International Journal For Technological Research In Engineering
Volume 3, Issue 9, May-2016 *ISSN (Online): 2347 - 4718*

A NOVEL SELF DEFENCE APPROACH FOR WEB APPLICATION SECURITY

Harsh A Bhatt[1], Prof. Tejendra Thakur[2]
[1]PG Student, GTU PG School, Gandhinagar, India, [2]Assistant Professor, UCET, Ahmedabad, India

Abstract: In this era of technology World Wide Web is one of the most powerful communication channel and service providers for information delivery over internet today. Most of the peoples are using web applications. Securing web is like securing our nation. So, internet security is very much encouraging task for us. Different kinds of attacks on web applications are happened like SQL Injection attack, XSS attack, DoS attack. We have also survey of such attacks happening in last three to four years. This paper is an effort to explore the effectiveness of an ontological knowledgebase for modeling, configuring and querying over WAF configurations. We are also discussed how to secure web application from different kind of attacks using Ontology. We have used ModSecurity web application firewall and try to improve default security policy rules of it. We have used Apache Server. Our results show that our proposed work meaningfully improves configuration errors, security policy rules of ModSecurity for SQLI attack. SQL Injection attack is our prime attack and based on this attack ontology we have create custom security policy rule for prevention against attack. DVWA is used to perform SQLI attack.

Keywords: SQL Injection attack, Cross Site Scripting (XSS), ModSecurity Web Application Firewall, Ontology, Security, Damn Vulnerable Web Application (DVWA), Apache

I. INTRODUCTION

Web applications are widely used in all around the world. Web application security is the prime concern for us. Web Application Firewall (WAF) is a security measure to protect web applications from external attacks that exploit vulnerabilities in web applications.[1] WAF provides operational security that mitigates the impact of attacks, but it does not eliminate the vulnerabilities[7] present in the web application implementation. Web application security through WAF requires knowledge of HTTP transactions parameters shared between the web application and client i.e. Request header, Request body etc. A comprehensive evaluation criterion for WAFs has been standardized by Web Application Security Consortium as WAFEC (Web Application Firewall Evaluation Criteria).WAFEC provides a comprehensive documentation in understanding the role of WAF for securing web applications and listing of evaluation criteria for estimating WAF performance. The evaluation criteria covers development architecture, HTTP and HTML support, detection techniques, protection techniques, logging, reporting and performance and is comprehensive enough for evaluating QoP for WAFs.[12] This standard covers three protection strategies for web application which includes external patching (virtual patching), positive security model which provide protection independent of the input validation and requires rules to be continuously updated for the web application. Management of WAF rule configuration for administrators has been a complex and error-prone task. Typical errors involved in WAF configuration are invalid rule syntax, writing redundant or illogical rules resulting in poor WAF configuration.[13] Some basic terms for our work is given below:

A. SQL Injection Attack

SQL Injection refers to an injection attack wherein an attacker can execute malicious SQL statements (also commonly referred to as a malicious payload) that control a web application's database server. An SQL Injection can destroy your database.[15]

B. ModSecurity Web Application Firewall

ModSecurity is a hybrid WAF that relies on the host web server for some of the work.[9] The only supported web server at the moment is Apache 2.X. It filters incoming and outgoing data and is able to stop traffic that is considered malicious according to a set of predefined rules.
Rules are created and edited using a simple text format, which affords you great flexibility in writing your own rules. Once you master the syntax of ModSecurity rules you will be able to quickly write your own rules to block a new exploit or stop a vulnerability being taken advantage of. ModSecurity is a tool that will help you secure your web applications. ModSecurity resolves around two things: 1. Configuration and 2. Rules. [10]

C. Ontology

Ontology refers to formal, explicit specification of a shared conceptualization.
It deals with the question of appearance vs. reality:

- What characterizes being?
- Eventually, what is being?
- How should things be classified?

Ontology represents detailed relationships between concepts. Ontology is an explicit specification of a representational vocabulary for a domain: definitions of classes, relations, functions, constraints and other objects. Most commonly used ontology language are XML, RDF and RDFS, and OWL.[8] It gives the actual specification about any things in proper manner.

D. Damn Vulnerable Web Application (DVWA)

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications.

II. PROBLEM IDENTIFICATION

In survey part we have done analysis of recent web based attacks. Following figure shows the Top 10 source countries for web application attacks, Q1 2015

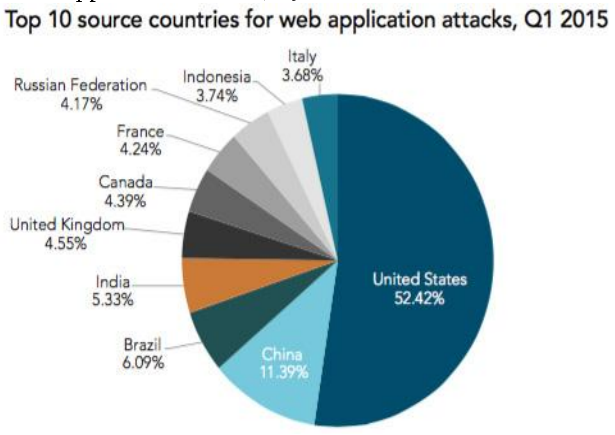


Figure 1: Top 10 Source countries for web application attacks, Q1 2015[6]

After reviewing all the different papers and survey of different attacks we have come to know that how the web attacks are harmful and how it will affect in terms of financial loss in all over world.

Following fig.2 shows problem identification in existing system:

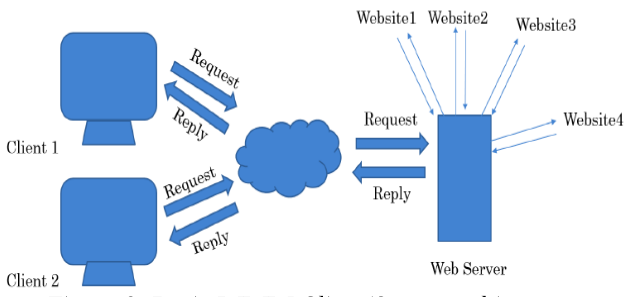


Figure 2: Basic WWW Client/Server architecture

This is basic architecture which shows two different clients are send an HTTP Request to web server through an internet medium. There are four different website are hosted on main web server. When web server get request from client it will forwarded to website and website sent a reply to web server according to client's request and finally web server sent a reply to client through internet. The main problem in this architecture is if any of this multiple website which is hosted on main web server has any types of vulnerabilities then an attacker will exploit it. As a result a main web server will be down or it will be hacked and due to this big financial loss should be occurred. Existing system is working with Web Application Firewall to protect web-applications from such kind of attacks.[11] Different WAF like mod_security, mod_evasive, etc. are used to prevent web-applications from different attacks like XSS, DoS, SQL Injection, etc.

Following are some problems with the existing system that can be identified:

- WAF are difficult to configure
- WAF's rules can be difficult to write and understand
- WAF's rules can also be overwrite
- Web server will be hacked due to exploitation of different kind of vulnerabilities
- WAF is difficult to audit
- Overall existing system still lacks in efficient configuration

III. PROPOSED WORK

To Countermeasure the problem in existing system and to overcome lacks of configuration in WAF, we have proposed to create Ontological knowledge base of mod_security web application firewall and find possible applications of this ontological knowledge base. Fig.3 shows our proposed architecture:

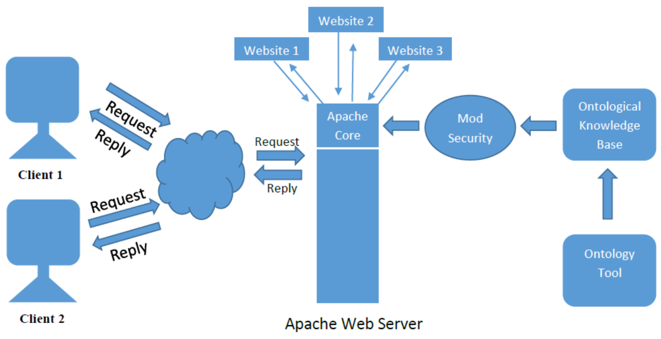


Figure 3: Proposed architecture

We have create ontological knowledgebase for ModSecurity. Using this ontological knowledge base we have modify default security policy rule and write our custom security policy rule for ModSecurity.

IV. IMPLEMENTATION DETAILS

First we start our implementation work with creating different ontology for ModSecurity. We used protégé tool[5] to create ontology. Following fig.4 and fig.5 shows ontology for Logging and Rule Language of ModSecurity.

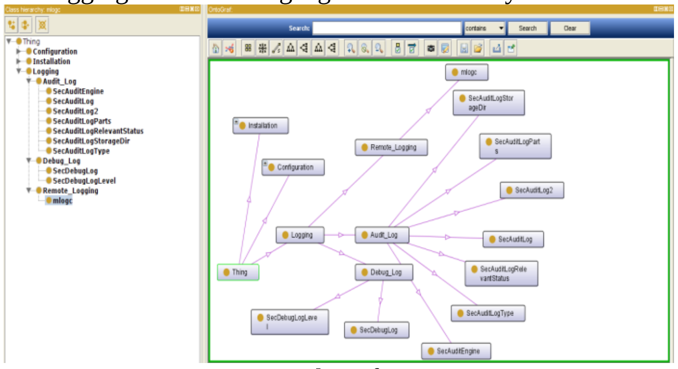


Figure 4: Ontology for Logging

Fig.4 shows ontology for Logging. Logging is a big part of what ModSecurity does. There are mainly three different types of Logging:

- Debug Log: The debug log is going to be your primary troubleshooting tool, especially initially, while you're learning how ModSecurity works.[14]
- Audit Log: When modsecurity detects an event has occurred that it has been instructed to log, it will generate an audit log entry, and if properly configured an audit log event file. The audit log event file is the most useful piece of information the system will collect, so its vital modsecurity be setup correctly to capture this.[14]
- Remote Logging: ModSecurity comes with a tool called mlogc (short for ModSecurity Log Collector), which can be used to transport audit logs in real time to a remote logging server.

In this Logging ontology each component is connected with each other via Is-a relationship.

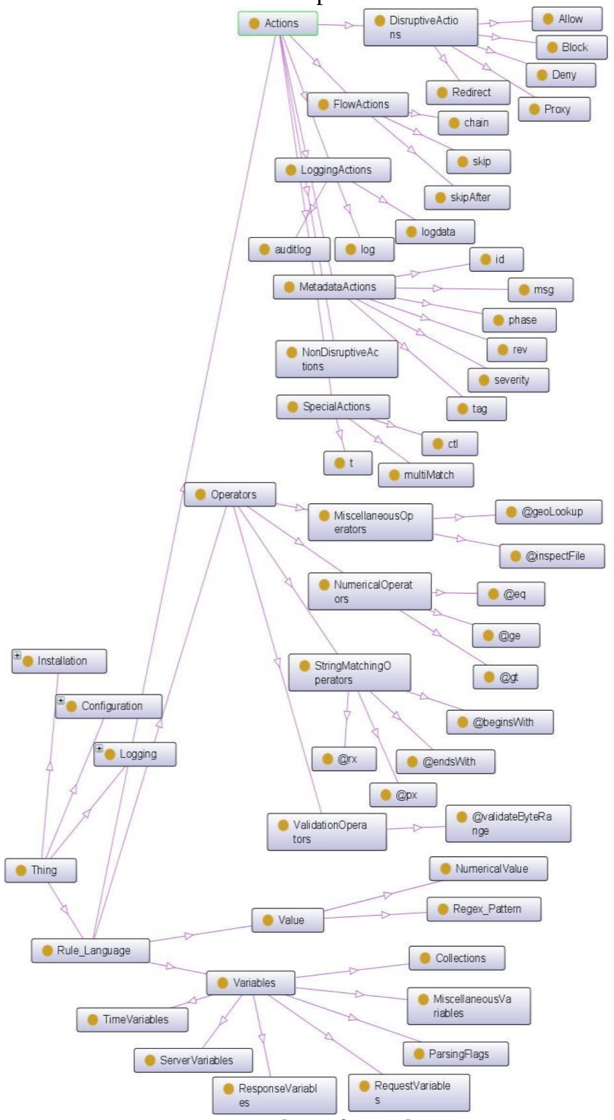


Figure 5: Ontology for Rule Language

Fig.5 shows ontology for Rule Language. It plays an important role to create security policy rules for ModSecurity. Basic rule syntax to create security policy rule for ModSecurity is given below:

SecRule VARIABLES OPERATOR ACTIONS

A. Variables: Tells ModSecurity where to look. Identify parts of a HTTP transaction each rule works with.

- Request Variables
- Server Variables
- Response Variables
- Miscellaneous Variables
- Parsing Flags
- Collections
- Time Variables

B. Operators: Tells ModSecurity how to look. Specify how a (transformed) variable to be analyzed is. Only one operator is allowed per rule.

- String Matching Operators
- Numerical Operators
- Validation Operators
- Miscellaneous Operators

C. Actions: Specify what should be done when a rule matches.

- Disruptive Actions
- Metadata Actions
- Non-Disruptive Actions
- Flow Actions
- Metadata Actions
- Logging Actions
- Special Actions

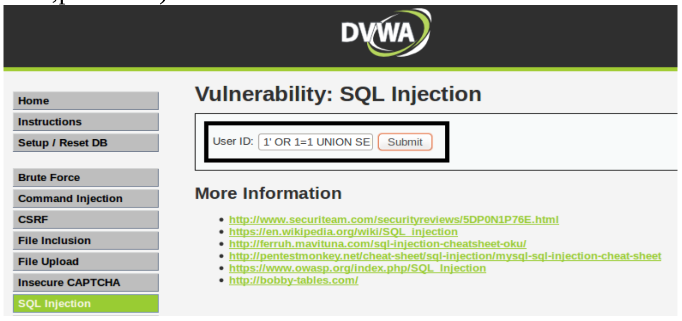
D. Value: Value concept refer to the value against which we are matching our Rule target. Each value concept is linked to the condition individual through hasvalue property.[12]

- Numerical Value
- Regex Pattern

Now we have start to perform SQLI attack on the existing system. We have used Ubuntu linux, Apache web server, Damn Vulnerable Web Application (DVWA) used to perform SQLI, phpMyAdmin is used to easily configuration of DVWA database. Install and configure ModSecurity WAF. Set security level of DVWA to low and start SQLI attack on it. We have also create database of DVWA and configure it in phpMyAdmin.

In existing system, ModSecurity Web Application Firewall is configure in Apache. We have used one magic string for attack and used the same magic string in the proposed architecture so that we will identify the effectiveness of proposed architecture. Magic string which is used for SQLI attack is given below:

1' OR 1=1UNION SELECT null, concat(user, 0x0a,password) FROM users#



Vulnerability: SQL Injection

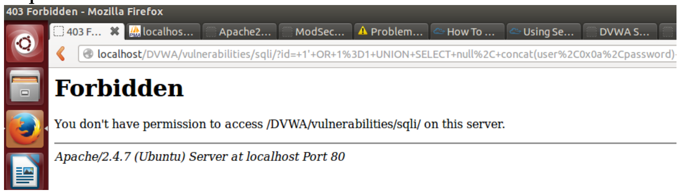
User ID: 1' OR 1=1 UNION SE

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Figure 6: SQLI attack with ModSecurity WAF

When an attacker insert magic string as input then ModSecurity WAF detect SQLI attack by matched data using default rule. Once an attacker click on submit then particular request is drop by Apache. Following fig.7 shows drop request of an attacker.


403 Forbidden - Mozilla Firefox

localhost/DVWA/vulnerabilities/sqli/?id=+1'+OR+1%3D1+UNION+SELECT+null%2C+concat(user%2C0x0a%2Cpassword)

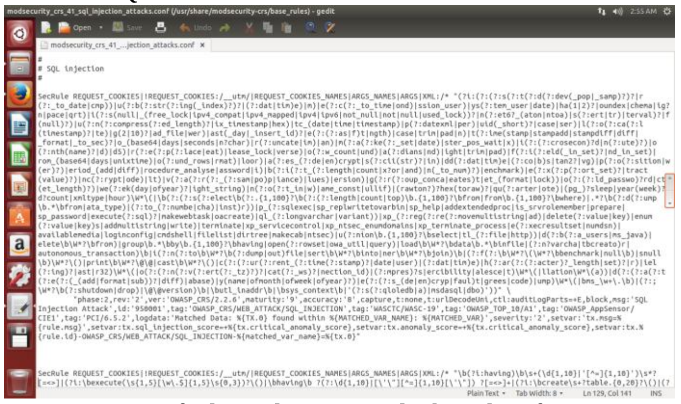
Forbidden

You don't have permission to access /DVWA/vulnerabilities/sqli/ on this server.

Apache/2.4.7 (Ubuntu) Server at localhost Port 80

Figure 7: Prevention against SQLI

We have also monitoring the audit log file of that scenario. Fig.8 shows default security policy rule of ModSecurity to detect an SQLI.


modsecurity_crs_41_sql_injection_attacks.conf (/usr/share/modsecurity-crs/base_rules) - gedit

```
#
# SQL injection
#

SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "(?i:(?:(?:s(?:t(?:d(?:dev(_pop|_samp)?)?|r
(?:_to_date|cmp))|u(?:b(?:str(?:ing(_index)?)?|(?:dat|tim)e)|m)|e(?:c(?:_to_time|ond)|ssion_user)|ys(?:tem_user|date)|ha(1|2)?|oundex|chema|ig?
n|pace|qrt)|i(?:s(null|_(free_lock|ipv4_compat|ipv4_mapped|ipv4|ipv6|not_null|not|null|used_lock))?|n(?:et6?_(aton|ntoa)|s(?:ert|tr)|terval)?|f
(null)?)|u(?:n(?:compress(?:ed_length)?|ix_timestamp|hex)|tc_(date|time|timestamp)|p(?:datexml|per)|uid(_short)?|case|ser)|l(?:o(?:ca(?:l
(timestamp)?|te)|g(2|10)?|ad_file|wer)|ast(_day|_insert_id)?|e(?:(?:as|f)t|ngth)|case|trim|pad|n)|t(?:ime(stamp|stampadd|stampdiff|diff|
_format|_to_sec)?|o_(base64|days|seconds|n?char)|r(?:uncate|im)|an)|m(?:a(?:ke(?:_set|date)|ster_pos_wait|x)|i(?:(?:crosecon)?d|n(?:ute)?)|o
(?:nth(name)?|d)|d5)|r(?:e(?:p(?:lace|eat)|lease_lock|verse)|o(?:w_count|und)|a(?:dians|nd)|ight|trim|pad)|f(?:i(?:eld(_in_set)?|nd_in_set)|
rom_(base64|days|unixtime)|o(?:und_rows|rmat)|loor)|a(?:es_(?:de|en)crypt|s(?:cii(str)?|in)|dd(?:dat|tim)e|(?:co|b)s|tan2?|vg)|p(?:o(?:sition|w
(er)?)|eriod_(add|diff)|rocedure_analyse|assword|i)|b(?:i(?:t_(?:length|count|x?or|and)|n(_to_num)?)|enchmark)|e(?:x(?:p(?:ort_set)?|tract
(value)?)|nc(?:rypt|ode)|lt)|v(?:a(?:r(?:_(?:sam|po)p|iance)|lues)|ersion)|g(?:r(?:oup_conca|eates)t|et_(format|lock))|o(?:(?:ld_passwo)?rd|ct
(et_length)?)|we(?:ek(day|ofyear)?|ight_string)|n(?:o(?:t_in|w)|ame_const|ullif)|(rawton?)?hex(toraw)?|qu(?:arter|ote)|(pg_)?sleep|year(week)?
d?count|xmltype|hour)\W*\(|\b(?:(?:s(?:elect\b(?:.{1,100}?\b(?:(?:length|count|top)\b.{1,100}?\bfrom|from\b.{1,100}?\bwhere)|.*?\b(?:d(?:ump
\b.*\bfrom|ata_type)|(?:to_(?:numbe|cha)|inst)r))|p_(?:sqlexec|sp_replwritetovarbin|sp_help|addextendedproc|is_srvrolemember|prepare|
sp_password|execute(?:sql)?|makewebtask|oacreate)|ql_(?:longvarchar|variant))|xp_(?:reg(?:re(?:movemultistring|ad)|delete(?:value|key)|enum
(?:value|key)s|addmultistring|write)|terminate|xp_servicecontrol|xp_ntsec_enumdomains|xp_terminate_process|e(?:xecresultset|numdsn)|
availablemedia|loginconfig|cmdshell|filelist|dirtree|makecab|ntsec)|u(?:nion\b.{1,100}?\bselect|tl_(?:file|http))|d(?:b(?:a_users|ms_java)|
elete\b\W*?\bfrom)|group\b.*\bby\b.{1,100}?\bhaving|open(?:rowset|owa_util|query)|load\b\W*?\bdata\b.*\binfile|(?:n?varcha|tbcreato)r|
autonomous_transaction)\b|i(?:n(?:to\b\W*?\b(?:dump|out)file|sert\b\W*?\binto|ner\b\W*?\bjoin)\b|(?:f(?:\b\W*?\(\W*?\bbenchmark|null\b)|snull
\b)\W*?\()|print\b\W*?\@\@|cast\b\W*?\()|c(?:(?:ur(?:rent_(?:time(?:stamp)?|date|user)|(?:dat|tim)e)|h(?:ar(?:(?:acter)?_length|set)?|r)|iel
(?:ing)?|ast|r32)\W*\(|o(?:(?:n(?:v(?:ert(?:_tz)?)?|cat(?:_ws)?|nection_id)|(?:mpres)?s|ercibility|alesce|t)\W*\(|llation\W*\(a))|d(?:(?:a(?:t
(?:e(?:(?:_(add|format|sub))?|diff)|abase)|y(name|ofmonth|ofweek|ofyear)?)|e(?:(?:s_(de|en)cryp|faul)t|grees|code)|ump)\W*\(|bms_\w+\.\b)|(?:;
\W*?\b(?:shutdown|drop)|\@\@version)\b|\butl_inaddr\b|\bsys_context\b|'(?:s(?:qloledb|a)|msdasql|dbo)'))" \
        "phase:2,rev:'2',ver:'OWASP_CRS/2.2.6',maturity:'9',accuracy:'8',capture,t:none,t:urlDecodeUni,ctl:auditLogParts=+E,block,msg:'SQL
Injection Attack',id:'950001',tag:'OWASP_CRS/WEB_ATTACK/SQL_INJECTION',tag:'WASCTC/WASC-19',tag:'OWASP_TOP_10/A1',tag:'OWASP_AppSensor/
CIE1',tag:'PCI/6.5.2',logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}: %{MATCHED_VAR}',severity:'2',setvar:'tx.msg=%
{rule.msg}',setvar:tx.sql_injection_score=+%{tx.critical_anomaly_score},setvar:tx.anomaly_score=+%{tx.critical_anomaly_score},setvar:tx.%
{rule.id}-OWASP_CRS/WEB_ATTACK/SQL_INJECTION-%{matched_var_name}=%{tx.0}"


SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "\b(?i:having)\b\s+(\d{1,10}|'[^=]{1,10}')\s*?
[=<>]|(?i:\bexecute(\s{1,5}[\w\.$]{1,5}\s{0,3})?\()|\bhaving\b ?(?:\d{1,10}|[\'\"][^=]{1,10}[\'\"]) ?[=<>]+|(?i:\bcreate\s+?table.{0,20}?\()|(?
```

Plain Text Tab Width: 8 Ln 129, Col 141 INS

Figure 8: Default ModSecurity rule that identify SQLI

Now we will move towards second scenario, we have created an SQLI attack ontology. Following fig.9 shows SQLI attack ontology:

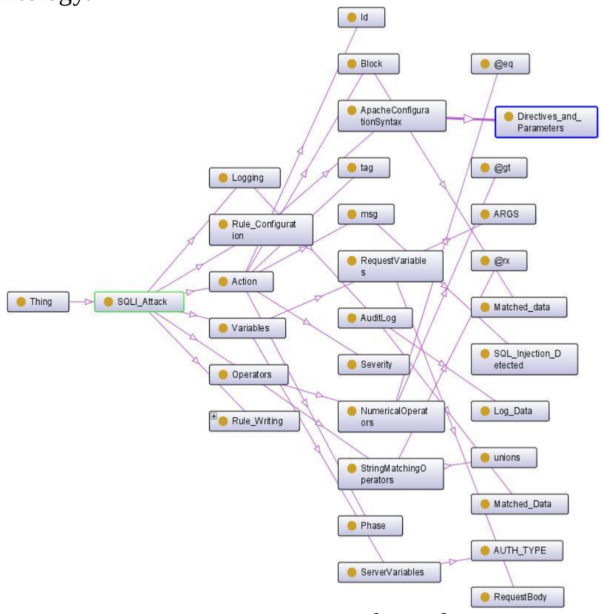


Figure 9: SQLI attack ontology

As shown in fig.8 we can see that how default security policy rule of ModSecurity is difficult to understand. So we have improved default security policy rule by modifying it using ontological knowledgebase and SQLI attack ontology. We have written custom security policy rule to detect SQLI attack in the following directory of Apache:

/usr/share/modsecurity-crs/base_rules/custom_rule.conf

Custom rule that we had written for ModSecurity to detect SQLI is given below:

SecRule“ARGS|!REQUEST_COOKIES:|REQUEST_COOKIES_NAMES | ARGS_NAMES |XML: /*”(?: select|union|having) \s*?[^\s]) (?: union select@)/ (?: union[\w(\s]*? select) “Phase:2, capture, block, tag: msg: SQL Injection Attack Detected logdata: ‘Matcheddata : % {TX.0} found within % {MATCHED_VAR_NAME} : % {MATCHED_VAR}’

We have completely followed the basic syntax to create our custom rule. We have written this security policy rule in custom_rule.config file and put the default security policy rule in comment. Once we completed this task we have enabled our custom rule in respective directory. We have written this security policy rule in custom_rule.config file and put the default security policy rule in comment. After written custom rule we enabled our custom rule in respective directory.

After enabling custom security policy rule to detect SQLI we have performed same SQLI attack by using same magic string as we used in first scenario. Fig. 10 shows same string as input for SQLI:

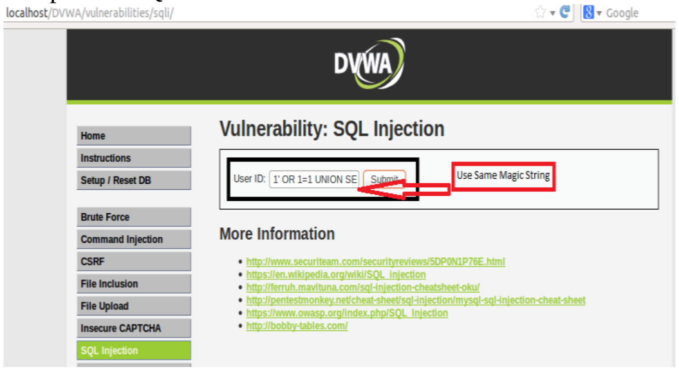
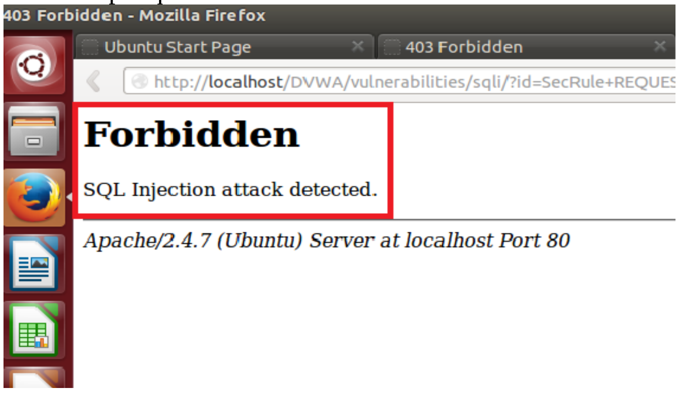

localhost/DVWA/vulnerabilities/sqli/
Google
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
Vulnerability: SQL Injection
User ID: 1' OR 1=1 UNION SE
Submit
Use Same Magic String
More Information
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
https://en.wikipedia.org/wiki/SQL_injection
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
https://www.owasp.org/index.php/SQL_Injection
http://bobby-tables.com/

Figure 10: Same string as input for SQLI

When an attacker insert same magic string as input then ModSecurity WAF detect SQLI attack by signature matched data using our custom rule. Once an attacker click on submit then particular request is drop by Apache. Following fig.11 shows drop request of an attacker:


403 Forbidden - Mozilla Firefox

Ubuntu Start Page | 403 Forbidden

http://localhost/DVWA/vulnerabilities/sqli/?id=SecRule+REQUES

Forbidden

SQL Injection attack detected.

Apache/2.4.7 (Ubuntu) Server at localhost Port 80

Figure 11: Prevention against SQLI scenario 2

We have also monitoring the audit log file of that scenario 2.

V. TEST OUTCOMES

In first scenario i.e. In Existing system, we have performed SQLI attack on DVWA with ModSecurity Web Application Firewall using same magic string so that we will compare the results of that particular attack. There is no modification in default security rule of ModSecurity Web Application Firewall that detect and prevent SQLI. We have install & configure ModSecurity Web Application Firewall in Apache server. As a test result we came to know that ModSecurity WAF has default rule set against different attacks so in this scenario an SQLI attack was detected by matched data and HTTP request is blocked by Apache. This second scenario is the existing system in which default security rules are applied for particular attack. Default security policy rule of ModSecurity for SQLI is very tough to understand and also difficult to write. If any mistakes are happen at the time of rule to identify it.

In second scenario i.e. In Proposed architecture, to overcome different issues regarding to security policy rule we have created Ontological Knowledgebase of ModSecurity. We have performed same attack as performed in scenario 2 i.e. SQLI attack using same magic string. We have also created one specific SQLI attack ontology. Using ModSecurity Ontological Knowledgebase and SQLI attack ontology we have created custom security policy rule of ModSecurity WAF for SQLI attack. We have written our custom security policy rule in custom_rule.config file in Apache. Once we have written our custom security policy rule we put default security policy rule of ModSecurity in comment to get the result of our custom rule. As a result we came to know that our custom security policy rule works and detects SQLI attack based on signature matched data. HTTP request is also blocked by Apache. Timeout for this result is fast as compare to the existing system. We have also modify the default ModSecurity security policy rules and made it as simple as possible.

VI. CONCLUSION AND FUTURE WORK

We have proposed an approach of using ontology modeling to represent WAF configuration knowledge for the administrator. Our ontological knowledgebase of ModSecurity effectively represents the semantic knowledge behind ModSecurity web application firewall configurations for the administrator in writing, editing and updating WAF configuration and helps him in inferring high level security policies from firewall configurations. Our ontological knowledgebase can also help administrator in identifying redundant, partially overlapping rules and helps in generating efficient rule sets for WAFs. We have create specific attack ontology for SQLI attack. Using ModSecurity ontological knowledgebase and specific ontology for SQLI we have modify default rule of ModSecurity and made one custom security policy rule for ModSecurity WAF to prevent our Apache server from specific one attack i.e. SQLI. On the basis of our proposed work we conclude that ontology is kind of new concept and by using ontological knowledge base administrator can be easily write security policy rules to protect our main server from different web based attacks. Our custom security policy rule is work and it can be easily understand.

In future, we can also create different custom security policy rule for specific attack by creating different specific attack ontology and provide security for our server or web application against different web based attacks. We can also try to create one tool in which we have set some default parameters for security policy rule generation when we provide input different parameters then as an output tool generate the custom rule according to the input parameters but it is very tough task.

REFERENCES

[1] G. Avramescu, M. Bucicoiu, D. Rosner, N.Ţăpuş, “Guidelines for Discovering and Improving Application Security”, 2013 19th International Conference on Control Systems and Computer Science
[2] A.Pramod , A.Ghosh, A. Mohan, M.Shrivastava and Dr. R. Shettar, “SQLI Detection System for a safer Web Application”, 2015 IEEE International Advance Computing - Conference (IACC)
[3] Li Li, Q. Dong, L. Zhu, and D. Liu, “The Application of Fuzzing in Web software security vulnerabilities Test”, 2013 International Conference on Information Technology and Applications.
[4] Y.Takamatsu, Y.Kosuga, and K.Kono, “Automated Detection of Session Manage-ement Vulnerabilities in Web Applications”, 2012 Tenth Annual International Conference on Privacy, Security and Trust.
[5] Emhimed Alatrash "Using Web Tools for Constructing an Ontology of Different Natural Languages" A Ph. D. Dissertation submitted to The Dept. of Computer Science Faculty of Mathematics University of Belgrade.
[6] Rajesh M. Lomte, Prof. S. A. Bhura Computer Science & Engineering Department, BNCOE, India. "Survey of different Web Application Attacks & Its Preventive Measures" IOSR Journal of Computer Engineering (IOSR-JCE)e-ISSN: 2278-0661, p-ISSN: 2278-8727Volume 14, Issue 5 (Sep. - Oct. 2013), PP 46-51 www.iosrjournals.org
[7] https://www.owasp.org
[8] https://en.wikipedia.org/wiki/Ontology_language
[9] Ivan Ristic, ModSecurity Handbook 'The Complete Guide to Securing Your Web Applications', Feisty Duck, Jan 2010. Available: ModSecurity Handbook PDF.
[10] Magnus Mischel, ModSecurity 2.5 'Securing your Apache installation and web applications', Birmingham, B27 6PA, UK. Packet Publishing, Nov 2009.
[11] https://www.modsecurity.org/rules.html
[12] Ali Ahmad, Zahid Anwar, Ali Hur and Hafiz Farooq Ahmad, "Formal Reasoning of Web

Application Firewall Rules through Ontological Modeling" School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Islamabad, Pakistan.
[13] OWASP, “ModSecurity Core Rule Set (CRS),” 2012.
[14] http://resources.infosecinstitute.com/analyzing-mod-security-logs/
[15] www.acunetix.com/websitesecurity/sql-injection/