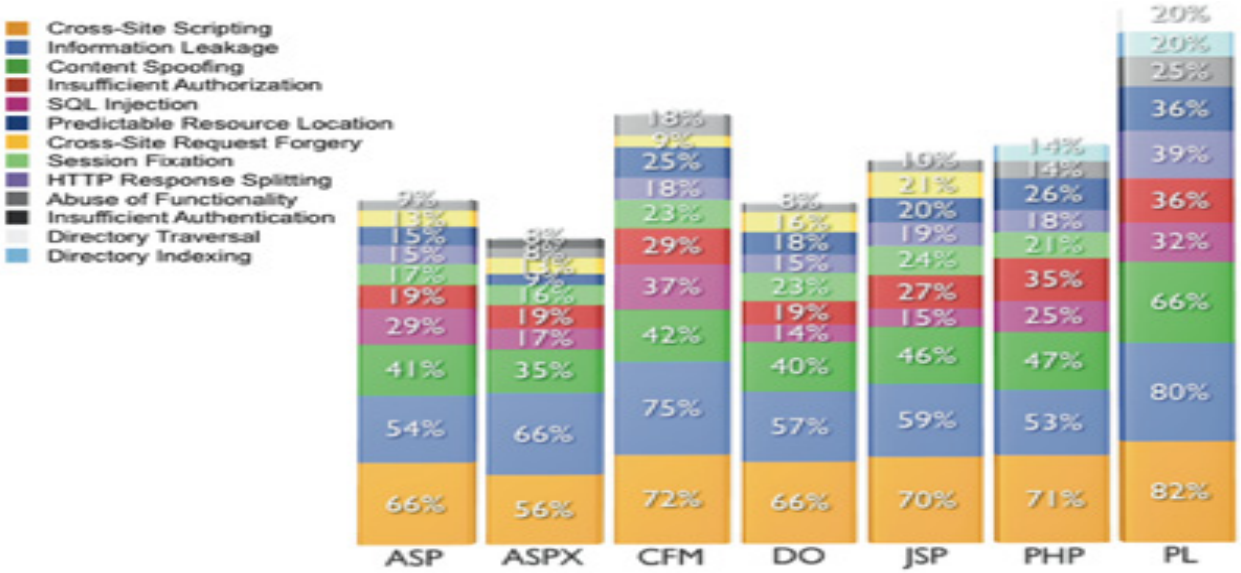


Fig [1]: Technology Breakdown with respect to attacks

The underlying layers like network layer are mature enough to survive attacks on that layer due to extensive research upon security of network layer. Web application layer is most vulnerable to these attacks unlike the network layer. Fig 2 shows the architecture of traditional network firewalls. Such kind of network firewall cannot stop the application layer attacks because of some conventional problems with web application layers as given in the following:

- Web applications are so dynamic that traditional network level firewalls using black listing approach are not able to detect these kinds of attacks.
- Web applications always require custom tuning.
- It does not protect Port 80 and 443[9].

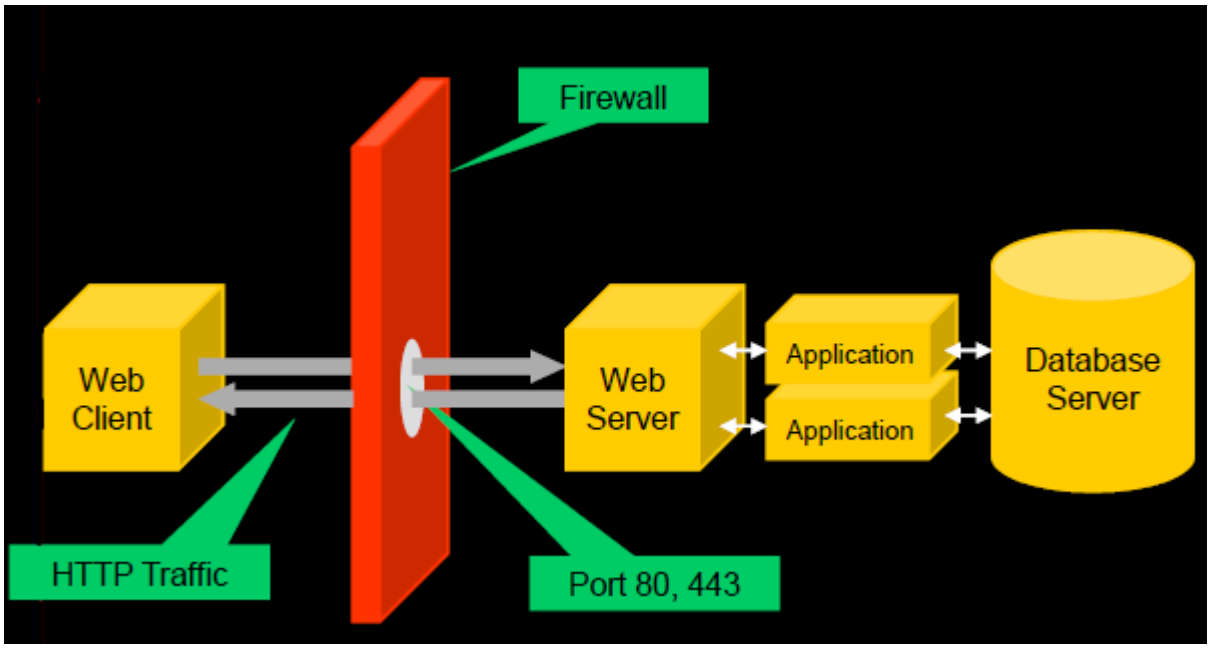


Fig [2]: Traditional Web Application Firewall [10]

Due to the problems mentioned above, the Web Application Firewalls (WAFs) are used to protect the web application layer. WAFs prevent web application form different classes of attacks like XSS, SQL Injection, and Directory Traversal etc. Traditionally WAFs use two different approaches to prevent from web attacks i.e.

1. Default allow model (Black List)
2. Default Deny model(White List)

1.1. Default allow model.

Defaults allow model is also called the black listing approach. It is widely used by most of the firewalls [25,26]. It allows all kinds of traffic but only stops the traffic which is detected by WAF. When the input(HTTP packet)is received at the security gateway, it is first compared to already created list of exploits or black list. If the input matches, it will be considered malicious

and consequently discarded. Traditionally the default allows model has been used for securing web applications. But this approach has exposed web applications to a numerous attacks like XSS,SQLI, Input validation attacks etc.

1.2. Default Deny Model

Default denies model is also known as the White List. White List approach denies all kind of traffic. It only allows the benign traffic. It maintains a list of all the parameters in the web application and allows the traffic according to input type of the parameter by disallowing rest of the traffic[27]. This will reduce the chances of Zero day attack and many other attacks such as Buffer Overflow, forceful browsing and Injection flows to mention a few. Moreover, it prevents the attacks done using PUT and Delete methods of HTTP.

Black List is used widely out of these two techniques though it has some associated (conventional) problem as given in the following:

- Blacklist always needs to be updated with emergence of every new attack for it contains the signatures of attacks developed by analyzing attacks on the web application.
- Blacklist cannot detect any variation in attack whose signature is present in the black list. Therefore, it is ineffective against the Zero Day Attack.
- It can expand the surface of attack by manifold.
- It can accept any request except the request it can handle.
- Database grows with time and makes the system slow.

Intrusion Detection Systems (IDSs) are being categorized as misuse detection and anomaly detection systems, most of which follows negative security model like SNORT-IDS [7]. Because

of the issues mentioned above these systems do not prove good against the polymorphic variation of attack and Zero Day attack. A very good approach would be the manual creation of Blacklist. This technique will be very effective against Zero Day Attack and polymorphic variations of attack. Currently no data mining approach has proved effective for the generation of automatic blacklist creation [8].

Blacklist is an inefficient technique for the detection of attacks on web application. Here we need an appropriate validation technique for the web applications that can make profile of these web applications and check the input types of the parameters. This allows only valid inputs. The purpose will be served by Default Deny Model (White List).

- It can generate profile of the Web application and generates policy through learning. This is a good technique for it performs proper validation of the parameters and is effective against unwanted traffic.
- It only allows the traffic detected by rules and rejects everything else. That's why it is effective against the Zero Day attack and new variation of attacks.
- It can prevent form the forceful browsing attacks, injection attacks like SQL injection, buffer over flow attacks and many more.
- It decreases the surface of attacks.
- It eliminates the attacks caused by error(s) in web server.

White List proved much effective against the web application attacks. Policy generation through learning is very efficient technique; therefore, we need a learning mechanism useful against the web attacks and can generate more mature and adaptive

1.3. Thesis organization

This thesis is ordered into eight different chapters. **Chapter 2** provides a background of the web application security, it gives an overview of learning techniques and why learning is important for White List. **Chapter 3** presents literature survey to have an idea about the existing limitations in White List, which can help us to propose an efficient and effective solution. **Chapter 4** highlights some facts and figure that demonstrates the increasing number of application level attacks. Furthermore, it emphasizes its criticality that demands a serious effort for an effective solution. **Chapter 5** presents the proposed solution and its system architecture. **Chapter 6** presents the proposed system design and implementation. **Chapter 7** demonstrates the system evaluation carried out to prove proposed system effectiveness over other existing systems. The system evaluation is mainly focused on the detection ability and overall system features. **Chapter 8** presents the conclusion and future work.

Chapter 2: Background

2.1. Introduction

75% of the total attacks are being deployed over the web application layer hence more vulnerable to the attacks as compare to network layer .Network firewall can block all the traffic but it leaves the traffic of port 443 and port 80 (ports for web application traffic).This is the reason, why web application layer is becoming vulnerable to attacks as network firewall leave the web application security over the application developers [9]. Developers have to write the safe code that is invulnerable to attacks. Most of the developers of the web applications do not know how to write safe code hence causing the chances of attacks to increase with overwhelming commencement of web applications. However, writing the safe code is not an easy task with some limitations in existing technologies e.g. there is a shortcoming of C that it is prone to Buffer Overflow attack or some components of web 2.0 like AJAX are given priorities and are scheduled before the security.

If we are not able to comply with the condition of working out the safe code then other possible solution is to deploy a WAF. It is mostly done for the web application security because these firewalls are developed specially for application layer attacks. Most of the WAFs have deficiencies because of using black listing techniques. So we need some mechanism that can stop the web application attacks effectively and efficiently. That's why we need White Listing for the security of Web Applications. Black listing is definitely not the trust worthy technique as it works only with the known attacks. So we

have to use the White Listing technique which helps in protecting form the Zero day attacks and many other attacks mentioned above in Chapter1.

2.2. Learning techniques.

White Listing is a very good approach for web application security but the problem with this technique is

When and how we can learn the rules?

How we can make this White Listing technique more effective?

How we can adapt the rule to create much better and more mature rules for the web application?

In answering these vital questions, we present learning techniques as a solution.

Learning will help us to:

- Detect the changes in the web application and adapt the rules according the web application.
- Learn from mistakes committed in past.
- Check the validity of inputs of each parameter.
- Differentiate between the free parameter and discrete parameter.

There are basically two types of systems.

1. Rule Based system
2. Case Based Reasoning systems.

2.2.1. Rule based systems

Rule bases system (RBS) are basically the If-then conditions and the rule based system contains set of rules which are conditions C1,C2,C3….Cn. In case of a new

problem, solution of the problem is checked according to the given conditions. Additionally RBS contains an inference engine in it. It works as a comparing module which holds the data in it, and then compares it in the working memory with the condition part of the rule and decides which rule to fire [10]. It also determines the best sequence of rules to fire. Its working will be more efficient as size of its knowledge base increases [10]. Rule based systems are very efficient and are widely used in most of the learning systems [10].

2.2.2. Case based reasoning systems.

Case-based reasoning (CBR) pertains to the concepts and methods that touches upon some of the fundamental subject including reasoning, knowledge representation, and learning from knowledge**.**CBR uses previous knowledge to solve the problem. It represents the human reasoning because when we have a new problem we refer to our previous experiences and then try to solve the problem on the basis of how previous problem was solved. Advantage of CBR is that we can learn from previous mistakes. With each new iteration of learning we have more mature rules for our system. It makes the knowledge acquisition task very easy and is faster than other conventional systems.

2.2.3. CBR VS Rule Based Systems.

CBR is opposite of the rule based system. Rule based systems are slower because it has the inference engine which works to compare and execute the conditions according to the rules. RBS needs a lot of time for knowledge acquisition, and it is adaptation will improve with a knowledge base having handsome number of rules.

This huge knowledge base will however increase the decision time which is very important aspect in a security application. On the other hand CBR can do reasoning with a limited amount of time. It uses an incremental approach for building Knowledge base. With the maintenance feature of CBR it eliminates the useless rules from the knowledge base and keeps knowledge base in a limit. It makes the knowledge acquisition task easy by using the previous cases and it can build the case base with less knowledge. In Security applications CBR is the best choice because it is faster than the rule based systems [11].

2.3. Advantages of CBR

2.3.1. Evade repeating mistakes made in the past.

It uses the incremental approach for the development of Knowledge base. It analyzes the previous and current cases and then updates the case base according to the pattern of inputs hence each and every time it corrects it.

2.3.2. Trim down knowledge acquisition.

It eliminates the need to extract a model or a set of rules, as is necessary in model/rule-based systems. The understanding of acquisition tasks of CBR consists primarily of the collection of appropriate experiences/cases, their demonstration and storage space.

2.3.3. Providing flexibility in comprehension modeling.

Other systems cannot often cater the problem which resides in the boundary of the solution or which cannot be determined by their rules. They need a very good understanding of the domain which leads to a huge knowledge base. In the case of

CBR it uses previous cases for domain knowledge which will provide reasonable adaptation of the new problem.

2.3.4. Analysis in domains that have not been fully implicit, distinct, or modeled.

In those situation where you have a small knowledge of domain CBR will also work in that domain with a small amount of knowledge.

2.3.5. Over the time learning.

It uses the incremental approach for learning of knowledge. As it caters more problems it also increases the knowledge base as well.

2.3.6. Reasoning in a sphere with a small knowledge.

At the start, the case base reasoner works on few cases of problem domain and then gradually builds its knowledge base with the increase of more cases. The accumulation of new cases will cause the system to enlarge in strategy that is determined by the cases encountered in its problem-solving activities.

2.3.7. Broaden the range of domains.

CBR can be extended to a broad range of domains. It is easy to implement and it can be represented in a limitless number of implementation in indexing and adaptation of new cases.

2.3.8. Shimmering human reasoning

CBR represents human reasoning as humans encounter a problem they refers to previous cases and then try to solve the problem so CBR also has previous cases referred to solve a new problem and to adapt the cases.

2.4. Architecture of CBR

Fig 4 we have the architecture of CBR

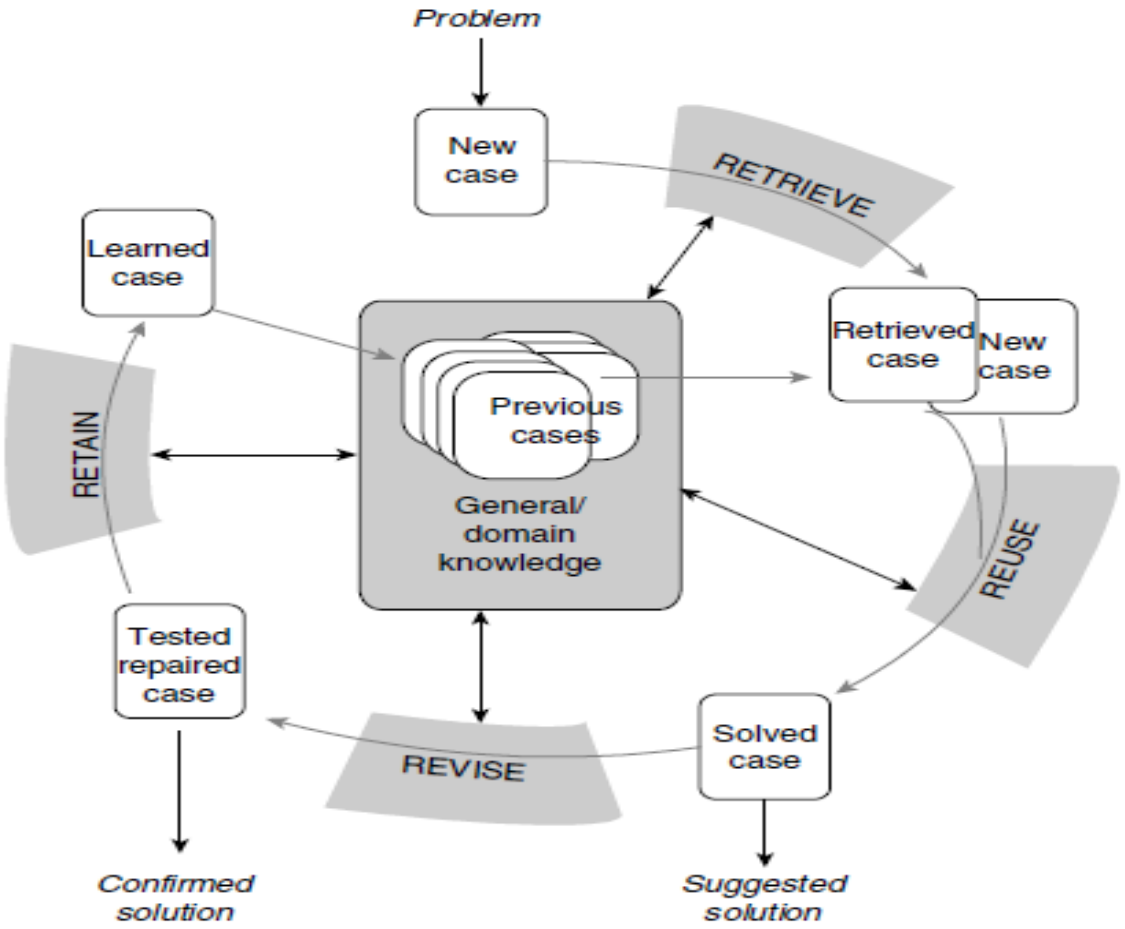


Fig 2.1.Architecture of CBR [11].

CBR have four phases

- **Retrieve**

  CASE retrieval is the process of finding the cases which are similar to the new problems which are very near to the problem. Fig [2.2] shows the Case Retrieval process of CBR.

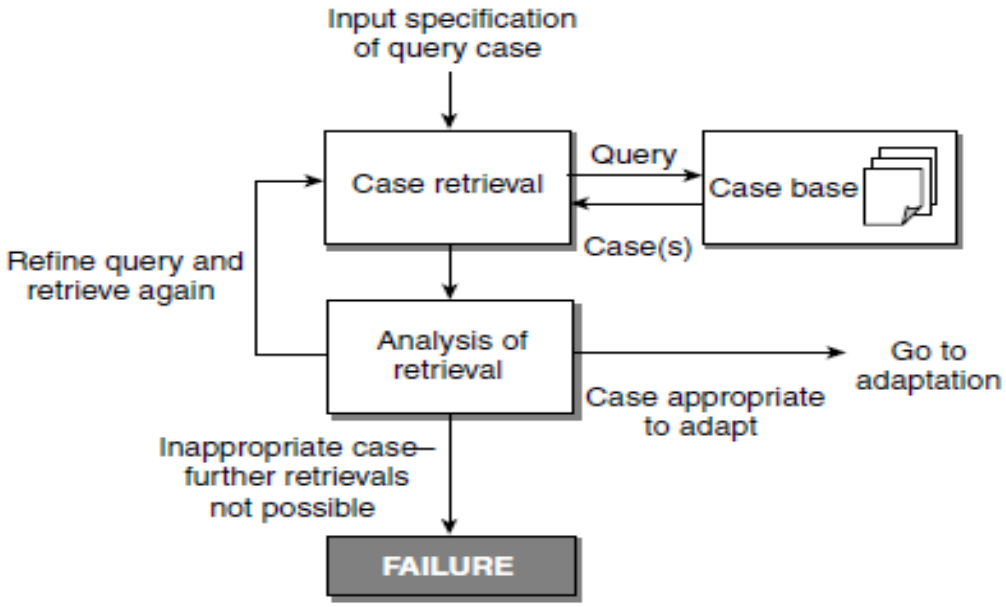


Fig 2.2 : Case retrieval process [11].

Whenever a new case comes to the case base, it will send a query to the case base and retrieve similar cases form the case base. The cases will be analyzed to find the similar cases form the case base [11].

- **Reuse**

  After the analysis of the previous cases, if CBR finds cases which is similar to the previous cases then it will use the pattern and solve the new case.

- **Revise**

  If the solution is not according to the problem then the process of revision will start. It also called the process of adaptation. It gives solution of the new problem and makes a new case out of it. Figure 2.3 show the process of revising the solution.

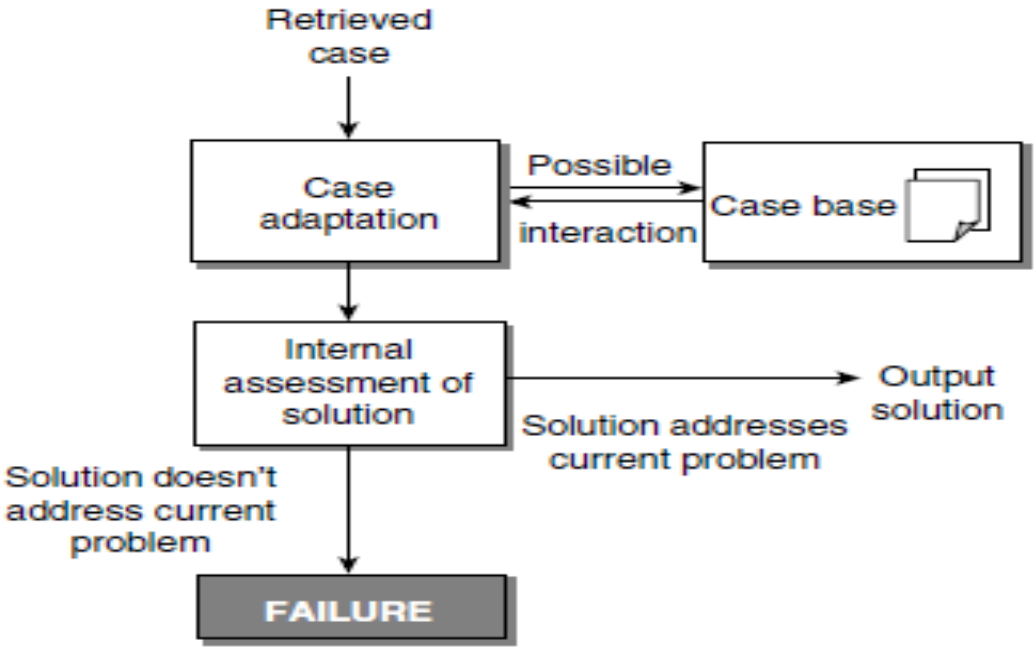


Fig 2.3. Case based adaptation [11].

- **Retain**

  The larger the case library, the greater the problem space covered. However, this would also downgrade system performance if the number of cases were to grow unacceptably high. Case based systems remove redundant or less useful cases to attain an acceptable error level is one of the most important tasks in maintaining CBR systems.

2.5. Summary

For the protection of web application we need some strong mechanism for the detection of web application attacks and also to avoid Zero Day Attacks. White Listing is most strong technique for the prevention of such attacks. Profile generation is the most powerful technique for White Listing and for this we need some learning technique, capable of learning for the log of the web application and generate a model of the web application.

Chapter 3: Literature Survey

3.1. Conventional input validation techniques

As we saw in the preceding section that the core foundation of various application layer attacks is due to the incapability of the web application to sieve malicious content being input. Fundamentally to filter or validate input we need to employee validation list that is the most widely used technique. This contains the list of either legitimate or illegitimate expected inputs. The input when arrives is first checked against this already built list and then marked as legitimate or illicit.

White List is used for authentication of user inputs that are being deployed on several web applications. The White List contains all legitimate input values that are considered permissible for the web application. The user input is first compared to the entries in the white-list and if found in the list, it is considered as legitimate and is permitted otherwise it will be blocked.

Few of the observed problems in deploying White List are as below:

3.1.1. Updating and maintenance problem:

The White List is generally application specific and it requires to be updated when there is any modification in the web application to which they are coupled with. Everyday attackers are becoming more and more dynamic and using various new kinds of attack vectors. It often uses code injection attacks like Forceful browsing, Cross Site Scripting (XSS), and SQL Injection etc. According to OWASP these kinds of attacks are endlessly escalating

day by day despite of many counter measures. Attackers are using more robust attacking techniques and hence able to circumvent the already engaged filtering techniques. To protect our applications from these modern attacks, there is need of unremitting update and maintenance of the White List. This is again a complex task as it needs nonstop development endeavor that is unfeasible in most of the cases.

3.1.2. Time intense task:

Development of White List is also a time intensive task and more often overlooked. Generally developers work in extreme pressure and under strict time deadlines, so they tend to focus on functionality rather than the safekeeping of the application. In most of the cases they are usually not aware of the security aspects of the application and leave the loop holes for the attacker to be exploited. Developer of the application knows, how to built the application but unaware of security concerns of web applications. In commercial level development the developers know the security concerns because it is very important for business and hence they follow security measures and try to employ defense mechanism for the security of application. Development of these defense strategies is itself time intensive and more often results in conflicting deadlines. Usually developers examine carefully all the input values to the application and then embed validation schemes, like in the application source code. It is highly time consuming task considering big applications have several inputs and to consider each and each value to embed in White List itself needs lot of time. It requires understanding the permissible range of the values, format, size etc of that thorough knowledge of input.

3.1.3. Incompatibility with legacy applications:

The other discrepancy with this conventional White List approach is that it cannot be deployed for existing applications. What could be our approach to deal with the security of already built legacy applications? It would be very hard to maintain or develop white Lists for existing applications that are already functioning. It consumes a lot of time as well as to embed these lists into already functioning applications would also require modifications (to make compatible with the changes) in to the existing application. Similarly to protect already built and functioning legacy applications the developers have to face two problems i.e. in most of the cases the source code is copyrighted and can't be altered and the 2nd problem is that most of the cases the source code is shipped in binary forms e.g., in the form of dynamic link library (.dll) file or java .class. In such cases, to include White List or black list in the source code is impossible. Here conventional approach completely fails, and the only solution to secure these applications is the employment of application layer gateway that constitutes the validation mechanism capable of filtering these malicious contents.

3.1.4. Dealing with many sources of input:

The other difficulty with this White List approach is that, there are many sources of conveyance of input to the web applications, so in order to properly authenticate input we must consider each and every source of input and then the White List should be made exclusively according to these input sources. There are many sources of input for web

application the four broad categories of sources of input are user input, databases, 3rd party applications, other network and desktop applications with which web applications interact through defined interfaces. This is shown in the fig 3.1. This is also a time intense job for developers to validate and analyze all the upcoming inputs towards web application that take away their consideration from properly applying the functionality and further increases the ambiguity in their minds. Maintaining of White List is another big challenge. The updating of validation list with respect to specific web application is respectively easy to perform but if there is a change in those applications that lies outside your web applications then it is difficult to understand those modifications and it further complicates the task of updating validation list.

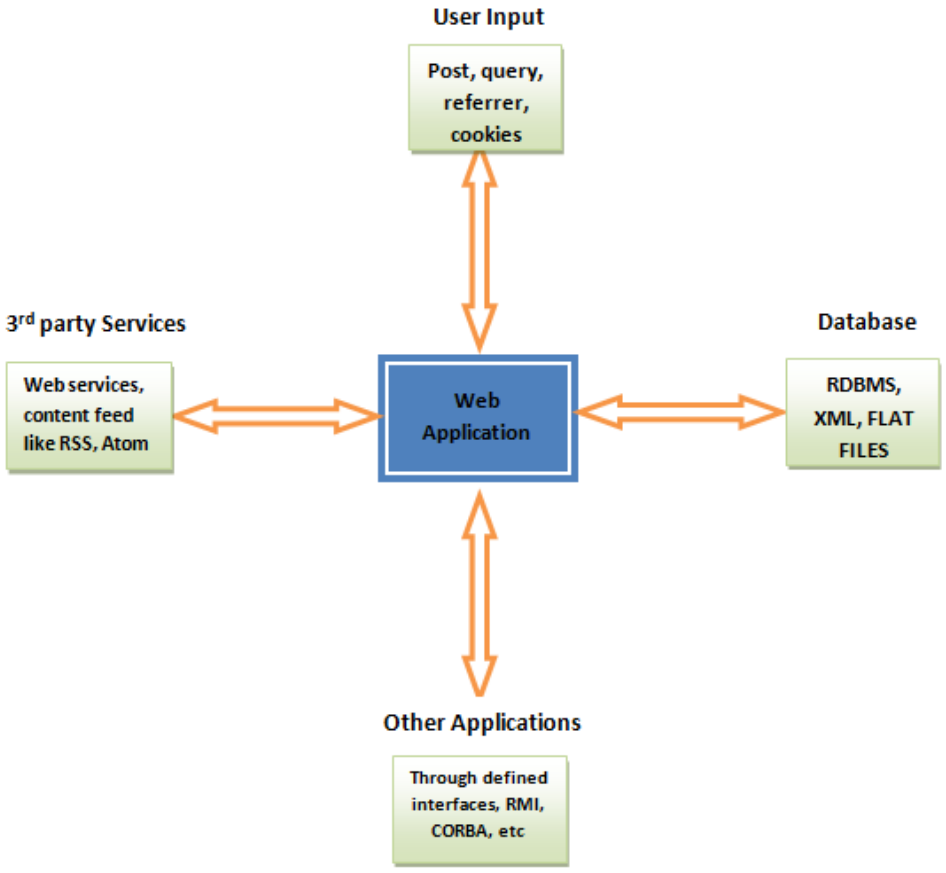


Figure 3.1: Input Sources to web application [12]

3.2. Review of papers

This section contains the review of some of the papers which can provide the overview of some problems of black lists, White Lists and dynamicity of the web applications.

3.2.1. Using generalization and characterization Techniques in the Anomaly-based Detection of Web Attacks [13].

This paper describes a novel approach to detect the malicious patterns in inputs of the web applications. The approach described in the paper uses an anomaly generalization

technique which translate malicious requests in to signatures then the similar anomalies are grouped together to identify similar kind of alerts in future. They have presented an anomaly generalization technique which can detect the malicious inputs and then also generalize them for the detection of similar attacks which can help in the detection of similar attacks. They are using probability techniques attribute length for the detection of buffer overflow attack, character distribution using character frequency analysis for the occurrence of characters in the http packets for the detection of buffer overflow attack, Directory traversal attack and SQl injection attacks. It uses token finder technique for the detection of malicious token in the http packet. It also uses a heuristic approach to infer the type of attack on the application to help the administrator to cater the false positives issues. The paper presents a novel technique for the detection of attacks on application layer but the problem is that it can detect only four types of attacks XSS, SQL injection, buffer overflow attack, and directory traversal attack .administrator plays a vital role for the detection of attack because it will group the similar anomalies and then administrator have to decide whether it is a false positive or a false negative and if administrator make a wrong decision then that particular type of anomalies are allowed in the system and this will make the system vulnerable to attacks. This system is not able to learn the behavior of normal traffic. At the same time, system is not fully automated because the administrator has to decide about the false positives and the false negatives.

3.2.2. Protecting a moving target: Addressing web application Concept Drift [14].

The paper focuses on the change management in the web applications. Web applications change with the passage of time and these changes cause attacks on the applications

because as every time logic changes in the application then number of bugs and loop holes also increases with the change and this help attackers to attack on the web application. The purposed technique in this paper will help to learn the changes in the application and then it automatically adapts the anomaly detection model according to the change for effective detection of attacks. It uses the technique of http response modeling and this helps to detect the parameter changes in the application and also it learn the new parameter in the application which helps to update the session and request model of the application. It models the request and response for the detection of various application level attacks like XSS, SQL injection and Directory traversal attack etc. It also models the session of the application to detect the CSRF, Session authentication attacks. The problem with this technique is that it can only detect the changes in the dynamic web application but for the static applications it cannot work. It also did not work in the case of java script if there is a change in the java script it cannot detect it. It also cannot detect the rich and media dependent response model these are the component that are installed at the client side like flash, adobe and Microsoft Silverlight application. If we can detect the changes in the application effectively then it would be very useful for the detection of web attacks because if one knows the model of the application you can identify the discrepancies in the application and with this information you can easily secure the application.

3.2.3. On the automated creation of understandable positive security models for web applications [8].

The paper presents a technique which is based on the creation of White List for the detection of web application attacks. According to the author network based firewalls

cannot protect against the web application attacks. To protect against the web attacks we have to define some techniques for the detection of web attacks. Although we can define black list, but black lists are ineffective against zero day attack and we have to make White List for the better detection of zero day attacks and other class of web attacks. Authors have given the concept of resource tree which define the hierarchal structure of the web application this will help to prevent the force full browsing attacks. Authors also have defined the parameter validity criteria. It also describes the validity of the parameter in terms of the input given by users. This technique uses XML for the faster access of data. This technique is very useful for the creation of White List and description of the parameter validity but it did not cater the change in the web applications because web application changes over time and with change of new parameters are introduced in the web application. So this technique will not work against the dynamic web sites, where thousands of users create new pages every day but this technique will not cater these changes. Another problem with this technique is that it did not create mature rules and also it did not check that the parameter is fake planted by an attack or not. It causes many attacks on the application in terms of parameter tempering.

3.3. Summary

This chapter has provided the literature survey that demonstrates various existing problems of White List. It also detailed the restrictions of these existing systems that make them ineffective for recent expanding application level attacks.

Chapter 4: Motivations

4.1. Research motivation

Web Applications security has become increasingly important. Traditionally the Default Allow Model has been used for securing web applications. But this approach has exposed web applications to a plethora of attacks. Default deny model, on the other hand provides fairly strong security to the web applications. This approach depends on building of a model for the application and then allowing only those requests that compiles the model and refusing every other request.

Black list approach has gained exponential growth in the knowledge base which makes the system slower and it is also a source of creating a large amount of false positives which result in the blocking of legitimate requests [15]. It is also a time consuming task and also needs hard work for the creation of black list signatures and it is ineffective against the polymorphic variation of various web attacks and also against Zero day attacks.

On the other hand White List is more secure and more accurate then the black list [15]. It also create generate less false positives then the black list [15]. White List is faster than the black list and it is very much effective against the zero day attacks and the polymorphic variation of web attacks.

There are various problems in existing systems as mentioned in chapter 3.it is hard to maintain and take too much time in its creation because in White List approach, we have to maintain each and every parameter of the web application. It itself is a hectic task for the administrator and with the change in the web application architecture we have to maintain

the White List. We need some mechanism to learn the changes in the web application and also check the validity of the parameter. It is also required to check the parameter in terms of input that, what the valid inputs for the parameter and it are learns the changes in the system and then adapt the new changes of the parameter by analyzing the previous and new information. It can learn from the mistakes done in the past. It has some mechanisms that eliminate invalid or useless rules for keeping the knowledge base up to date, useful and precise.

4.2. Research scope.

Application security needs to have main conventional technique which provides security to the application layer e.g. Black List and White List approach. White List has proven much better than the black List but it deals with many problems as mentioned above. Web Application security is the sub domain of the Application security which is further divided into White List and Black List as described in Fig 4.1. The focus of the thesis is to provide such a mechanism for White List that can learn the changes in the web application and can correct itself over the time and also address the conventional problems of the White Listing technique.

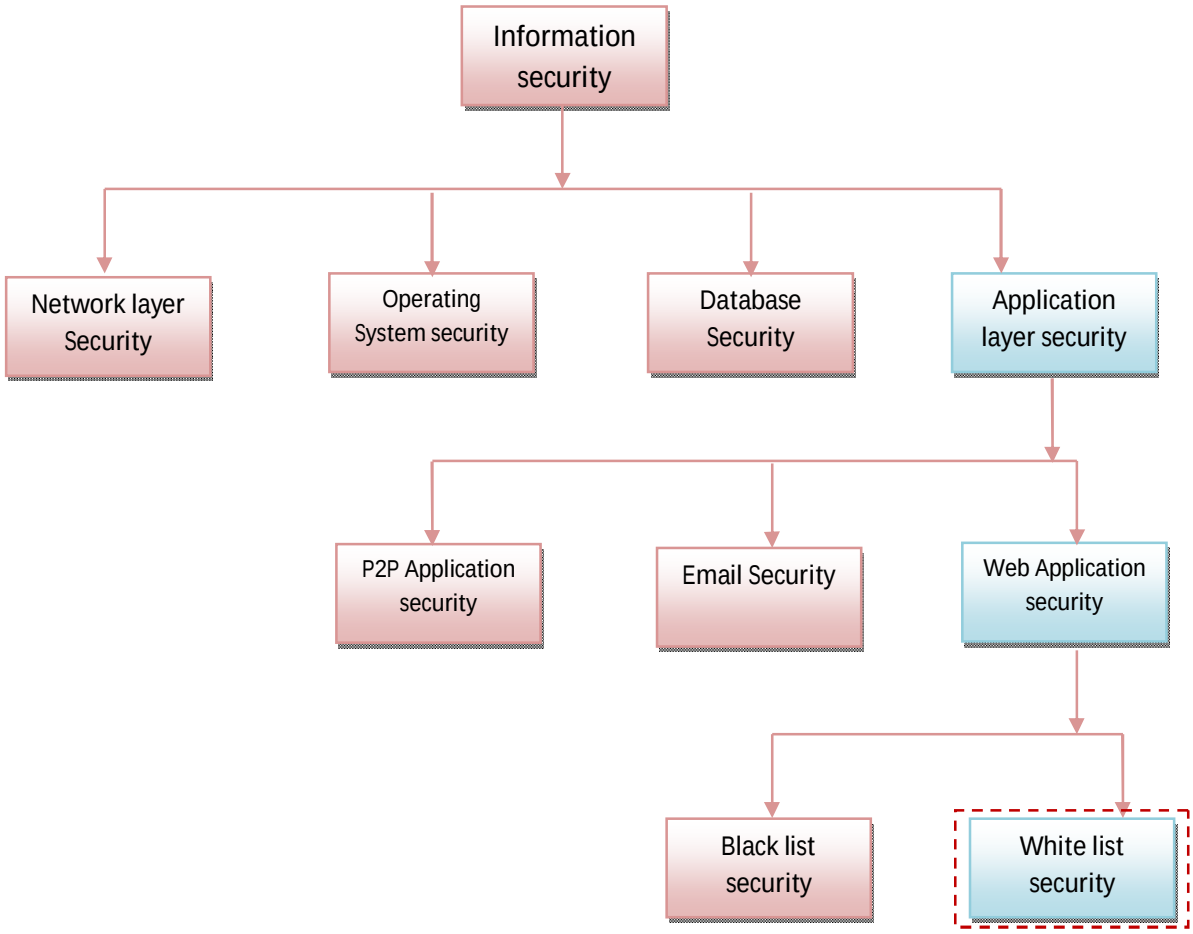


Figure 4.1: Research scope

4.3. Aims of purposed technique

In the light of the issues, we have explored in the criticism of existing solutions given rise to below explained potential solution.

Aims and objectives

- Study of the present techniques of generation of Positive Security Model (PSM) and their strengths and weaknesses.
- Development of a module for SWAF which analyses the valid application requests and generates XML cases for the application.
- Generation of more mature and stronger cases using learning technique.
- Develop such a mechanism that can make more mature rules and learn from the mistakes of the past and correct itself.
- It should be capable of learning and adapting the changes in the web application.

4.4. Summary

This chapter explains, how crucial the application level security is, and validity of White List security on the basis of different facts and surveys carried out by various authentic sources. Despite different existing White List security mechanisms, the rising figure of application level attacks has created the urge to investigate a valuable solution for application level attacks.

Chapter 5: System Architecture

5.1. Proposed solution

Considering all the issues asserted in chapter 3, architecture is designed to address those issues. The proposed system comprises of the features given in the following:

- System analyzes the log and consequently generates the profile of web application according to the Semi Structured XML format.
- System operates based on effective machine learning technique (namely CBR), which can generate mature White List rules.
- System is able to learn the changes in the web application.
- Generation of White List is an offline process that is anticipated to save run time processing.

5.2. Abstract architecture

Following are the three modular components of positive security:-

- Automatic White List *Cases Generation Module*
- Resource Tree
- Case Based Reasoning

This process of creating White List rules and learning through CBR has two iterations as given below

5.3. First time learning (1st iteration)

First iteration works to automatically generate SWAF White List Cases in Sami structured white list rules, whose detailed architecture is given in Figure 5.1.

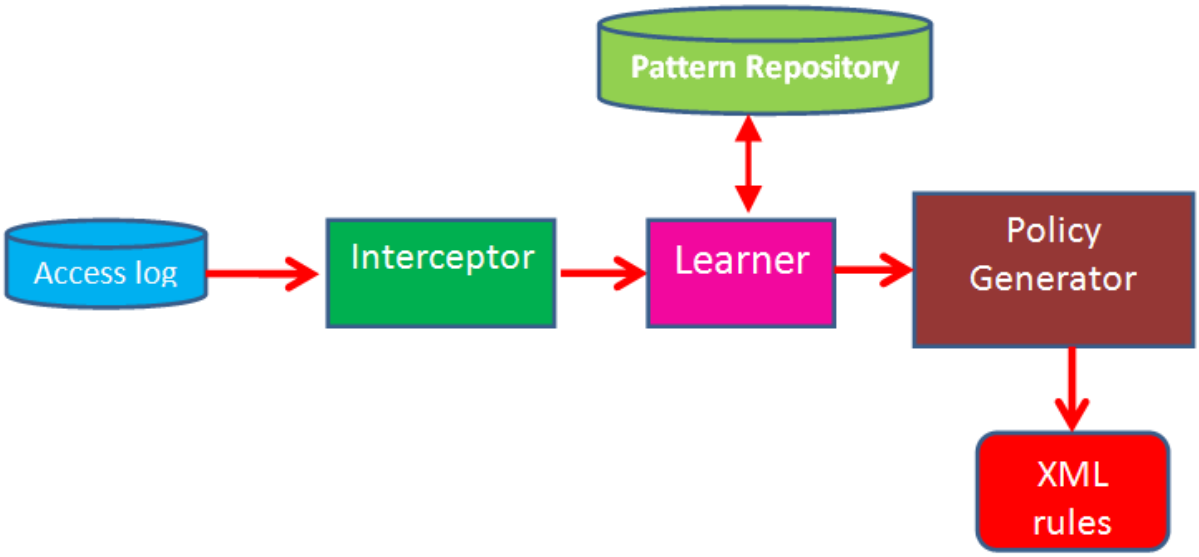


Figure 5.1: System Architecture

Each of the components illustrated above is given in the following.

5.3.1. Access log

There are two ways to collect the data from the web application

1. Web Crawler
2. Log Scanning

In these two techniques log scanning is the better technique because

- Web crawler cannot comprehend the dynamicity of the web application.
- Web crawler learning is not realistic as it is blind to the user inputs and can not access the restricted parts of the web application; log scanning leads us to realistic learning.
- We can observe the degree of web application's dynamicity with the help of log, as we can figure out different inputs given by user. By analyzing these inputs we can generate more mature White List rule.

Access log is the most important part of this module. It collects HTTP request and response, on the basis of this information, we decide which request is valid and which is invalid [12]. Moreover, it also helps us to see the request architecture and the response generated by the web application against the request. System collects all the data from the access log through log scanning. The request header checked for the generating White Listing rules are:

Date timestamp	Accept-Encoding
Hostname with port no	Accept-Charset
Client address with port no	Keep-Alive
Accept-Language	Http Referrer
Full Resource URL with parameter	Content-Length
Method name	Content-Type

Table 5.1: Access log components.

5.3.2. Interceptor

It segregates the valid and invalid requests by analyzing the response of request. Moreover, following potential operations are performed by this interceptor:

- On the basis of response, it checks the invalid request and then eliminates these requests.
- Checks which IP previously generated the attack and eliminates all the request form that IP.
- Formation of requests and then sends it to the learning module.

5.3.3. Learner

This module compares different parts of the request with patterns in Pattern Repository and picks the matching pattern (regex[i]).

- Learns cardinality and size of the parameters
- Checks the media type of the request.
- Checks the methods regex of the request.
- Sends the data to the case generator module.
- Min threshold (MT); it will see the minimum requests sufficient to create the case for specific resource; if the minimum threshold is met then it generates the case otherwise it does not.

- Max threshold (MXT): on achieving maximum threshold, the case confidence should be 100% and decision made by the case is 100% accurate requiring no extra defense mechanism.
- If an occurrence of a resource crosses MT then we create its case with the confidence level.

  Note: Confidence: is the parameter that helps us to determine that how strong the Case is.

5.3.4. Regex repository

The repository contains many regex and is completely configurable by administrator. We can also send the updated repository as an update.

5.3.5. Case generator module

It takes all the data in form of learning module and then generates the Semi Structured XML format. It also creates the resource tree and this is a very important component. Some of the advantages of the resource tree is listed blew.

- It protects against CSRF () attacks.

- It helps to track user behavior as shown in the Figure 5.2. If the normal behavior of the user (first login and then go to the inbox), but when it will directly go the Inbox this will show the malicious behavior.

- It tells us when to start the learning process. If a new URL comes then it sends to blacklist and include it in the resource tree. Whenever the new URL request is greater than the threshold given by administrator then resource tree will start the process of learning.

A typical structure of a resource tree is given in the Figure 5.2.

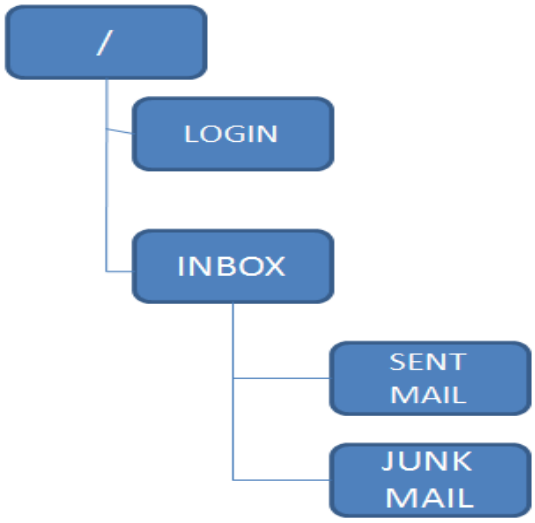


Figure 5.2: Resource tree.

After the end of first iteration Semi structure rules in XML are created for the web application, the structure of the Semi structure rules in XML are shown in figure 5.3.

```
<WARL-GURD>

 <Application name="10.3.18.230"   method_regex= "!^(GET|HEAD|POST)$"
  contentType_regex= "!^(application/x-www-form-urlencoded)">

          <Resource URI ="/webgoat/attack "     Method ="GET"
          param_regex= "!^(Screen|menu|stage|start|swaftokenid)$"  confidence="2"  >

              <parameter   name ="Screen"  datatype=" digits"   regex="^([0-9]+)$"    minlength="2"   maxlength="2"
              mincordinality= "0"   maxcordinality= "1"   Confidance="2"/>
         </Resource>

     </Application>
</WERL-GURD>
```

Figure 5.3: XML rule structure.

5.4. Second iteration of the system

After first iteration learning, the resulting cases will be added to the case base. Whenever administrator flushes the database, the learning process starts before the flushing of database to generate the profile. This means that every time the cases are generated by a new data (as a result of flushing the data). For the generation of mature rules, we need some process for the integration of the previous cases and the new cases. For this purpose we use Case based reasoning.

For the first learning in first iteration, our learning module will calculate these given parameters:

- **Min threshold (MT):** How many minimum requests are sufficient to create the case for the specific resource? If the minimum threshold is met then it generates the case otherwise not.

- **Max threshold (MXT):** Max threshold means maximum number of requests for a resource on which the rule is 100% accurate and its confidence is 100%. By achieving maximum threshold decision made by this rule will be correct.
- **Confidence:** The confidence parameter helps us to determine how strong rule is.

Figure 5.3 shows the Semi structure rules in XML rules that contain these parameters.

5.5. 2nd time learning

After first time learning iteration, we get a case base containing cases. The second time learning takes place in following scenarios

- Before flushing access log database (Time based).
- With Significant changes in the resource tree

After any of the above cases is triggered, algorithm is re-executed and cases are generated in temporary files. We retrieve the cases from previous file by matching their application name and resource URI.

```
<WARL-GURD>

<Application name="10.3.18.230"  method_regex= "!^(GET|HEAD|POST)$"
  contentType_regex= "!^(application/x-www-form-urlencoded)">

        <Resource URI ="/webgoat/attack "    Method ="GET"
        param_regex= "!^(Screen|menu|stage|start|swaftokenid)$"  confidence="2"  >

            <parameter   name ="Screen"  datatype=" digits"   regex="^([0-9]+)$"    minlength="2"   maxlength="2"
            mincordinality= "0"   maxcordinality= "1"   Confidance="2"/>
        </Resource>

    </Application>
</WERL-GURD>
```

Fig 5.4: Selection criteria of CBR

5.5.1. **Case retrieval**

Case retrieval is an important aspect of CBR. Once we have a problem case, we try to retrieve similar one out of the previous cases form the case base. This is carried out on the basis of a similarity metric. Later, the decision about *reusing* the retrieved case(s) is made on the basis of similarity threshold. Figure 5.4 shows the selection criteria of cases from CBR.

- Application name is checked followed by resource name checking.
- If application name and resource are matched, parameters similarity is computed.
- The formula for similarity assessment of cases in case base is “common/common + different”.

5.4.3. Case Learning (Adaptation)

The adaptation process is done by comparing two XML file. First file contains the learned data from the previous cases existing in the case base. This is shown in Figure 5.5.

```
<WARL-GURD>

 <Application name="10.3.18.230"   method_regex= "!^(GET|HEAD|POST)$"
  contentType_regex= "!^(application/x-www-form-urlencoded)">

        <Resource URI ="/webgoat/attack "     Method ="GET"  last modified = "10/08/2010"
        param_regex= "!^(Screen|menu)$"  confidence="2"  >

            <parameter   name ="Screen"  datatype=" digits"   regex="^([0-9]+)$"    minlength="2"   maxlength="2"
            mincordinality= "0"   maxcordinality= "1"   Confidance="2"/>
             <parameter  name ="menu"   datatype=" digits"    regex="^([0-9]+)$"    minlength="4" maxlength="4"
             mincordinality= "0" maxcordinality= "1"    Confidance=2     />

        </Resource>

    </Application>
</WERL-GURD>
```

Fig 5.5: Cases learned from the previous data.

The second file contains the cases emanating from the new learning cycle as shown in the Figure 5.6.

```
<WARL-GURD>

 <Application name="10.3.18.230"   method_regex= "!^(GET|HEAD|POST)$"
  contentType_regex= "!^(application/x-www-form-urlencoded)">

        <Resource URI ="/webgoat/attack "     Method ="GET"  last modified = "10/08/2010"
        param_regex= "!^(Screen|menu|stage)$"  confidence="2"  >

            <parameter   name ="Screen"  datatype=" digits"   regex="^([0-9]+)$"    minlength="2"   maxlength="2"
            mincordinality= "0"   maxcordinality= "1"   Confidance="2"/>
             <parameter  name ="menu"   datatype=" string"    regex="^([-0-9a-zA-Z]+)$"    minlength="6" maxlength="7"
             mincordinality= "0" maxcordinality= "1"    Confidance="1"    />
             <parameter  name ="stage"  datatype=" digits      regex="^([0-9]+)$"   minlength="1" maxlength="1"
             mincordinality= "0" maxcordinality= "1"   Confidance="2"     />

        </Resource>

    </Application>
</WERL-GURD>
```

Fig 5.6: Cases learned from the new data.

Whenever new learning takes place as a result of new case coming, having no corresponding solution in the case base; previous and new case base files are analyzed. If there is some difference in the both of the files (case bases) then the adaptation process starts. Algorithm of adaptation process is given blew.

5.4.3.1. Adaptation process algorithm

1. Match the application name and the resource name. If they are 100% similar then we will fetch the cases and then start the adaptation process.

2. In the adaption process there are three conditions:

a. It will first see the parameter name if it is 100 % matched then see its min regex, max length regex and the min max cordiality. If all if all are 100 % matched as in the first parameter “Screen” then it will add the confidence by adding the previous confidence and the new confidence. As you can see in the results file, the new confidence for the new case is 3.

b. In the case of 2nd parameter named “Menu” this minimum and the maximum value is changed such that parameter properties do not match. Therefore, it will adapt this case.

    i. For minimum value it will apply this function

    “min {v1,v2 }”

    where v1= previous value

    V2= new value

    it takes the min from both values.

    ii. For maximum value it will apply following function

    “max{v1,v2 }”

    where v1= previous value

    V2= new value

    it takes the max from both values.

    iii. If the regex is not matched then it matches first regex with all the current inputs. New inputs that are not matched are sent to the blacklist. If these

inputs match with the system then the previous regex is retained otherwise new one is created.

iv. Because all the things are not 100% same then it applies min confidence function which is" minconf{c1-c2}". It will minus the confidence from case and assigns new value to the case. The 2nd parameter's new confidence for the new case is 1 in the result file.

c. If a parameter comes which does not exist in the previous file then it is added to the previous case.

After the adaptation process is done, the XML file will be generated which contains the results. Figure 5.7 shows these Result files of the previous two files described in Figure 5.5 and 5.6 respectively.

```
<WARL-GURD>

<Application name="10.3.18.230"   method_regex= "!^(GET|HEAD|POST)$"
 contentType_regex= "!^(application/x-www-form-urlencoded)">

        <Resource URI ="/webgoat/attack "     Method ="GET"  last modified = "10/08/2010"
        param_regex= "!^(Screen|menu|stage)$"  confidence="2"  >

            <parameter   name ="Screen"  datatype=" digits"   regex="^([0-9]+)$"    minlength="2"   maxlength="2"
            mincordinality= "0"   maxcordinality= "1"   Confidance="4"/>
             <parameter  name ="menu"   datatype=" string"    regex="^([-0-9a-zA-Z]+)$"    minlength="4" maxlength="7"
             mincordinality= "0" maxcordinality= "1"    Confidance="1"     />
             <parameter  name ="stage"  datatype=" digits      regex="^([0-9]+)$"   minlength="1" maxlength="1"
             mincordinality= "0" maxcordinality= "1"   Confidance= "2"      />


        </Resource>

    </Application>
</WERL-GURD>
```

Fig 5.7: Result.

5.4.4. Case maintenance

CBR repository keeps on growing so CBR maintenance becomes important. CBR case base may contain the redundant and useless rules e.g. if we have a web page that is not used since last year then the rule is not useful so we have to delete it from the case base. While Maintaining CBR, we have to enter another property of the rule named “last modified as shown in figure 5.8.

```
<WARL-GURD>

 <Application name="10.3.18.230"   method_regex= "!^(GET|HEAD|POST)$"
  contentType_regex= "!^(application/x-www-form-urlencoded)">

        <Resource URI ="/webgoat/attack "     Method ="GET"  last modified = "10/08/2010"
        param_regex= "!^(Screen|menu)$"  confidence="2"  >

            <parameter   name ="Screen"  datatype=" digits"   regex="^([0-9]+)$"    minlength="2"   maxlength="2"
            mincordinality= "0"   maxcordinality= "1"   Confidance="2"/>
             <parameter  name ="menu"   datatype=" digits"    regex="^([0-9]+)$"    minlength="4" maxlength="4"
             mincordinality= "0" maxcordinality= "1"    Confidance=2     />


        </Resource>

    </Application>
</WERL-GURD>
```

Fig 5.8: Case based maintenance.

5.5. Case based reasoning advantages

1. Maturity: makes the rule base mature by integrating previous and newly learned cases.
2. Adaptation: provides mature cases and decides the confidence factor of the case.
3. Confidence (weak/strong) cases: system can decide upon weak and strong cases. Also, if some packet which is stopped by a week case then it will generate mail to the administrator so that if a false positive comes we can stop it. If a week case stops an attack its confidence will increase and if a strong case makes a false positives or an attack its confidence will decrease.

4. Dynamicity of the application: changing nature of the parameters can decide how dynamic a web application is.

5.6. Main features of the module

1. Log scanning for the case making use of interceptor.
2. Request validation on the basis of response generated by web server.
3. Checks the min length and max length
4. Checks the min cardinality and max cardinality.
5. Learning of Free and fix parameters using cardinality function.
6. Calculates the confidence level of the parameter basis on number of requests.
7. On the basis of confidence, we can introduce decision weighting of the system.
8. Matching of regex with the input
9. Semi Structured XML case generation.
10. Automatic creation of resource tree
11. Uses CBR for the integration of the new data, the case bases and learn the new cases.
12. Learns when to add the case in the case base and when to delete it.
13. Matures case creation.

5.7. Summary

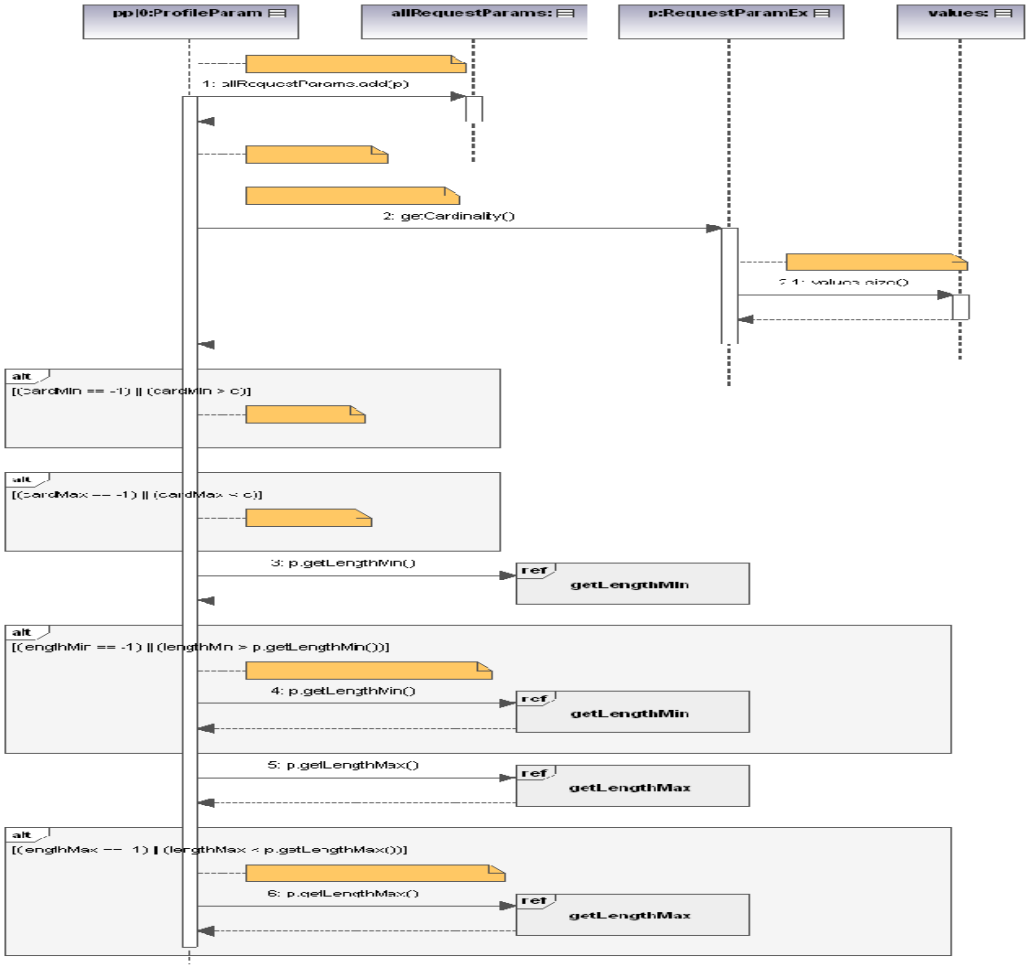
This chapter contains the architecture of the system. It describes the processing of different Components of system in detail, different advantages of system and how the system cater most of the problems discussed in chapter 3.

Chapter 6: Design and implementation

This chapter describes the design and implementation of Web Application Model Generator. The chapter is comprised of sequence diagrams, and class diagrams of WAMG. Furthermore it contains the description of log scanning and its parsing details.

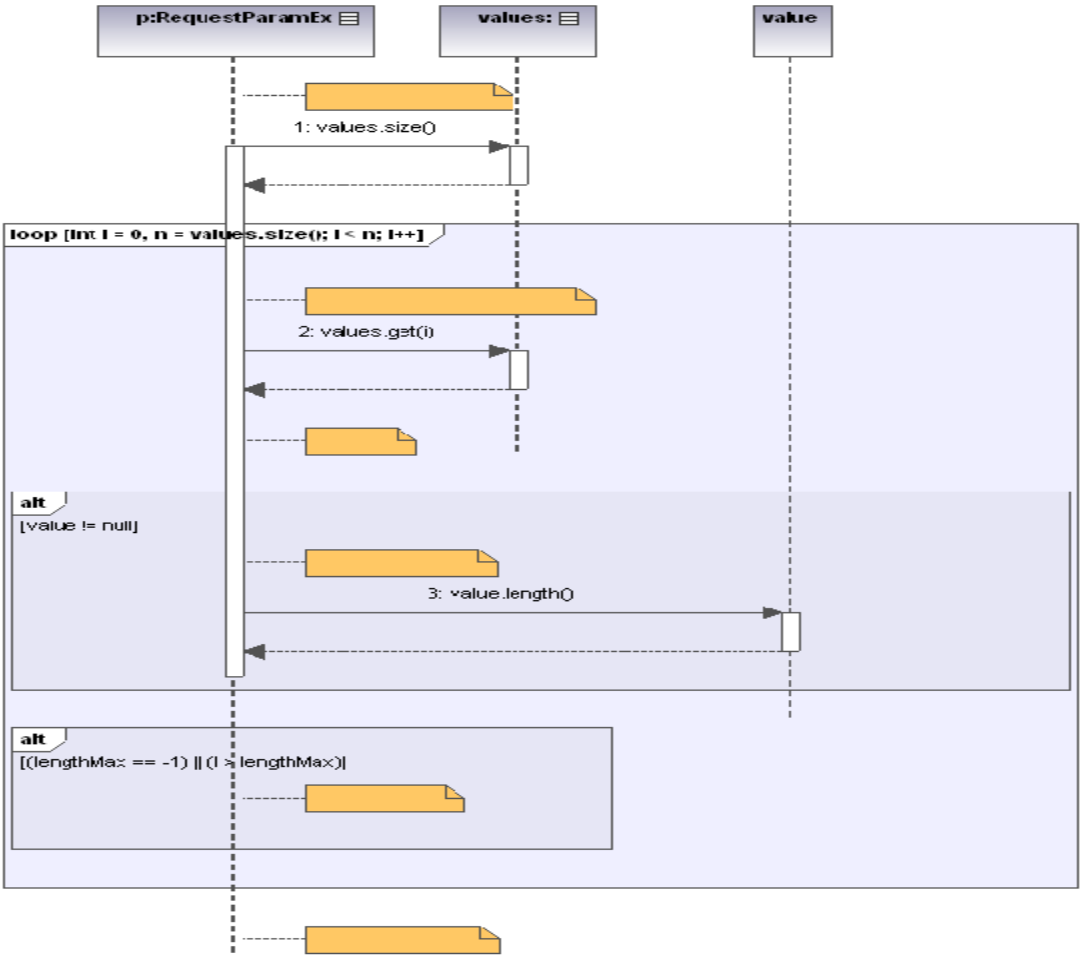
6.1. WAMG Sequence Diagrams.

This section contains sequence diagrams. It shows that how processes are communicating with each other. It shows placement or sequence of the system. System first takes the request from the access log, choose the valid request among those requests and generate the rules for web application. Whenever the system repeat the first iteration (generate White List), CBR starts analyzing new file with the old one, if it sees changes in different parameter, it adapt the new rules and generate a new file.



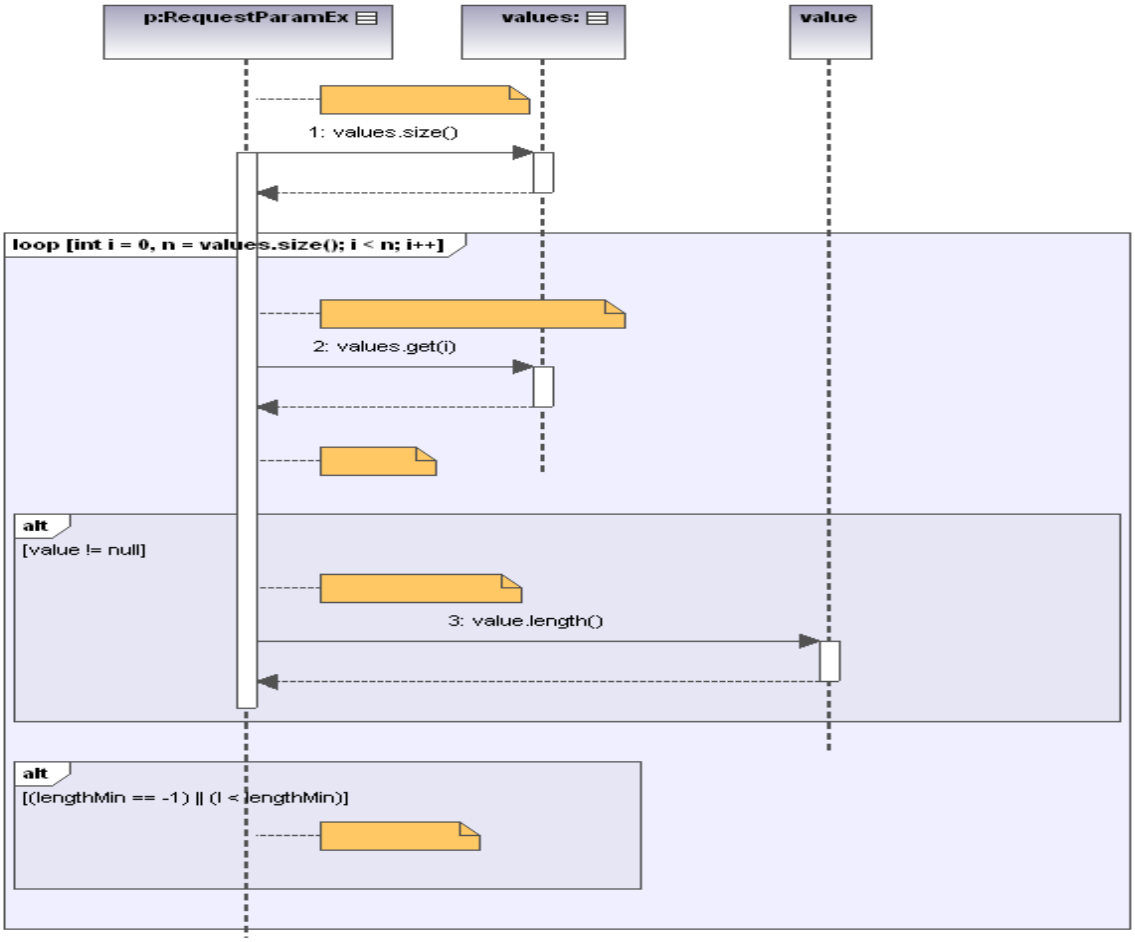
6.1.1. Accommodate parameter

Parameters are the basic unit of a web application. A web application consists of many parameters. Most of the attacks executes with tempering these parameters. Thus securing parameter of the web applications are a major task. For this purpose Discrete or Free parameter reorganization is very important. Cardinality gives us the information that either the parameter is discrete or fixed. With the analysis of different inputs we can take the minimum and maximum length of the parameter which prevents buffer overflow attacks.



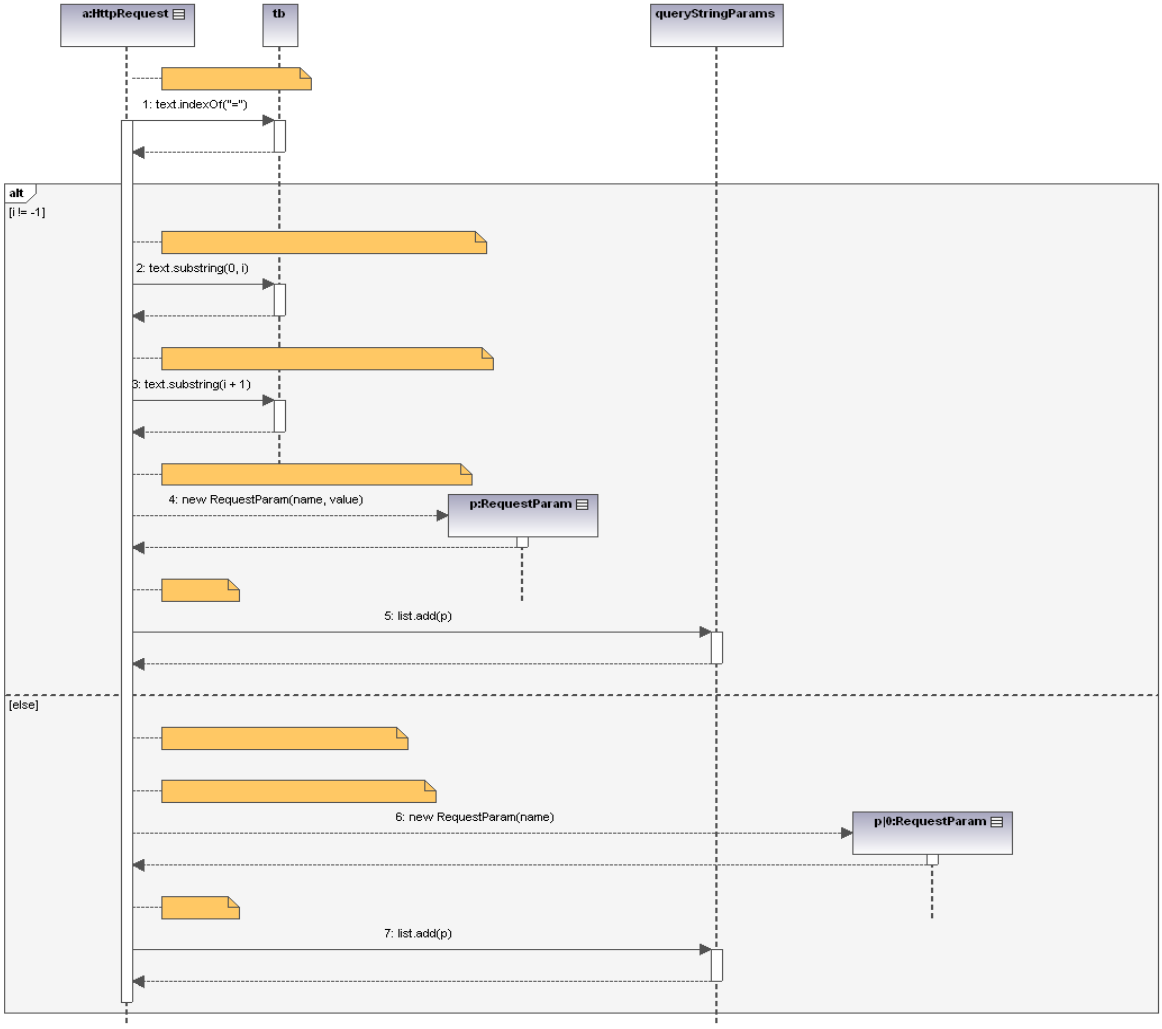
6.1.2. Get length MAX

Maximum and minimum input values help us to prevent the buffer overflow attack.With the analyses of different inputs given by users we can obtain these values range. This diagram describes how maximum values are obtained from the access log of the web application. First we get all the values from the access log and then analyze these values and take the biggest of these values as maximum length



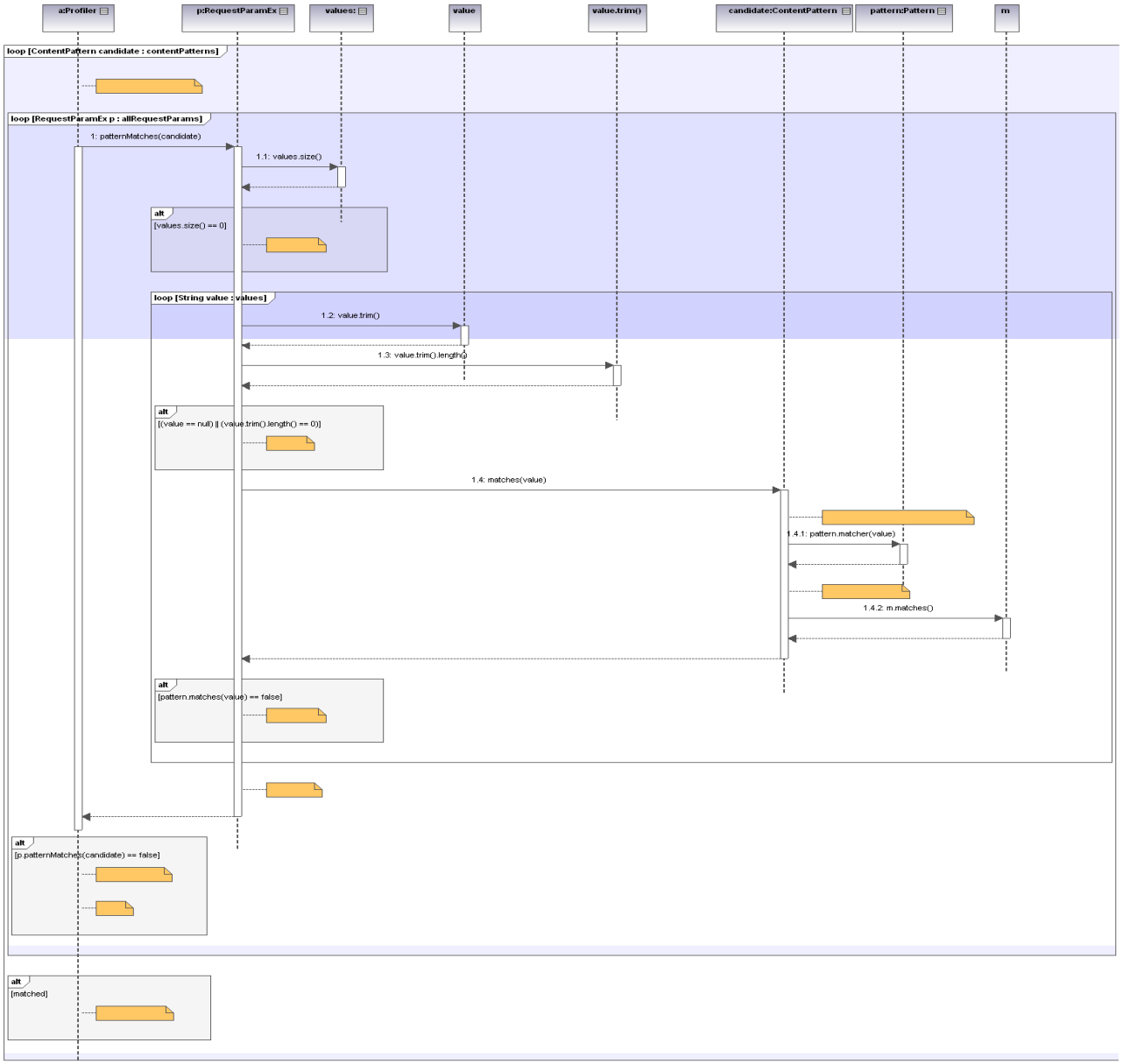
6.1.3. Get minimum length

Maximum and minimum input values help us to prevent the buffer overflow attack and with the analysis of different inputs given by the users, we can get these values. This diagram describes how minimum values are obtained from the access log of web application.



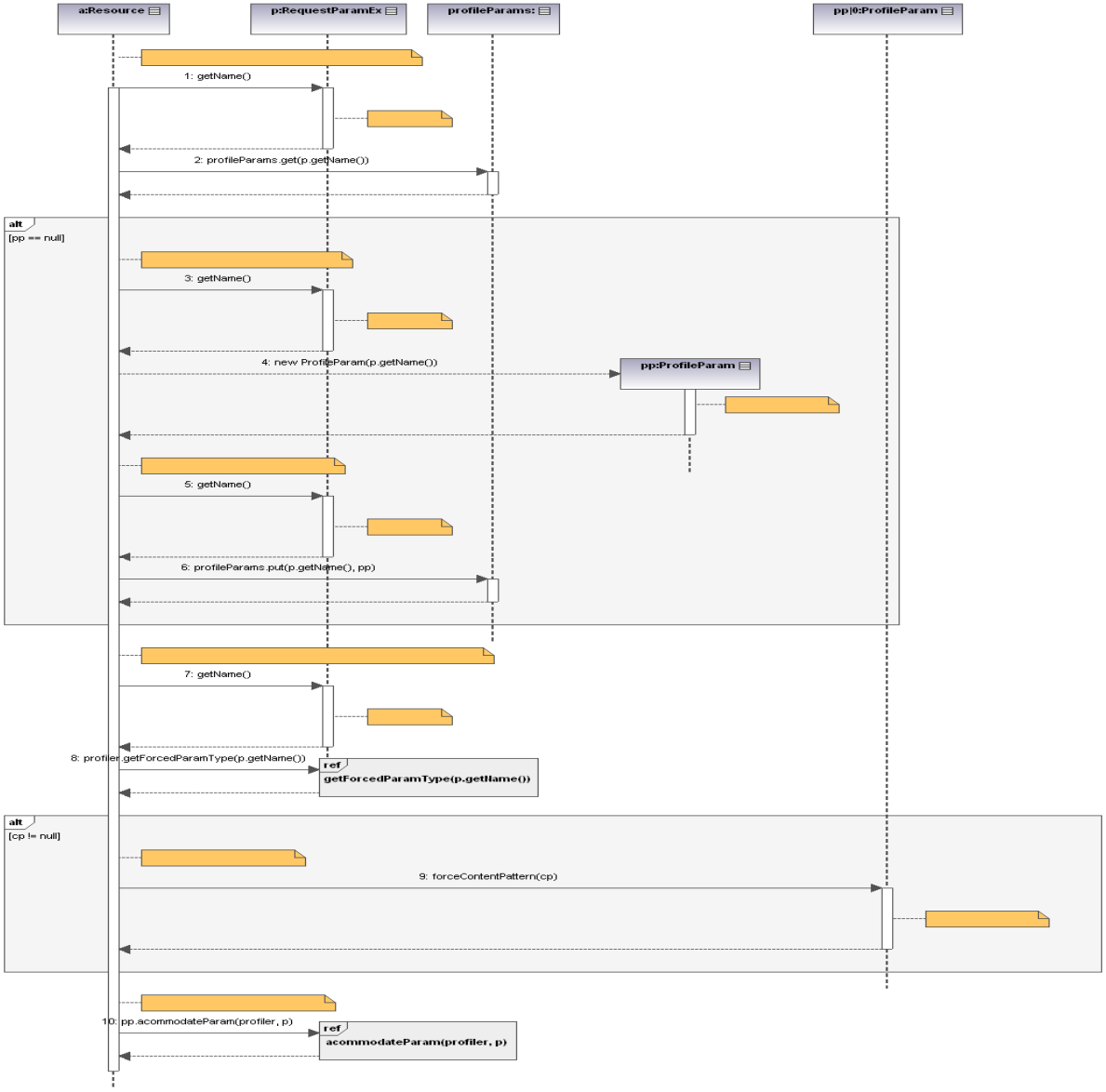
6.1.4. Parse parameter

System generates cases on the basis of web application parameters. This sequence diagram show that how it recognize the parameter and then added it into the list. It fist access the name and value of the parameter form the access log and then add the information into a list.



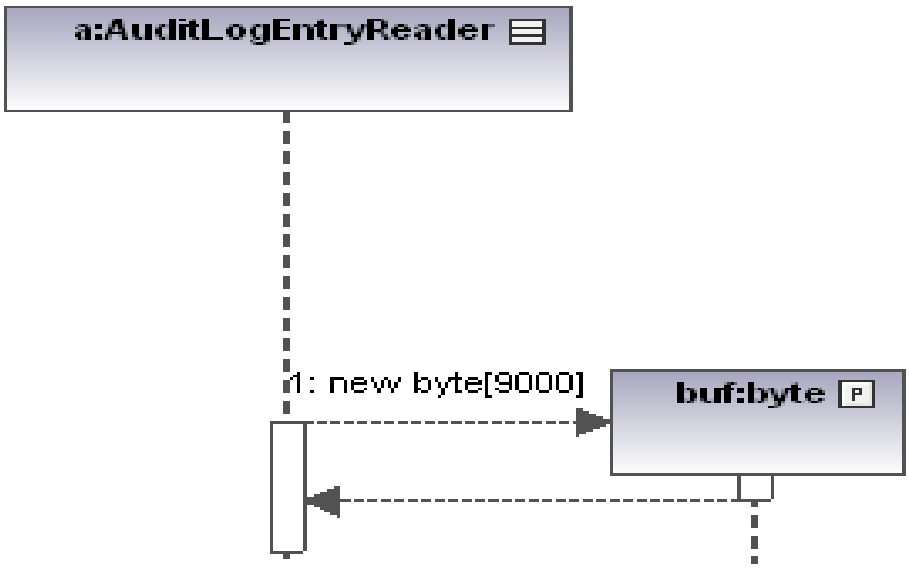
6.1.5. Match content parameter

This figure describes that how contents of the parameters are processed. When request comes to the web application, it checks the parameter value length and then its Regex followed by results.



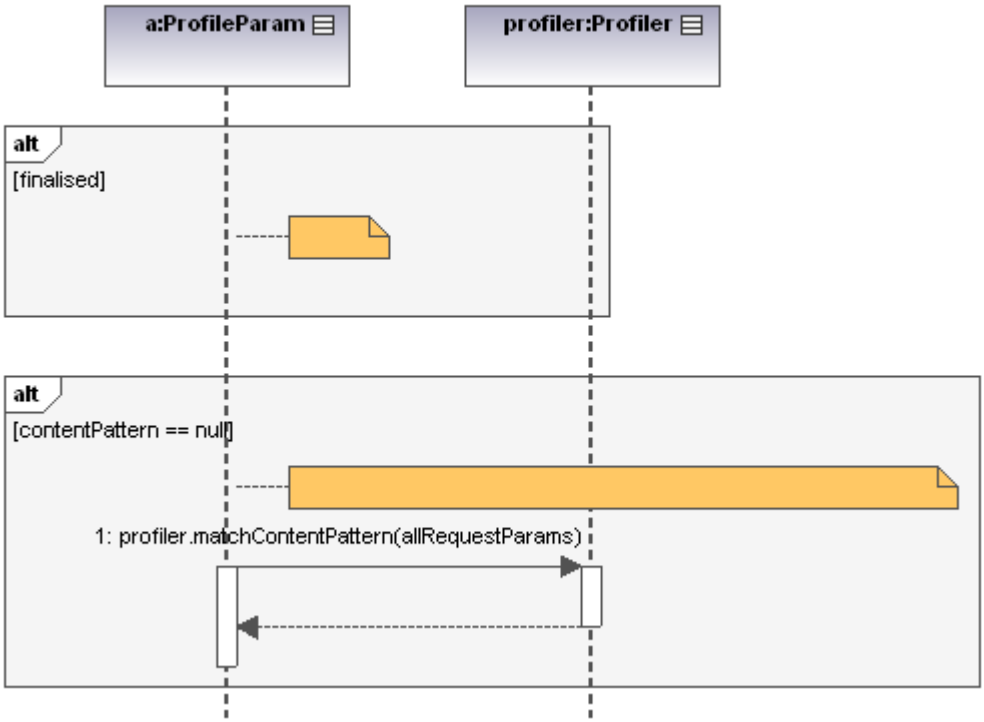
6.1.6. Process parameter

This sequence diagram shows the process of a case creation of a parameter. First we get the name of the parameter and check its type to assign an appropriate type of regex to parameter. We process the parameters by analyzing various inputs of the parameter which helps us to decide the regex and the data types of the parameter.



6.1.7. Audit log entry reader.

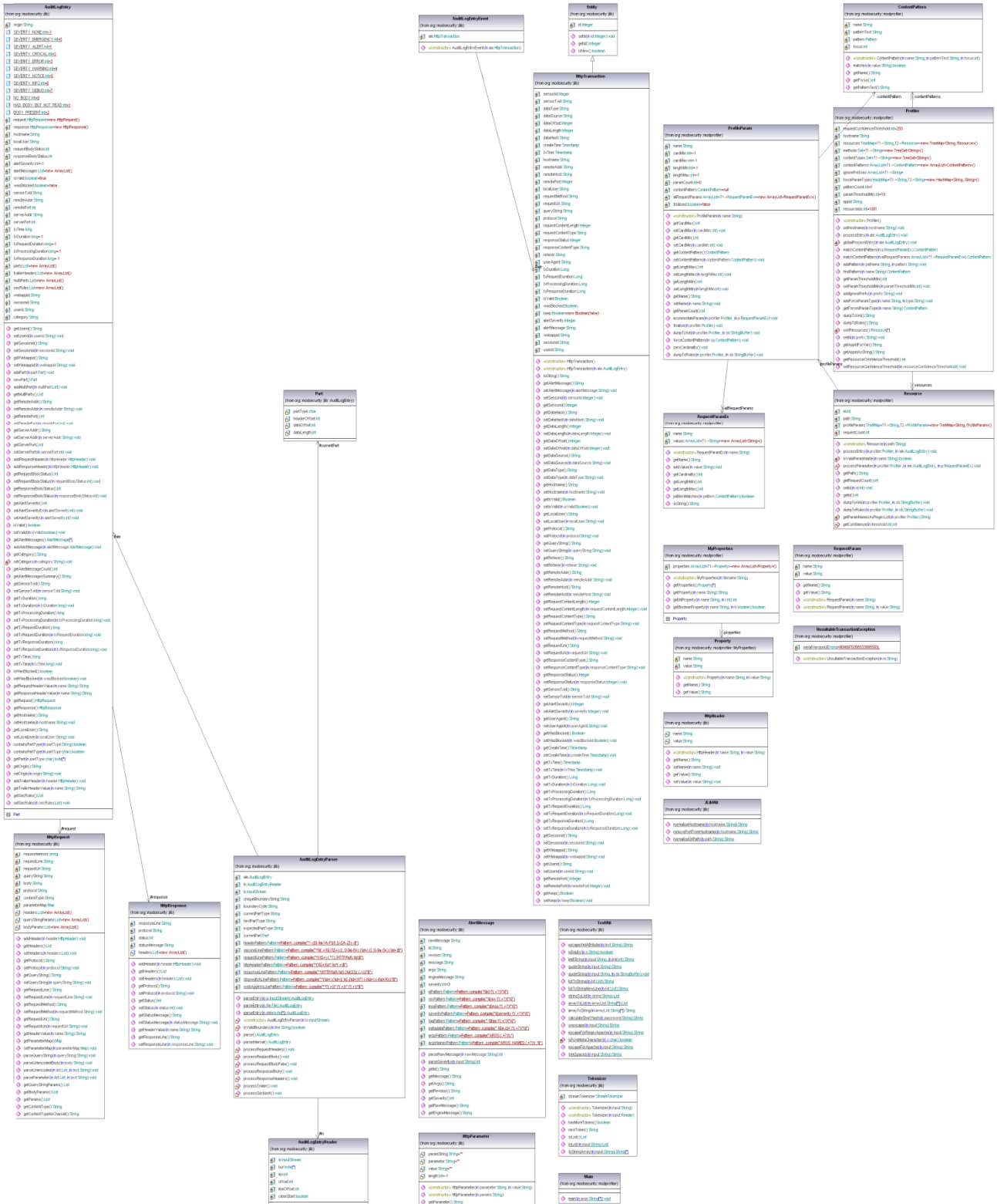
For the generation of CBR cases first we need the http request and response data from the access log. This sequence diagram shows the procedure of fetching data from access log then we establish a connection with the database and fetch all the data from the database in a byte buffer.



6.1.8. Match content patterns.

When system gathers all the inputs from the access log and then match those inputs with the pattern repository (that are regex) and choose a suitable pattern according to the input. The regex is actually the pattern of the inputs, this will help us to allow only the benign inputs and discard malicious inputs.

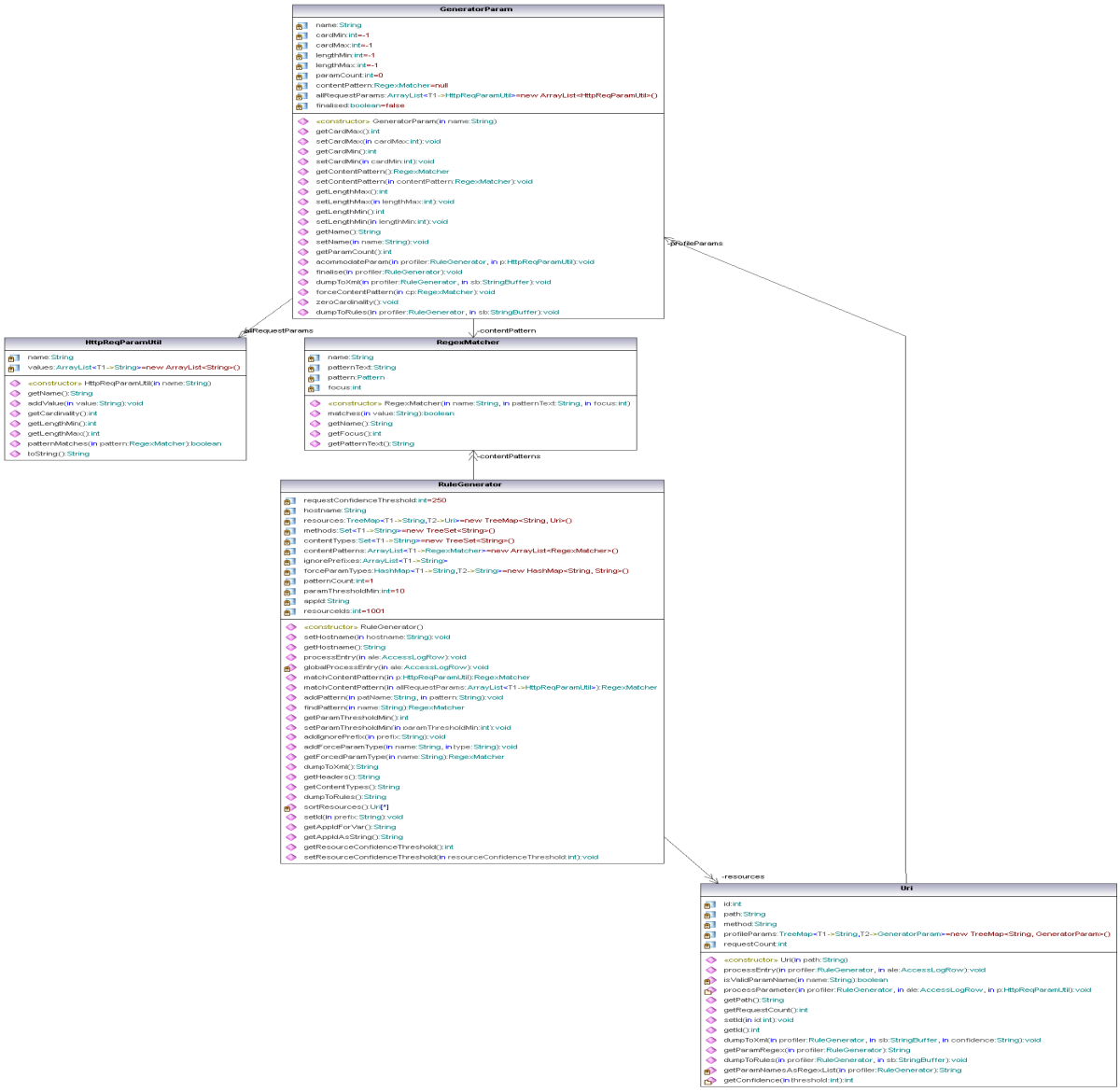
6.2. Class Diagram



Partial class diagrams

6.2.2. Class diagram for learning data using access log.

The learner module is responsible for the creation of WAMG cases it will take data from access log .Whenever a HTTP request comes it will be recorded in the access log and when the server send the response against the request it will also be recorded in the access log. On the basis of request and response learner will decide that which request is legitimate and which request is illegitimate.



6.2.3. Class diagram for rule generator.

This module is responsible for generating WAMG rules. The class *generator param* will generate the contents of the parameter like parameter name, lengths, cordiality etc. Then sends it to the *regex matcher* class which will match the regex and then all the information will go to the rule generator class and it will generate the WAMG rules.

6.2.4. Class diagram for HTTP transaction.

HTTP transaction class sets and gets the HTTP headers from access log. This class gets the HTTP properties and headers from access log and then the values of this class is accessed by the *property class* which send it to the *learner class* to learn the parameter of the application.


CbrRule
(from secplus::analyzer::cbr::vo)
applicationURL:String
methodRegex:String
contentType_regex:String
resourceURI:String
resourceMethodType:String
paramRegex:String
confidence:String
parameterName:String
parameterDatatype:String
parameterMaxLength:int
parameterMinLength:int
parameterRegEx:String
minCord:String
maxCord:String
paramID:int
applicationType:String
Property1:StoreParameters
Property2:Queue<T1->Main>
getConfidence():String
setConfidence(in confidence:String):void
getContentType_regex():String
setContentType_regex(in contentType_regex:String):void
getMethodRegex():String
setMethodRegex(in methodRegex:String):void
getParamRegex():String
setParamRegex(in paramRegex:String):void
getMaxCord():String
setMaxCord(in maxCord:String):void
getMinCord():String
setMinCord(in minCord:String):void
getApplicationType():String
setApplicationType(in applicationType:String):void
getParamID():int
setParamID(in paramID:int):void
getApplicationURL():String
setApplicationURL(in applicationURL:String):void
getParameterDatatype():String
setParameterDatatype(in parameterDatatype:String):void
getParameterMaxLength():int
setParameterMaxLength(in parameterMaxLength:int):void
getParameterMinLength():int
setParameterMinLength(in parameterMinLength:int):void
getParameterName():String
setParameterName(in parameterName:String):void
getParameterRegEx():String
setParameterRegEx(in parameterRegEx:String):void
getResourceMethodType():String
setResourceMethodType(in resourceMethodType:String):void
getResourceURI():String
setResourceURI(in resourceURI:String):void
y2
y1
StoreParameters
(from secplus::analyzer::cbr::vo)
parameterName:String
parameterDatatype:String
parameterMaxLength:int
parameterMinLength:int
parameterRegEx:String
minCord:String
maxCord:String
paramID:int
Property1:CbrRule
getMaxCord():String
setMaxCord(in maxCord:String):void
getMinCord():String
setMinCord(in minCord:String):void
getParamID():int
setParamID(in paramID:int):void
getParameterDatatype():String
setParameterDatatype(in parameterDatatype:String):vo
getParameterMaxLength():int
setParameterMaxLength(in parameterMaxLength:int):vc
CBRRulesHandler
(from secplus::analyzer::cbr::dal)
pathToXML:String
Property1:StoreParameters
Property2:CbrRule
«constructor» CBRRulesHandler(in path:String)
getDocument():Document
insertWhiteListRule(in ruleVariables:CbrRule):int
getApplicationRules(in applicationName:String):ArrayList<T1->CbrRule>
getApplicationResources(in aplicationName:String, in doc:Document):ArrayList<T1->StoreResources>
getResources(in appElement:Element):ArrayList<T1->StoreResources>
getResourceParameters(in aplicationName:String, in resName:String, in resType:String, in doc:Document):ArrayList<T1->StoreParameters>
getParametersList(in paramElement:Element):ArrayList<T1->StoreParameters>
getAllApplications():ArrayList<T1->StoreApplications>
addParameters(in doc:Document, in ruleVariables:CbrRule, in resElement:Element):Element
addResource(in doc:Document, in ruleVariables:CbrRule, in appElement:Element):Element
addApplication(in doc:Document, in ruleVariables:CbrRule):Element
deleteParameter(in ruleVariables:CbrRule):Element
deleteApplication(in ruleVariables:CbrRule):Element
deleteResource(in ruleVariables:CbrRule):Element
editParameter(in oldRuleVariable:CbrRule, in newRuleVariable:CbrRule):int
ApplicationResources
(from secplus::analyzer::cbr::vo)
applicationNames:String
resourcesParameters:HashMap
Property1:CbrRule
getApplicationNames():String
setApplicationNames(in applicationNames:String):void
getResourcesParameters():HashMap
setResourcesParameters(in resourcesParameters:HashMap):void
StoreApplications
(from secplus::analyzer::cbr::vo)
appName:String
methodRegex:String
contentTypeRegex:String
appNode:Node
Property1:CbrRule
getAppNode():Node
setAppNode(in appNode:Node):void
getContentTypeRegex():String
setContentTypeRegex(in contentTypeRegex:String):void
getMethodRegex():String
setMethodRegex(in methodRegex:String):void
getAppName():String
setAppName(in appName:String):void
StoreResources
(from secplus::analyzer::cbr::vo)
resURI:String
resMT:String
resRegex:String
resConf:String
Property1:CbrRule
getResConf():String
setResConf(in resConf:String):void
getResRegex():String
setResRegex(in resRegex:String):void
getResMT():String
setResMT(in resDT:String):void
getResURI():String
setResURI(in resURI:String):void
Main
(from secplus::analyzer::cbr::dal)
Property1:CbrRule
main(in args:String[*]):void
ruleGenerate(in rules1:List, in rules2:List):List
isSameRule(in rule1:CbrRule, in rule2:CbrRule):boolean
isSameResource(in rule1:CbrRule, in rule2:CbrRule):boolean
ParameterID
(from secplus::analyzer::cbr::dal)
parameterValue:int
Property1:CbrRule
getParameterDocument():Document
getParameterValue():int
setParameterValue(in parameterValue:int):void
ResourceParameters
(from secplus::analyzer::cbr::dal)
resourceURI:String
resourceMethodType:String
parametersList:HashMap
getParametersList():HashMap
setParametersList(in parametersList:HashMap):void
getResourceMethodType():String
setResourceMethodType(in resourceMethodType:String):void
CBRRulesManuplator
(from secplus::analyzer::cbr::dal)
pathToXML:String="E:/rule.xml"
whitelistRulesCache:HashMap=new HashMap()
getDocument():Document
whitelistProfileCreator():HashMap
getApplicationResources(in aplicationName:String, in doc:Document):ArrayList<T1->StoreResources>
getResources(in appElement:Element):ArrayList<T1->StoreResources>
getResourceParameters(in aplicationName:String, in resName:String, in resType:String, in doc:Document):ArrayList<T1->StoreParameters>

6.2.5. Class diagram OF CBR module.

CBR is the most important module of WAMG. Its responsibilities are to compare the previous and new files and then according to the situation it adapts the rules and then makes a new file out of it. This module helps to improve the White List profile ,learns from previous mistakes and then corrects itself, also control the size of knowledge base by eliminating the useless rules of the web application.

6.2. Summary

This chapter contains the Design and Implementation of WAMG in term of class and sequence Diagrams. Each implementation item presented has also been described.

Chapter 7: Evaluation

7.1. System evaluation

This sole purpose of this system is to generate white list rules for web applications in semi structured XML cases. The system is evaluated to generate XML cases successfully for all kind of Web Applications and to stop injection flaws, forceful browsing, and buffer overflow attacks. The false positives and false negative rate is another deciding factor for the success of this module.

7.2. Evaluation environment.

The proposed system is evaluated in the following execution environment

Processer	2.0 GHz core 2 duo.
RAM	2 GB
OS	Microsoft Window 7
Java	Version 1.6

Table:7.1 : Evaluation Enviournment

7.3. Evaluation criteria

Evaluation is performed with the most profound way of evaluating intrusion detection system.The most effective way of evaluating WAMG effectiveness is to calculate its false positive and false negative rates.

False Positive = F_P: the total number of benign messages that are classified as malicious

False Negative = F_N: the total number of malicious messages that are classified as benign

Total #Normal = T_N: the total number of normal or benign messages

Total #Attack = TA: the total number of malicious messages

Detection Rate= [(TA-FN)/TA]*100 &**False Alarm Rate** = [FP/TN]*100

7.4. Tools used

The tools used to test this module effectiveness are given blew.

Name	Description
WebScarab	This tool is developed by OWASP for testing web application Firewall. It intercepts an HTTP request and then the attacker can tamper the HTTP packet and send to server [16].
WebGoat	This tool is used to test the web application firewall this is a Web Site which contain security lessons. This can help to make see the effectiveness of the firewall [17].
Badstore	This is a web application and the purpose of this application is to understand the vulnerabilities existed in the web application [18].
Nikto	Nikto is an open source web application scanner which uses to comprehensively test the web servers. It contains thousands of dangerous file to test the web applications [19].
Moodle	This is a web application template used for content management.

Table:7.2 :Tools Used

7.5. Evaluation

Attack Type	#Normal Record	#Attack Record	False Positive	False Negative	Detection Rate	False Alarm Rate
XSS	350	200	2	4	98.00	0.57
SQL Injection	280	150	2	1	99.33	0.72
HTTP Request Splitting	200	4	2	0	100.00	1.00
Buffer overflow	300	250	0	0	100.00	0.00
HTTP Response Splitting	100	5	1	2	98.00	20.00
JS charcode	20	1	0	0	100.00	0.00

Table: 7.2: Evaluation

We use three web applications Web Goat, Moodle and World Press for this purpose. Our module uses 18 regular expressions which represent widely used data types like digits, hex etc and popular fields e.g. passwords, credit cards etc. Above table shows the false positives and false negative rates and the number of normal parameter and malicious parameter used to obtain these results. Detection rate shows that out of malicious parameters how much are detected. False alarm rate depicts that how many benign packets are detected as malicious, false alarm rate is also very low against all the attacks. This technique prevent XSS, SQL injection, HTTP request splitting, buffer overflow, HTTP response splitting, JS charcode attacks efficiently.

7.6. Summary

This chapter contains results of the experiments. It shows us that the purposed technique is efficient against various web application attacks. It also includes the test environment and tools which are used for testing purpose.

Chapter 8: Conclusion and Future Work

8. Conclusion

Rapid increase in application level attacks brings the focus of security experts to existing security systems, figuring out their limitations and weaknesses. A lot of research has been done on network based systems and they are quite effective only for network level exploits.

The most prevalent security mechanisms are taking care of network level attacks. Few security systems are operating at application level but are prove to be quite ineffective in providing solid defense against application level attacks. For providing web application security there are two conventional techniques Black List and White List. Black List is proven to be inadequate for prevention of web application attacks due to overhead of signatures updates. White List is proven to be effective but there are some problems with this technique. The purposed technique addresses the conventional problems of White List and also helps to detect various web application attacks. The learning capability of the system detects new changes in the application and craft rules according to the changes. It also detects errors in the knowledge base and corrects itself. The Evaluation shows that this technique is effective against various web application attacks.

8.1. Future Work

- Introduction of multithreading in the White List generation module, will make system more efficient.
- Introduction of Threshold leaning in the White List generation module.

References

[1] MITRE, “Web application attacks statistic”, http://www.mitre.org/.

[2] WHITE HAT, “Web application attacks statistic”, http://www.whitehatsec.com.

[3] OWASP, “Web application attacks statistic”, http://www.owasp.org.

[4] Acunetix ,“Web application attacks statistic”, http://www.acunetix.com/.

[5] Jeff Orloff, “Applicure Webhacking Facts and Figure”, http://www.applicure.com/blog/web-application-hacking-facts-figures.

[6] Robert Abela, Website Defender, “General facts and figure on hacking”. http://www.sitesecuritymonitor.com/web-hacking-facts/.

[7] Martin Roesch. “Snort: Lightweight intrusion detection for networks”. *In LISA, pages 229–238. USENIX, 1999.*

[8] Christian Bockermann, Ingo Mierswa, Katharina Morik, ”On the Automated Creation of Understandable Positive Security Models for Web Applications”, *Sixth Annual IEEE International Conference on Pervasive Computing and Communications.*

[9].Ofer Shezaf, “Positive Security Model for Web Applications, Challenges and Promise”, *OWASP IL Chapter leader CTO, Breach Security Positive Security Model.*

[10] Frederick Hayes, “Rule Based systems”, *Communications of the ACM ,September 1995.*

[11].Sankar. K. PAL, SIMON C. K. SHIU,” Foundation of soft case based reasoning”, *ISBN 0-471-08635-5*

[12] Ali Hur, “Semantic based detection of application layer attacks using ontology”,*SEECS 2009*.

[13] William Robertson, Giovanni Vigna, Christopher Kruegel, and Richard A. Kemmerer, “Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks”, *Proceeding of the Network and Distributed System Security (NDSS) Symposium San Diego, CA February 2006.*

[14] Federico Maggi, William Robertson, Christopher Kruegel, and Giovanni Vigna, “Protecting a Moving Target: Addressing Web Application Concept Drift”, *Raid 2009.*

[15] Processor, “Black list VS White” List.http://www.processor.com/articles//P2724/33p24/33p24chart1.pdf?guid=.

[16] OWASP, “Web Scarab”, http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project.

[17] OWASP, “Web Goat”, http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project.

[18] Bad Store, “Bad store” ,http://www.badstore.net/.

[19] CIRT.net, “Nikto”, http://cirt.net/nikto2

[20] Christopher Kruegel, Giovanni Vigna, “Anomaly Detection of Web-based Attacks”, *CCS’03, October 27–31, 2003, Washington, DC, USA.*

[21]Vern Paxson, “A System for Detecting Network Intruders in Real-Time”, *7th USENIX Security SymposiumSan Antonio, Texas, January 26-29, 1998.*

[22] Kenneth L. Ingham, Hajime Inoue, “Comparing Anomaly Detection Techniques for HTTP”, *RAID 2007.*

[23] Grzegorz J. Nalepa,AntoniLig, ”Designing Reliable Web Security Systems Using Rule-Based Systems Approach”, *AWIC 2003.*

[24] Yao-Wen Huang, Shih-Kun Huang, Chung-Hung Tsai, “Web Application Security Assessment by Fault Injection and Behavior Monitoring”, *ACM ,May 20-24, 2003*.

[25] Armorlogic, “Profance”, http://www.armorlogic.com/web-application-firewall.html.

[26] ModSecurity, “Mod security”, http://www.modsecurity.org.

[27] Bruce Schneier Blog, “black listing vs. white listing”, http://www.schneier.com/blog/archives/2011/01/whitelisting_vs.html.