Self-Protecting Technology for Web Applications

Ms. Swathi R

Researcher, Bengaluru, Karnataka, India

---***---

Abstract - *Self-defense for running applications is a security-based technology. It keeps an eye on the running program, which keeps an eye on incoming traffic to see whether any incoming attacks are present. If they are, it blocks them by using the information from the current application. By evaluating the inputs and preventing the inputs that could enable assaults in the application, it is claimed to enhance the security aspects of software. RASP reduces the reliance of applications on external hardware, such as firewalls, for runtime security protection. We outline the fundamental ideas behind Runtime Application Self-Protection Technology (RASP), a relatively new security strategy that is soon to be widely adopted.*

Key Words: Self Protection, Security, Network Traffic.

1. INTRODUCTION

Networks are expanding in size, becoming more complicated, and becoming harder to secure. In order to better detect risks, security providers are incorporating new methodologies into their software products, such as studying users' typical routines. Applications are scattered and complicated, thus application security strategies should be divided up to allow for a more in-depth investigation of the system's data. Attacks are becoming more sophisticated, and present access safeguards are no longer able to stop them. Due to shortcomings in application security, production applications are at danger. Once they take place on a network, security issues have very little visibility.

There's a strong need of module that can:

• By keeping an eye on and analysing exploitation attempts, you can spot attack trends and prevent callers from accessing the application. Context-aware detection.

• Recognize and record how an application performs, interacts with its ecosystem, and flows data in and out.

• Identify unusual searches, application inputs, feature usages, and compare to the database.

• Recognize typical application usage patterns and document them in the database.

If something is strange, deny access to the features.

• When an application has known vulnerabilities, defend against assaults without the requirement for development fixes.

1.1 EXISTING SYSTEM

RASP is a security technology that is built or linked into an application or application runtime environment and is capable of controlling application execution as well as detecting and preventing real-time attacks [1].

RASP prevents attacks by self-protecting or automatically reconfiguring in response to particular network situations without human intervention (threats, faults, etc.).

When an application is executed (runtime), RASP becomes active, causing the programme to supervise itself and identify fraudulent input and behavior. RASP processes both the application's behavior and its context in real time.

As a result, regular security analysis is used, with the system responding appropriately to any detected attacks. [2].

Web applications are a common target, and attackers commonly employ them to gain access to a system (network)[3].

Maintaining network security while preserving the flexibility needed by web application developers is the primary security goal of a business. The idea that a web application firewall (WAF) is the best security measure is one that is frequently expressed. Before running the web application, WAF filtering and code testing methods are implemented, evaluating incoming traffic flow for known attack patterns and blocking inputs from reaching the application. The consensus is that a WAF's signature base and pattern matching engine determine how trustworthy it is.

2. PROPOSED SYSTEM

There are two used cases involved:

I. **The Self Protecting tool placed inside the application**

User attempting to log in while still logged in: This user action can be viewed as suspicious, and the user is prompted to log out first to preserve the integrity of the programme. Features of the Proposed System's Security.

a. When a user logs in or out, the log files are printed with their timestamp and distinctive ID (such as their account number).

b. A Boolean value is always set to true whenever a log-in action is carried out. If the user logs out, the value just becomes false.

c. Users who are already logged in cannot log in from another tab of the browser without first logging out.

d. The aforementioned modifications, which were performed during the course of this project, were not existent in the system that was in place before.

e. Using the current system, we were able to access the information and identify suspicious and harmful activity. The application in our suggested solution does not access the database or modify any source code. We create logs for each login and logout event to help us detect nefarious behaviours like several users using the same account to log in. By analysing the real-time generated logs, our application will look for any anomalous activity.

f. Since we won't be accessing the database, we will save all of the routine actions in the logs itself instead of the database, which was how the system was set up before. Without altering the source code, our suggested application will function as a general-purpose add-on that keeps track of both normal and aberrant activity. It runs in parallel in the background. As opposed to the prior system, where we had to access the database and modify the source code, the application in the existing system was not general-purpose for any web application.

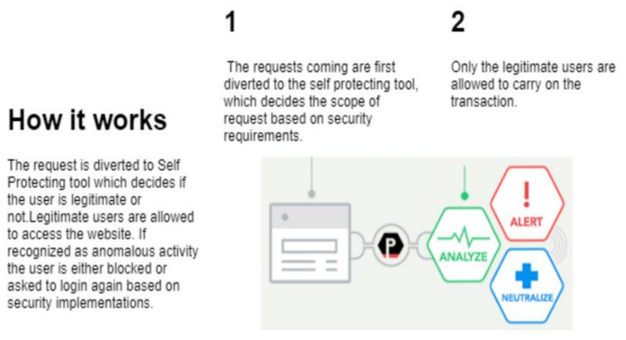

How it works
The request is diverted to Self Protecting tool which decides if the user is legitimate or not.Legitimate users are allowed to access the website. If recognized as anomalous activity the user is either blocked or asked to login again based on security implementations.
1
The requests coming are first diverted to the self protecting tool, which decides the scope of request based on security requirements.
2
Only the legitimate users are allowed to carry on the transaction.
ALERT
ANALYZE
NEUTRALIZE

Fig- 1 : Working of 1st case

II. The Self Protecting tool placed inside the application

Security Features in the Proposed System

a. When a user attempts to commit SQL injection, their behavior is deemed suspect, and they are prohibited from accessing the website. The fundamental block diagram is displayed in Fig. 1.3.

b. The website prints a 404 message if a SQL injection is executed.

c. The aforementioned modifications, which were performed during the course of this project, were not existent in the system that was in place before.

d. We were looking through the error log files on the current system to find unusual requests and suspicious activity. The self-protecting mechanism is included in the application and does not access the database in our suggested system. We use the self-protecting tool within the programme to detect harmful activities like SQL injection inside the application. This tool also aids in preventing injection assaults made to the website. Our programme uses a self-protecting tool, as shown in figure below, to examine all anomalous activity.

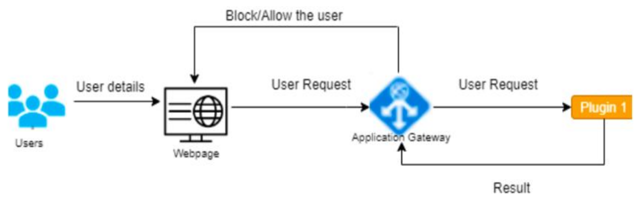


Fig- 2 : Working of 2nd case

3. METHODOLOGY

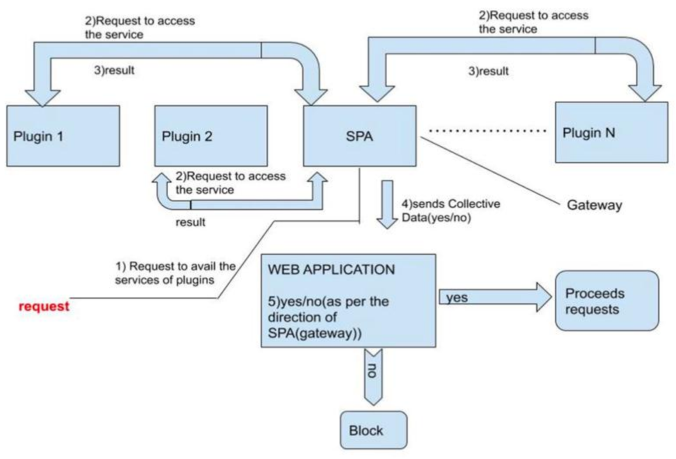

2)Request to access the service
3)result
2)Request to access the service
3)result
Plugin 1
Plugin 2
SPA
Plugin N
2)Request to access the service
result
4)sends Collective Data(yes/no)
Gateway
1) Request to avail the services of plugins
request
WEB APPLICATION
5)yes/no(as per the direction of SPA(gateway))
yes
Proceeds requests
no
Block

Fig- 1 : Block Diagram of Proposed System

RASP limits attacks and only permits authorised users to access the programme by distinguishing between intruders and legitimate users for information. It serves as an additional layer of security for the application and keeps intruders out while it is running. Even small vendors can employ runtime self-protecting applications to secure their products because they are affordable. The size or kind of the application has no bearing on it.

4. IMPLEMENTATION

Server Web Apache (Apache HTTP Server)

Open-source web server software includes Apache HTTP Server. The term "web server" refers to a device that enables website owners to publish material online.

i. Selected a web application and following was implemented:

   a. Disable the web application access from remote system
   b. Understanding Apache HTTP Server
   c. Creating a Virtual Host in Apache and direct all communication of each web application through that each Host when access remotely

ii. Apache's HTTP request interpretation includes logging the following information in a file:
   a. the date and time,
   b. the URL or module that was requested,
   c. and any unusual activity.

iii. The development of a Java class that will read the following input and print it to the console
   a. Web Application Name
   b. User login details

iv. Calling the JAVA Class from Apache and provide the inputs.

Traffic & Data Analysis

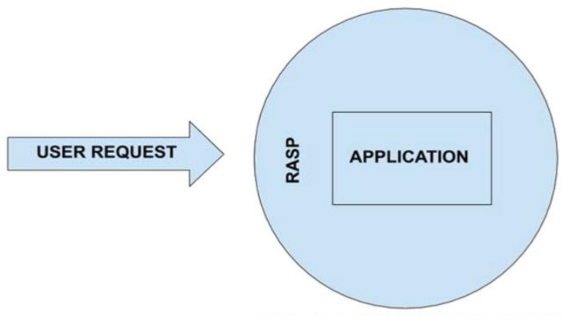
There are numerous phases involved in traffic analysis, such as gathering network data, examining that data, and analysing it to improve system performance.

Steps:

i. Network Data Analysis
ii. Active Passive Attacks

Create a second Java class that analyses the CSV file's contents and transforms it into scenarios or policies (a series of instructions in the form of a txt, prop, or XML file) that prevent users from accessing the website.

Implementation of Self protecting tool placed outside the application



The circle denotes the RASP,in the web application is stored.Before the user gets the access of the application,the RASP is going to analyse the user's request,and then only it will get the access of the application,

Fig- 3 : Implementation of 1st case

We access the GET request in the Apache proxy when the user tries to access the programme using the URL, and we extract the account number from there. We have our details to be obtained with information taken from the GET request header. These details are subsequently transmitted to a Perl script, where validation occurs. At the back end, we execute the Perl script, which calls the Java executable file. The header information is checked against the user's previously collected data. Jar file running in the background does validation. We read the JAR log file and do the appropriate tests. This log file is read in Jar, and user authentication is verified.

The information will be provided by the Jar file, and we have established the prerequisites for carrying out the action. The specific account number obtained from the log file and the GET request in Apache (account number of the user if the user has logged in or logged out). The log file contains all information on each user's login or logout.

Jar generates 0 or 1 depending on the data in the log file. Values 0 and 1 show that the user has previously logged out and hasn't done so recently, respectively. This data was submitted to a Perl script. So, we look to see if the user has engaged in any suspicious behaviour. If so, a message about unauthorised access is displayed. An error notice therefore develops if the value is 1. It prevents the user from using the application. If the number is zero, the user has logged out and is now free to utilise the website. He can then carry out any function within the application. Creating a new account or logging into an existing one can be done next.

Implementation of Self protecting tool placed inside the application

We send the user request to the gateway when a user attempts to access an application via a URL, and the gateway then sends the information to the plugin. The plugins check for the presence of binary SQL injection patterns, and if they do, they return 0 otherwise. If the result is 0, the user is prohibited from visiting the website; otherwise, they are permitted. If a pre-existing account is accessed, we verify that the username, password, and account number are correct and that the user has logged out. Access is not permitted if it is invalid. Allow the user to carry out operations like checking the balance, transferring money, withdrawing money, or closing the account if it is valid. Delete the entry from the database if the user closes their account.

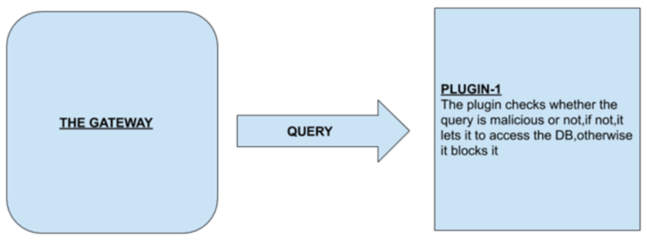

THE GATEWAY
QUERY
PLUGIN-1
The plugin checks whether the query is malicious or not,if not,it lets it to access the DB,otherwise it blocks it

Fig – 4: Implementation of 2nd case

5. OUTCOMES

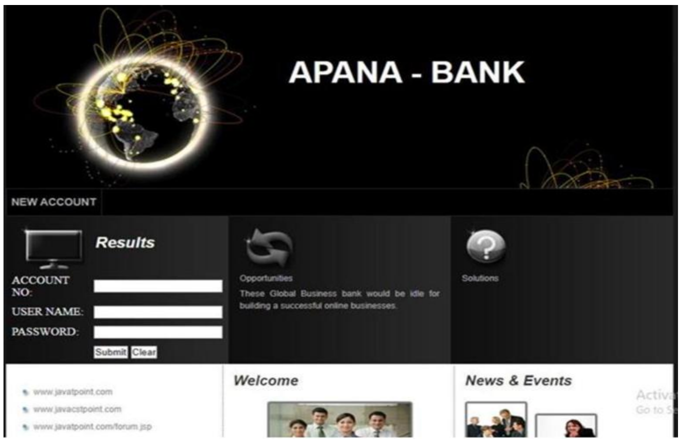


Fig - 5: Real time view of web application

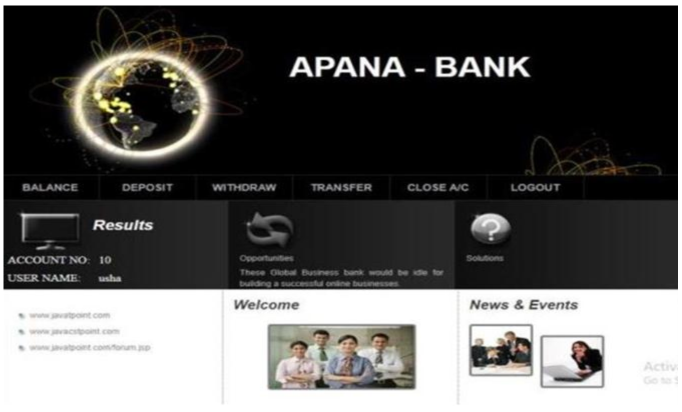
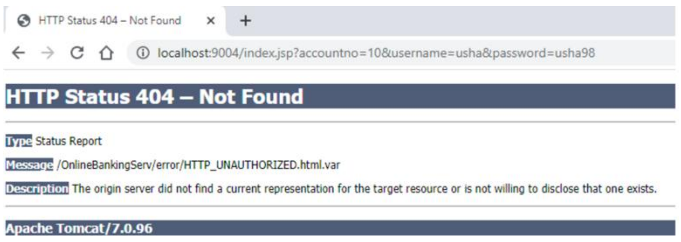


Fig – 6: Legitimate user allowed when self-protecting tool is used



HTTP Status 404 – Not Found

localhost:9004/index.jsp?accountno=10&username=usha&password=usha98

HTTP Status 404 – Not Found

Type Status Report

Message /OnlineBankingServ/error/HTTP_UNAUTHORIZED.html.var

Description The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.

Apache Tomcat/7.0.96

Fig -7 : illegitimate user not allowed to access when self-protecting tool is used

6. CONCLUSION

Self-Protecting Applications are software friendlier, rely little on the type of software, and require minimal technical expertise to utilize. They also offer a respectable service for safeguarding Web Applications against criminal activity. It prevents data leakage and shields the application from the outside world. It safeguards the system from emerging dangers posed by any user simply by evaluating the patterns of user behavior. With its extremely transparent and practical methods, SPA, in contrast to other security software that is difficult to use and incompatible, can be utilized by any web site, from a straightforward single page web application to a large social media application. Additionally, it doesn't call for any modifications to the Application's source code. SPA is therefore genuine software for protecting Web applications.

REFERENCES

[1] Gartner, IT Glossary, http://www.gartner.com/it-glossary/runtime- application-self-protection-rasp/

[2] Veracode, https://www.veracode.com/security/runtime-application-self-protection-rasp

[3] Zhongxu Yin, Zhufeng Li & Yan Cao "A Web Application Runtime Application Self-protection Scheme against Script Injection Attacks" 2018

[4] Salemi Macro "Automated rules generation into Web Application Firewall using Runtime Application Self-Protection" 2020

[5] Amal Saha and Sugata Sanyal. Application layer intrusion detection with combination of explicit-rule-based and machine learning algorithms and deployment in cyber-defence program! https://arxiv.org/pdf/1411.3089.pdf, November 2014.

[6] Alexander Fry. Runtime application self-protection (rasp), investigation of the eectiveness of a rasp solution in protecting known vulnerable target applications. https://www.sans.org/reading-room/whitepapers/ application/runtime-

application-self-protectionrasp-investigation-effectiveness-rasp-solutionprotecting-vulnerable-target-applications-38950, April 2019.

[7] Vivek Gite. Linux incrond inotify monitor directories for changes and take action. https://www.cyberciti.biz/faq/linux-inotifyexamples-to-replicate-directories/, December 2018.

[8] Adrian Lane. Understanding and selecting runtime application self- protection. https://securosis.com/assets/library/attachments/ Understanding_RASP_Immunio_V2.pdf, August 2016.

[9] Dariusz PałkaMarek and ZacharaMarek Zachara. Learning web application rewall - benets and caveats. https://www.researchgate. net/publication/226351120_Learning_Web_Applica tion_ Firewall_-_Benefits_and_Caveats, August 2011.

[10] Amal Saha and Sugata Sanyal. Application layer intrusion detection with combination of explicit-rule-based and machine learning algorithms and deployment in cyber-defence program! https://arxiv.org/pdf/1411.3089.pdf, November 2014.