Proceeding
ISSN 2723-4509 (Online)
International Conference
on Social, Sciences and Information Technology
Kisaran, August 19th, 2020, page. 187 - 194
DOI: https://doi.org/10.33330/icossit.v1i1.711
Available online at https://jurnal.stmikroyal.ac.id/index.php/ICdoSSIT

INTRUSION DETECTION SYSTEM AND MODSECURITY FOR HANDLING SQL INJECTION ATTACKS

Ruri Ashari Dalimunthe[1*], Sahren [2]
[1]Computer engineering, Sekolah Tinggi Manajemen Informatika dan Komputer Royal, Indonesia
[2]Computer System, Sekolah Tinggi Manajemen Informatika dan Komputer Royal, Indonesia

Corresponding author:
ruriazharidalimunthe@royal.ac.id

Keywords:
Database; Instrusion Detection System; ModSecurity; SQL Injection; Sqlmapproject

ABSTRACT

SQL Injection (SQLI) is the main type of attack that will threaten the integrity, confidentiality, and authenticity or functionality of database-based web applications. This allows an attacker to gain unauthorized access to a back-end database by exploiting vulnerabilities in the system to carry out attacks and access existing resources. Therefore, the best prevention techniques against SQL Injection attacks are needed to protect an individual or organizational data from hacking. In this study, using two security techniques, namely using the Intrusion Detection System as a sensor that will detect if an SQL Injection attack occurs, and using a web-based firewall (ModSecurity) as a security system that will block attacks. The purpose of this research is to build a capable security system that will detect and block any SQL Injection attacks against the database. the proposed system was tested using the Sqlmapproject attack tool. Sqlmapproject is used to attack web applications before and after protection. The results show that the proposed security system is functioning properly and can protect the database system on the web well, high performance, and efficiency.

INTRODUCTION

Security becomes one of the important points that we needed to be considered, the system was connected to a public network or the internet becomes very crucial [1]. At the moment, human beings needed most dependent on the presence of information or data, especially for information or digital data. The greater the need for information, the more incidents or security breaches of the existing information system. Important information or data can be used by parties who are not responsible for their own benefits [2]. All modern applications mostly use a centralized database to convey information. Injection attacks occur when someone deliberately uses unauthorized channels to send malicious SQL commands to the database server [3]. Where the channel is widely used as a gap, the input data channel is forgotten to be validated.

Proceeding
International Conference
on Social, Sciences and Information Technology
Kisaran, August 19th, 2020, page. 187 - 194
DOI: https://doi.org/10.33330/icossit.v1i1.711
Available online at https://jurnal.stmikroyal.ac.id/index.php/ICdoSSIT
ISSN 2723-4509 (Online)

The SQL Injection attack is one of the vulnerabilities in user input that is not validated and is also the most commonly used application to attack a web-based information system. SQL injection was basically an SQL command that was injected into an SQL statement via an input field that was not validated or protected [3]. SQL injections basically have been done several types including SQL injection vulnerability with SQL injection Attack [4]. SQL injection can also be categorized with classic SQL injection and modern SQL injection, this was developed in accordance with the development of query security in every database storage development such as MAMP and XAMP [4].

SQL Injection attacks on web-based information systems can actually be detected using the Intrusion Detection System (IDS). An intrusion detection system (IDS) can be defined as tools, methods, resources that provide assistance to identify and report on computer network activities [5]. In IDS itself there are two types of detection methods, namely Rule-base Detection and Anomaly Detection detection [6].

This research will try to detect SQL Injection attacks by using the Rule Base Detection method on IDS. Where with this method a separate rule will be made which will be used to detect when an SQL Injection attack will occur and then record it in the system log. In addition to using IDS Rule base Detection in this study, another method will be used, namely using the Web Application Firewall to handle SQL Injection attacks on the system.

Web Application Firewall is a process for securing a web. This process is in the form of a mechanism that works to prevent access and modification by unknown users, to data from the web stored online [7],[8]. For Web Application Firewall, this research will use ModSecurity. ModSecurity itself is a Web Application Firewall module that is open source and cross-platform. Mod Security can filter incoming and outgoing data to stop dangerous traffic. In addition, ModSecurity can analyze and detect dangerous traffic [9],[10].

Several previous studies have been carried out relating to SQL Injection Attack, Intrusion Detection and Web Application Firewall. Here the authors take several sources as a reference and comparison to build this research. [6], Conducting research on the implementation of a SQL Injection attack detection system using Algortima K-Nearest Neighbor. [3] conducted research on analyzing SQL Injection attacks using SQLMAP tools. [5] made a research on the development of the Intrusion Detection System against SQL Injection attacks using the Learning Vector Quantization Method. [7], conducted research by implementing a Web Application Firewall on the Web Mytra Dasboard with the ModSecurity Web Application Firewall module. [11], conducted research on the use of the Intrusion Detection System Base Query Singnatures on online databases. [12], conducted research on security systems in cloud computing using two security methods namely Intrusion Detection System (IDS), and Instrusion Prevention System (IPS).

This study aims to build a security system with the Intrusion Detection System and ModSecurity Web Application Firewall as an information system protector from the SQL Injection attack by a hacker.

ISSN 2723-4509 (Online)
DOI: https://doi.org/10.33330/icossit.v1i1.711
Available online at https://jurnal.stmikroyal.ac.id/index.php/ICdoSSIT

METHOD

The security method employed for the Web Server security system from the SQL Atack attack is done in stages, the first is to design the server as a Web Server and Database Server. The second stage is the configuration of the Intrusion Detection System (IDS) and the Web Appication Firewall ModSecurity. The third stage creates a SQL Injection attack scenario on the server. Each of these stages the authors describe in the form of topology in Image 1 below:

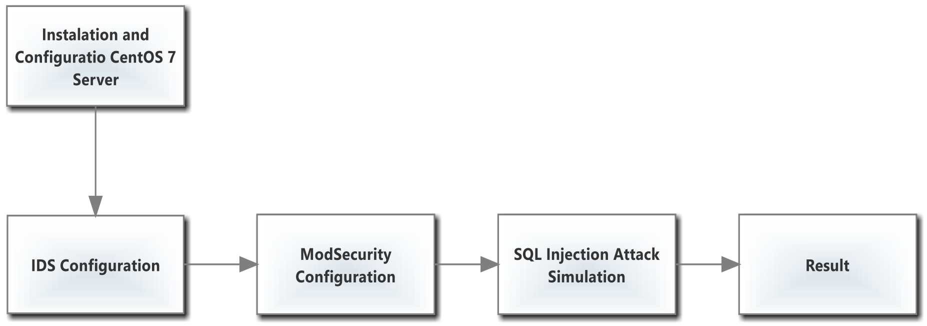

Instalation and Configuratio CentOS 7 Server
IDS Configuration
ModSecurity Configuration
SQL Injection Attack Simulation
Result

Image 1. Workflow Stages

In the first stage, the server that is used to install is CentOS 7, which will be used as a web server and database server, the server will be run virtually. This server will later become the target of an attack with SQL Injection and then try to be protected against these attacks.

In the second service, install Linux-based tools (applications), namely snort, barnyard, BASE and firewall. Next, configure the snort in the snort.conf file related to several related snort variables including Ipvar HOME_NET, Ipvar EXTERNAL_NET, RULE PATH, SO_RULE_PATH, PREPROC_RULE_PATH, making snort inline type afpacket (config daq_dir, config daq_mode, output) snort is related to snort.log, and the creation of rules for snort is done in the local.rules file in the `/ etc / snort / rules / directory`.

Third Stage. Salain IDS here the author also uses a Web Application Firewall with the ModSecurity module. The concept behind WAF is very similar to the concept of a traditional firewal work, where a firewall works based on a set of rules that are configured on a firewall or also commonly called a rule. The WAF rules are specifically designed to filter network traffic using the http protocol. This rule is also able to detect common attacks, such as probes (get preliminary info before carrying out attacks) from SQL Injection Attack. Some ModSecurity features that can be used in the security system built in this research are log checking, access to every part of the request addressed to the server and responding to the inspection results, having rules based on regular regular expressions that are flexible, checking uploaded files, validation in real time and also buffer overflow protection. ModSecurity itself can be used in two modes namely embedded mode by adding ModSecuriy as apache server module and Network Gateway Mode, in this mode all web traffic passes through a proxy where ModSecuriy will be installed as a reverse proxy.

In the Fourth stage, there will be a SQL Injection attack on the server, this trial will be carried out twice, namely when the server has not been configured security and after the server is given security with IDS and ModSecurity. SQL Injection is a way of exploiting security holes that appear at the level or "layer" of the database and its applications. The security hole is shown in the form of a response from incorrect data requests to the server that were intentionally sent by the attacker.

Examples of vulnerability loopholes that are often victims of SQL Injection are:

1. Control, control, or filter characters are not well defined and correctly (Incorrectly Filtered Escape Characters),
2. Incorrect type of selection and handling of variables and program parameters (Incorrect Type Handling),
3. Vulnerabilities Inside the Database Server)
4. SQL Injection; etc.

One of the SQL Injection Blind Sql Injection techniques is an attack technique that is carried out by entering the syntax or Sql command on a web that has a vulnerability to attack, to find out the contents of the database from the web. The following example script SQL injection attack program can be seen in Image 2 below.

```
localhost/index.php?id=-9union
select
1,group_concat(table_name),3,4,5
command from
information_schema.tables where
table_schema=database()--
```

Image 2. Example of SQL Injection Script

RESULT AND DISCUSSION

In this study a security system and reporting instructions for security or interference from attacks on the Web Server will be made, using the Instrusion Detection System and ModSecurity. The attack that will be detected and blocked is the SQL Injection attack. Experiments using one computer as a server computer, with IP Address 192.168.56.102, one computer as penetration testing with IP Address 192.168.100.101. The topology can be seen in Image 3, the attack will be carried out on a server that is still in a state without security against various security problems SQL Injection attacks that will use SQLMap tools that are run through the command line interface with the command:

```
#python sqlmap.py -url http://192.168.56.102/read.php?id=26 -dbs
```

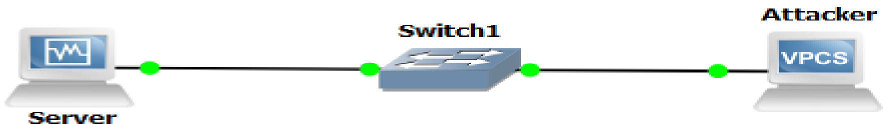


Image 3. Initial Topology

ISSN 2723-4509 (Online)
DOI: https://doi.org/10.33330/icossit.v1i1.711
Available online at https://jurnal.stmikroyal.ac.id/index.php/ICdoSSIT

Image 3 shows that the connection between penetration testing or attacker and the server computer is done directly without any security wall being passed first. This results in no filtering of the data traffic that passes through. Whereas in Image 4 we can see that the relationship between penetration testing or attackers with server computers has been limited by the ModSecurity Web Application Firewall and will be monitored by the Intrusion Detection System (IDS) so that incoming data traffic will be checked or filtered first and a report will be made of the indication of the attack.

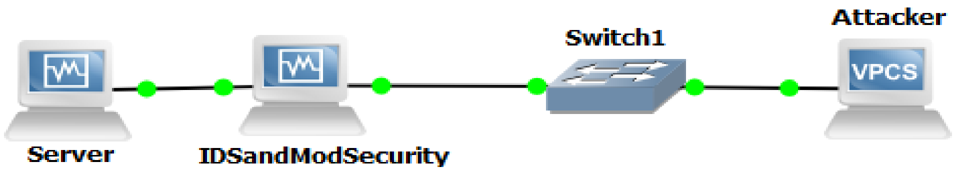
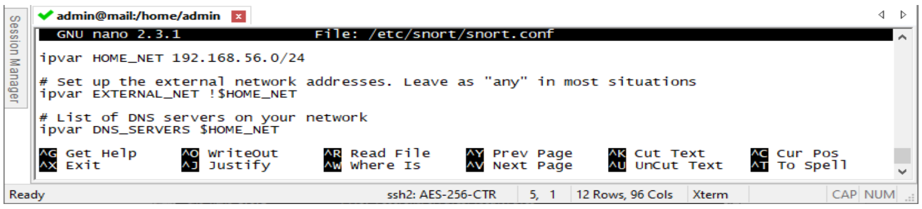


Image 4. Final Topology

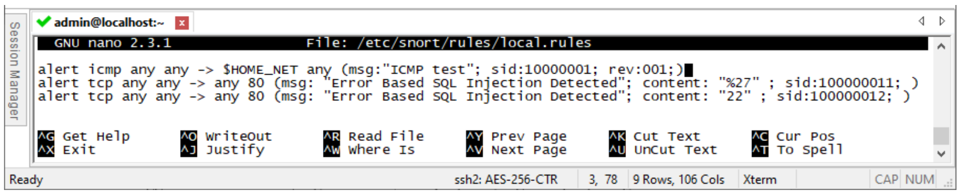
To carry out defense and detection mechanisms from SQL Injection attacks, several components and configurations are required. First here will be prepared IDS detection system with Snort, where later this system will detect if there is a suspicious traffic anomaly on the system in accordance with the rules or rules that have been made previously. In the following Image 4, the main configuration of IDS will be displayed.



```
  GNU nano 2.3.1                File: /etc/snort/snort.conf

ipvar HOME_NET 192.168.56.0/24

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET !$HOME_NET

# List of DNS servers on your network
ipvar DNS_SERVERS $HOME_NET

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Image 5. IDS Main Configuration

The Intrusion Detection System (IDS) used works based on the rules that have been used. In this study there are two kinds of rules, the first is the rule that has been provided by the community rule snort and placed in the `/etc/snort/rules/community.rules` directory on the system. In addition, we can also use local rules that we make ourselves, as shown in Image 5 below.



```
  GNU nano 2.3.1              File: /etc/snort/rules/local.rules

alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:10000001; rev:001;)
alert tcp any any -> any 80 (msg: "Error Based SQL Injection Detected"; content: "%27" ; sid:100000011; )
alert tcp any any -> any 80 (msg: "Error Based SQL Injection Detected"; content: "22" ; sid:100000012; )

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Image 6. Rule IDS

After the intrusion detection system is finished, the second security system, Web Application Firewall ModSecurity, will be prepared. There are several configuration files related to ModSecurity, including:

DOI: https://doi.org/10.33330/icossit.v1i1.711
Available online at https://jurnal.stmikroyal.ac.id/index.php/ICdoSSIT

a) Main Configuration File
Is the main configuration file for the mod_security apache module.
```
# nano /etc/httpd/conf.d/mod_security.conf
```
b) Configuration Directory
Contains all other configuration files for Apache mod_security.
```
# cd /etc/httpd/modsecurity.d/
```
c) Core Rule Set (CRS) configuration
The configurations contained in this file must be adjusted for specific requirements before implementation.
```
# nano /etc/httpd/modsecurity.d/crs-setup.conf
```
d) Debugging ModSecurity Rules To debug mod_security rules and other problems.
```
# /var/log/httpd/modsec_debug.log
```
e) ModSecurity Log File
All requests that trigger ModSecurity events (as detected) or server errors are logged ("RelevantOnly") enter this file.
```
# /var/log/httpd/modsec_audit.log
```
In Image 6 the following shows the results of initial testing of the security system on the web server to cope with SQL injection attacks.

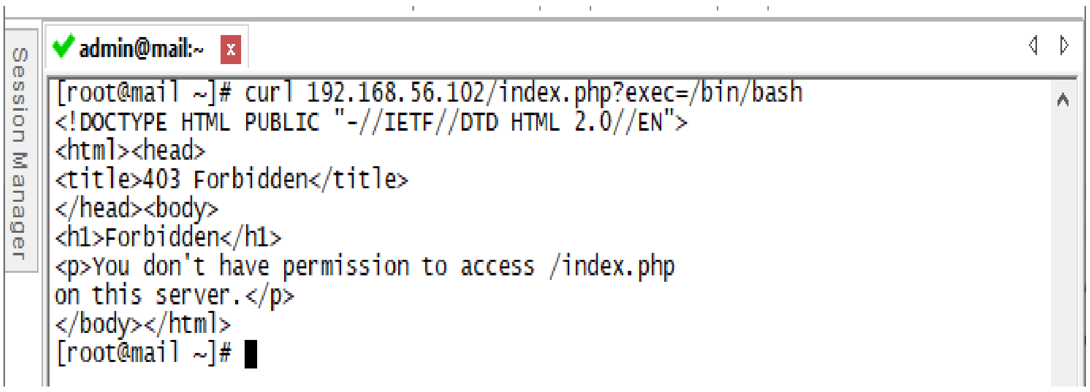
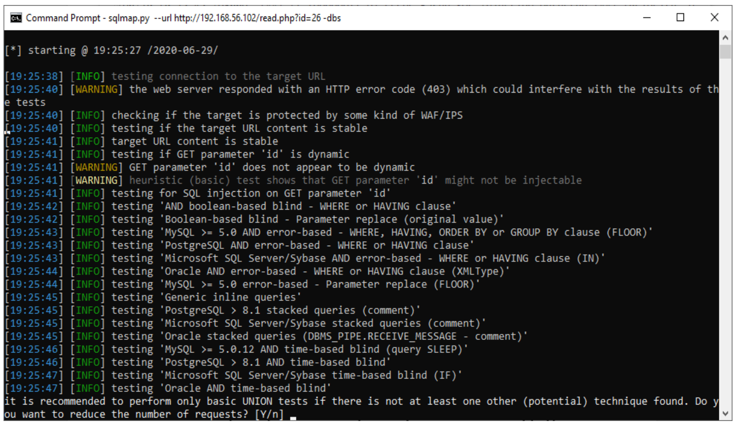

```
[root@mail ~]# curl 192.168.56.102/index.php?exec=/bin/bash
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access /index.php
on this server.</p>
</body></html>
[root@mail ~]#
```

Image 7. ModSecurity Test Results

After all the Intrusion Detection System (IDS) Configuration and ModSecurity configurations have been completed, the final test can then be performed to obtain the final results. Is the webserver can still be attacked with SQL Injection and whether the attack detection system on IDS works well. In Image 7, the injection process is directed at the server.

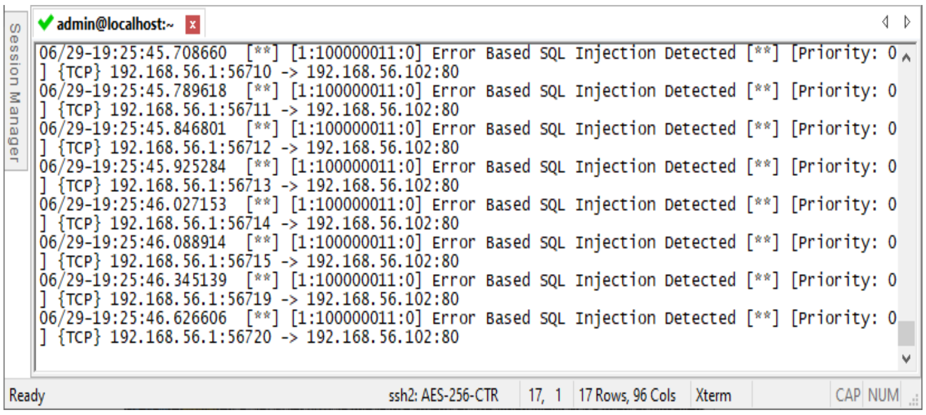
ISSN 2723-4509 (Online)
DOI: https://doi.org/10.33330/icossit.v1i1.711
Available online at https://jurnal.stmikroyal.ac.id/index.php/ICdoSSIT



```
Command Prompt - sqlmap.py  --url http://192.168.56.102/read.php?id=26 -dbs

[*] starting @ 19:25:27 /2020-06-29/

[19:25:38] [INFO] testing connection to the target URL
[19:25:40] [WARNING] the web server responded with an HTTP error code (403) which could interfere with the results of th
e tests
[19:25:40] [INFO] checking if the target is protected by some kind of WAF/IPS
[19:25:40] [INFO] testing if the target URL content is stable
[19:25:41] [INFO] target URL content is stable
[19:25:41] [INFO] testing if GET parameter 'id' is dynamic
[19:25:41] [WARNING] GET parameter 'id' does not appear to be dynamic
[19:25:41] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[19:25:41] [INFO] testing for SQL injection on GET parameter 'id'
[19:25:42] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[19:25:42] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[19:25:43] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[19:25:43] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[19:25:43] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[19:25:44] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[19:25:44] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[19:25:45] [INFO] testing 'Generic inline queries'
[19:25:45] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[19:25:45] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[19:25:45] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[19:25:46] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[19:25:46] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[19:25:47] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[19:25:47] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do y
ou want to reduce the number of requests? [Y/n]
```

Image 8. SQL Injection Attack

In Image 7 it can be seen that the connection directed to the webserver with injection activity has been dropped. On the results of the attack, we can also see that the SQL injection attack failed to be performed on all parameters tested. For detection of SQL Injection activities that lead to the server can be seen in Image 8 below.



```
06/29-19:25:45.708660  [**] [1:100000011:0] Error Based SQL Injection Detected [**] [Priority: 0
] {TCP} 192.168.56.1:56710 -> 192.168.56.102:80
06/29-19:25:45.789618  [**] [1:100000011:0] Error Based SQL Injection Detected [**] [Priority: 0
] {TCP} 192.168.56.1:56711 -> 192.168.56.102:80
06/29-19:25:45.846801  [**] [1:100000011:0] Error Based SQL Injection Detected [**] [Priority: 0
] {TCP} 192.168.56.1:56712 -> 192.168.56.102:80
06/29-19:25:45.925284  [**] [1:100000011:0] Error Based SQL Injection Detected [**] [Priority: 0
] {TCP} 192.168.56.1:56713 -> 192.168.56.102:80
06/29-19:25:46.027153  [**] [1:100000011:0] Error Based SQL Injection Detected [**] [Priority: 0
] {TCP} 192.168.56.1:56714 -> 192.168.56.102:80
06/29-19:25:46.088914  [**] [1:100000011:0] Error Based SQL Injection Detected [**] [Priority: 0
] {TCP} 192.168.56.1:56715 -> 192.168.56.102:80
06/29-19:25:46.345139  [**] [1:100000011:0] Error Based SQL Injection Detected [**] [Priority: 0
] {TCP} 192.168.56.1:56719 -> 192.168.56.102:80
06/29-19:25:46.626606  [**] [1:100000011:0] Error Based SQL Injection Detected [**] [Priority: 0
] {TCP} 192.168.56.1:56720 -> 192.168.56.102:80
```

Image 9. Detection of SQL Injection Attacks on IDS

In Image 8 it can be seen that the security system for detecting SQL Injection attacks with the Intrusion Detection System is working. Marked by the appearance of the report with the information Based SQL Injection Detected. In existing reports, we can also know the attacking IPAddress and the attacking port whether port 80 or 443. This system works in real-time meaning the system will automatically detect when the attack starts and will stop when the attack stops.

CONCLUSION

Based on trials conducted in this study, it concluded that the Intrusion Detection System (IDS) and Web Application Firewall (WAF) ModSecurity managed to detect and prevent SQL Injection Attack on the Web Server.

Proceeding
International Conference
on Social, Sciences and Information Technology
Kisaran, August 19th, 2020, page. 187 - 194
DOI: https://doi.org/10.33330/icossit.v1i1.711
Available online at https://jurnal.stmikroyal.ac.id/index.php/ICdoSSIT

ISSN 2723-4509 (Online)

BIBLIOGRAPHY

[1] S. Sahren, R. A. Dalimuthe, and M. Amin, “Penetration Testing Untuk Deteksi Vulnerability Sistem Informasi Kampus,” *Pros. Semin. Nas. Ris. Inf. Sci.*, vol. 1, no. September, p. 994, 2019, doi: 10.30645/senaris.v1i0.109.

[2] Sutarti, P. Pancaro, Adi, and I. Saputra, Fembi, “Implementasi IDS (Intrusion Detection System) Pada Sistem Keamanan Jaringan SMAN 1 Cikeusal,” *J. PROSISKO*, vol. 5, no. 1, pp. 1–8, 2018.

[3] S. Lika, R. Dwi, P. Halim, and I. Verdian, “1 , 2 , 3,” vol. 4, no. 2, pp. 88–94, 2018.

[4] I. Print, “Analisis Forensik Serangan SQL Injection dan DoS Menggunakan Instrution Detection System Pada Server Berbasis Lokal,” *Inform. Mulawarman J. Ilm. Ilmu Komput.*, vol. 2, pp. 0–4, 2020.

[5] A. S. Irawan, E. S. Pramukantoro, and A. Kusyanti, “Pengembangan Intrusion Detection System Terhadap SQL Injection Menggunakan Metode Learning Vector Quantization,” *J. Pengemb. Teknol. Inf. dan Ilmu Komput. Univ. Brawijaya*, vol. 2, no. 6, pp. 2295–2301, 2018.

[6] R. D. B, “Implementasi Sistem Pendeteksi Serangan SQL Injection dengan Menggunakan Algoritme K-Nearest Neighbor,” vol. 3, no. 12, pp. 10984–10992, 2019.

[7] R. Nurachmad Syaefuddin, “Implementasi Web Application Firewall pada Web Mytra Dashboard dengan Menggunakan Modul ModSecurity,” no. April, 2018, doi: 10.13140/RG.2.2.15824.00006.

[8] I. A. Yari, B. Abdullahi, and S. A. Adeshina, “Towards a framework of configuring and evaluating modsecurity WAF on tomcat and apache web servers,” *2019 15th Int. Conf. Electron. Comput. Comput. ICECCO 2019*, no. Icecco, 2019, doi: 10.1109/ICECCO48375.2019.9043209.

[9] T. Jain and N. Jain, “Framework for Web Application Vulnerability Discovery and Mitigation by Customizing Rules Through ModSecurity,” *2019 6th Int. Conf. Signal Process. Integr. Networks, SPIN 2019*, pp. 643–648, 2019, doi: 10.1109/SPIN.2019.8711673.

[10] J. J. Singh, H. Samuel, and P. Zavarsky, “Impact of paranoia levels on the effectiveness of the modsecurity web application firewall,” *Proc. - 2018 1st Int. Conf. Data Intell. Secur. ICDIS 2018*, pp. 141–144, 2018, doi: 10.1109/ICDIS.2018.00030.

[11] A. Jumaa and A. Omar, “Online Database Intrusion Detection System Based on Query Signatures,” *J. Univ. Hum. Dev.*, vol. 3, no. 1, pp. 282–287, 2017, doi: 10.21928/juhd.20170315.14.

[12] S. Khadafi, B. D. Meilani, and S. Arifin, “Sistem Keamanan Open Cloud Computing Menggunakan Ids (Intrusion Detection System) Dan Ips (Intrusion Prevention System),” *J. IPTEK*, vol. 21, no. 2, p. 67, 2017, doi: 10.31284/j.iptek.2017.v21i2.207.