FreeBSD:
Enhancement Proposal

EECS 4314: Advanced Software Engineering | Assignment 4
April 8th, 2023

Authors: **Pink Fluffy Unicorns**

Benedict Miguel	216237364	benedictmiguel106@gmail.com
Jiachen Wang	215453988	reimsw98@my.yorku.ca
Keshav Gainda	216891319	gaindakeshav@gmail.com
Ketan Bhandari	217550765	ketan11@my.yorku.ca
Mohaimen Hassan	216187809	hassan08@my.yorku.ca
Pegah Fallah	216262081	fpegah@my.yorku.ca
Shami Sharma	217262510	shami012@my.yorku.ca
Sidharth Sudarsan	216697120	lensman@my.yorku.ca
Suryam Thaker	217281031	suryam@my.yorku.ca
Xiaochuan Wang	214107106	wangx997@my.yorku.ca

Table of Contents

Abstract

This report proposes the addition of a built-in Web Application Firewall (WAF) feature to FreeBSD. Currently, FreeBSD relies on third-party WAF solutions, which are complicated to configure and maintain. This feature aims to enhance security against malicious server requests, improve performance by reducing overhead, provide cost savings, simplify security management, and potentially increase FreeBSD adoption. Two implementation approaches are to implement it as a kernel module or as a userland application. Integrating the WAF in the kernel space offers benefits such as direct access to hardware resources, and increased security against tampering. This implementation may be more disruptive during updates, requires extensive testing, and expert knowledge in FreeBSD. The userland application method offers benefits such as simpler maintenance, and easier updates and patches without system restarts. However, WAF functionalities are limited in comparison to kernel level implementation, and it may be subject to vulnerabilities such as buffer overflow. As a result, the first approach was chosen. The kernel space implementation can be done in two ways; creating a separate subsystem for the WAF, or extending the functionality of an existing subsystem. The chosen approach was to integrate the WAF into the Networking subsystem to avoid major changes to the architecture, and limit impact on other dependency relationships. Four key components are specified in the design of the built-in WAF: Core Module, WAF Engine, Rule Management, and Logging and Reporting. The report finds that the high-level impacts of this enhancement include enhanced resource management, greater control for administrators, and stronger security by making the WAF more tamper-resistant. The low-level impact involves specific changes and interactions within various subsystems, including networking, security, process management, memory management, and system I/O. In order to test the impact of these interactions, various tests will be conducted on every subsystem involved, including network traffic testing, network load testing, protocol testing, penetration testing, and others. The addition of this feature comes with risks such as a potential impact on system performance, false positives, configuration complexity, and the possibility of attackers evading the WAF. Lastly, there are limitations to a WAF's protection, resource constraints, and compatibility issues with different web servers and applications.

Proposal Motive

FreeBSD is a popular operating system choice when it comes to servers due to its security, stability and scalability. Moreover, FreeBSD has a reputation for its performance optimization, which makes it a viable option for hosting network-based services. Thus, we see FreeBSD being used in various high-traffic sites, from services of small businesses to big tech giants.

Being widely utilized by high-traffic services, there is definitely a concern for the security of these machines. Web application attacks are involved in more than 40% of all data breaches, making them the single most common type of attack vector.[10] This prompts a need for greater measures to ensure the safety of the web applications. Improving network security of FreeBSD would greatly benefit all its users, especially the smaller-scale individual users and companies without extensive hardware to secure network perimeters.

Proposed Enhancement

To improve the security of the web applications running on FreeBSD servers, we propose a built-in Web Application Firewall (WAF) for the OS. The WAF will provide an additional layer of security against common attacks such as SQL Injection, cross-site scripting, cross-site forgery and DoS. by providing defense at the Layer 7 of the OSI model.[9] Malicious requests directed at the server are filtered out by the WAF using a set of rules

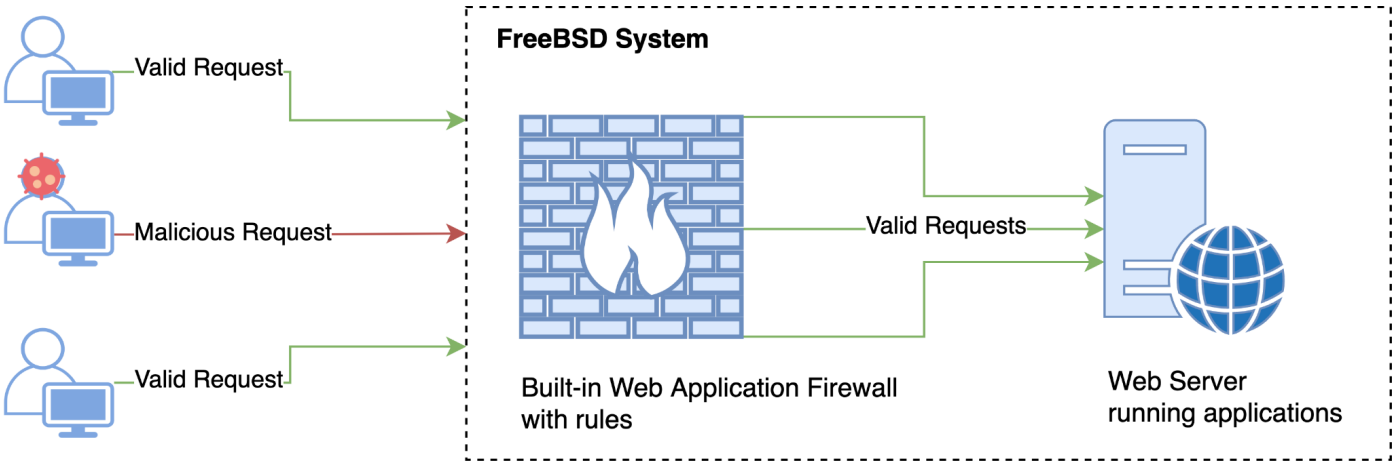


Fig. 1. A high-level view into the functionality of the proposed built-in WAF

Current State

FreeBSD includes, by default, IP Firewall, Packet Filter, IPFILTER and NetBSD Packet Filter firewalls, none of which can provide the services of Web Application Firewalls.[2] This means that for FreeBSD to be used as a scalable and secure web server/applications, users must install a third-party WAF such as ModSecurity or NAXSI and configure it for the specific web server being used. While these third-party solutions can provide effective WAF functionality on FreeBSD systems, they can be complex to configure and maintain as the network and the services on it scale up.

Values and Benefits

Enhanced Security

The built-in WAF provides an additional layer of security for network services running on FreeBSD against a wide range of attacks such as cross-site scripting, cross-site forgery and SQL injection among others.

Improved Performance

By integrating the WAF natively into the OS, the system will have reduced overhead in comparison to a third-party solution and be able to offer better performance with the given resources. A solution designed specifically for FreeBSD can take full advantage of its unique capabilities.

Cost and Management

With a built-in solution, users would not need to pay for additional software or services to secure their web applications. This can also make it easier to maintain secure web servers for regular users and system administrators with minimal configuration.

Increased Adoption

The inclusion of the WAF could make FreeBSD an attractive option for independent users and small scale organizations looking for a secure and reliable platform for network services. Eventually, this can lead to increased adoption and usage in the market.

Implementation Approaches

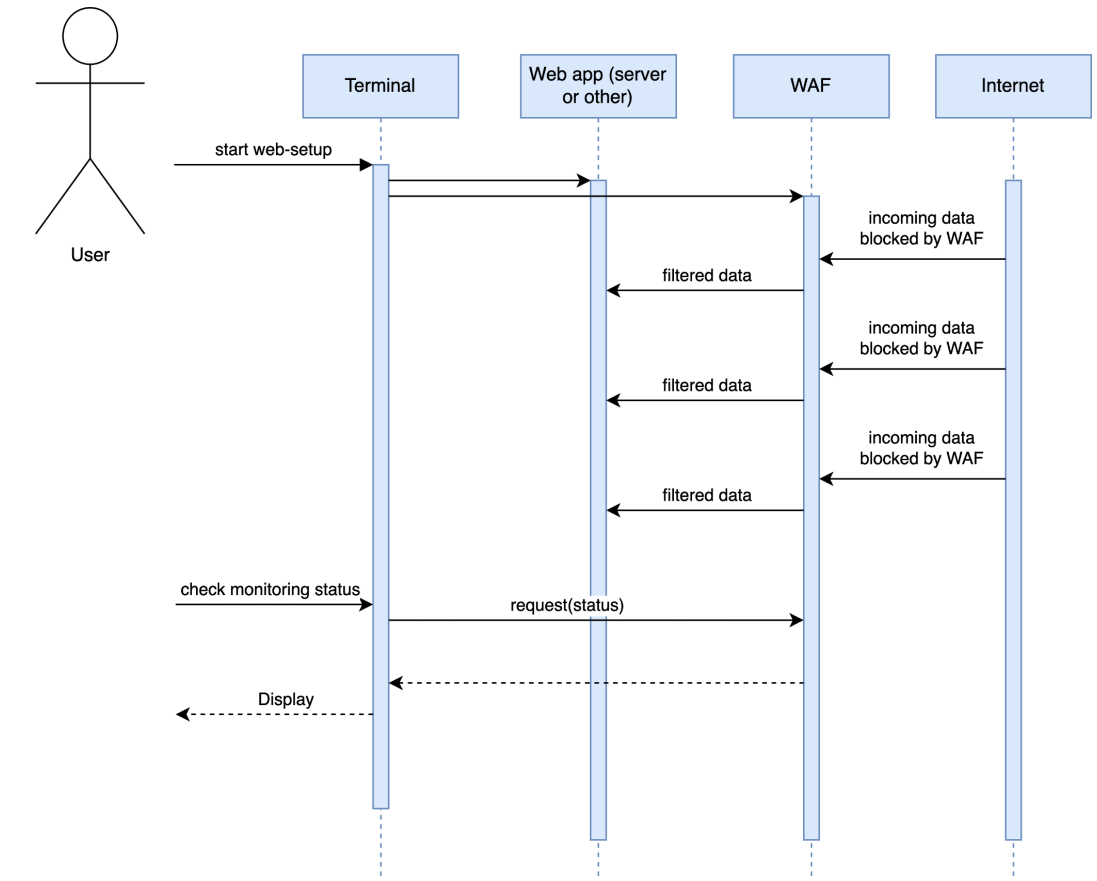
The following approaches were decided based on the benefits it would provide, the drawbacks, and the level of knowledge and experience required to implement.

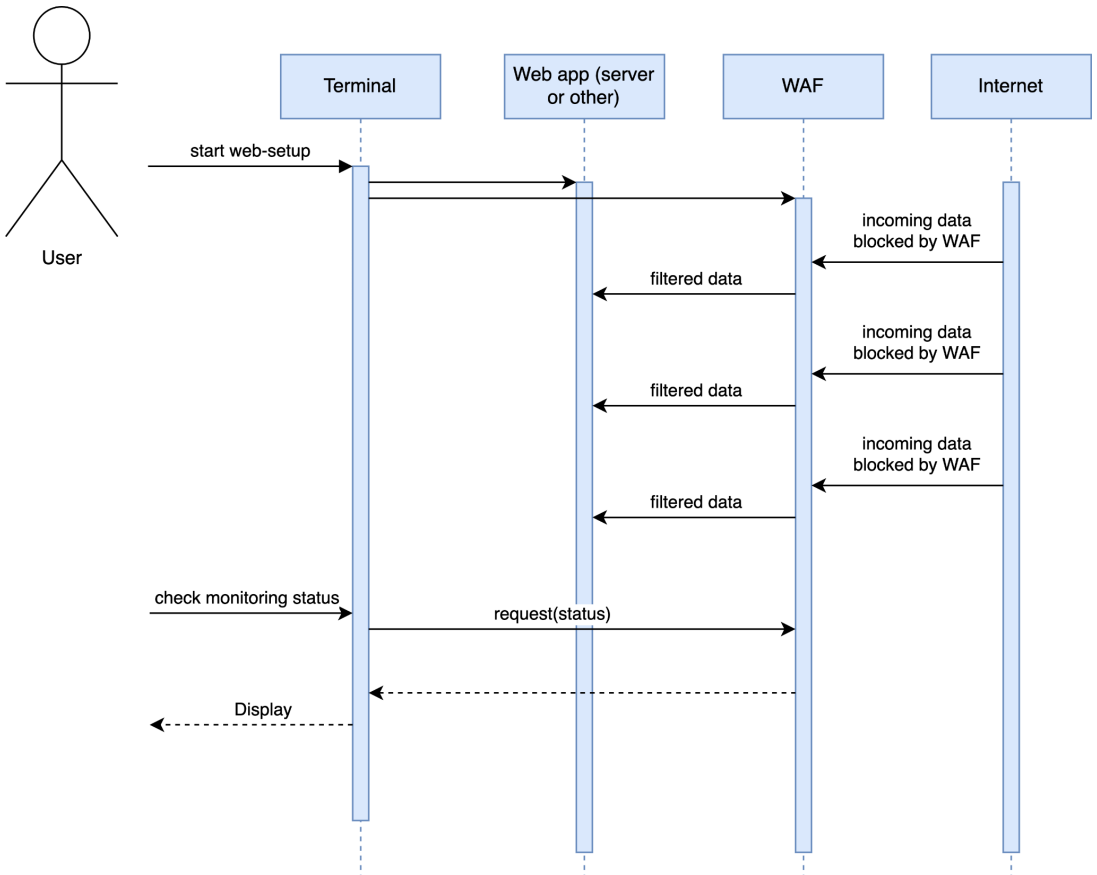
Kernel Module

There are two methods in implementing a WAF as a kernel module. The first method would be to create a separate subsystem for a WAF. The second method involves using an existing subsystem and extending its capabilities. The method of choosing how to implement WAF in the kernel will be discussed further into the document. With regards to implementing WAF in the kernel it comes with the benefits of having a lower overhead due to having a direct access to hardware resources. Integration in the kernel space would also allow fine-grained control on the capabilities of the WAF, allowing features to be enabled or disabled. Existing in the protected kernel space, it comes with an increase to security in terms of attackers tampering with the WAF. However, there are a myriad of complications that come with implementing in the kernel. One of them being the required advance knowledge in FreeBSD to develop and maintain the WAF. Integration in the kernel space may also introduce new vulnerabilities that can potentially impact other subsystems, requiring intensive testing. Finally, the module would require rebooting the whole system when an update or a patch has to be applied, which can be disruptive to system operations.

Userland Application

The second approach involves creating a native application that runs outside the kernel and provides WAF functionalities on top of the OS. The benefit of doing this is that the development and the maintenance is simpler in comparison to a kernel level implementation. By implementing outside the protected kernel space, the risk of system crashes is also reduced as it runs in its own protected environment. System disruptions are also reduced because updates/patches can be done without restarting the whole system. However, being an application outside the kernel also has its downsides. These downsides include a reduced performance and system utilization as the application does not have direct access to the kernel. WAF functionalities are also limited in terms of advanced network security features. Being a user-space application, the application itself can be subject to certain vulnerabilities such as buffer overflow.


User
Terminal
Web app (server or other)
WAF
Internet
start web-setup
incoming data blocked by WAF
filtered data
incoming data blocked by WAF
filtered data
incoming data blocked by WAF
filtered data
check monitoring status
request(status)
Display


User
Terminal
Web app (server or other)
WAF
Internet
start web-setup
incoming data blocked by WAF
filtered data
incoming data blocked by WAF
filtered data
incoming data blocked by WAF
filtered data
check monitoring status
request(status)
Display

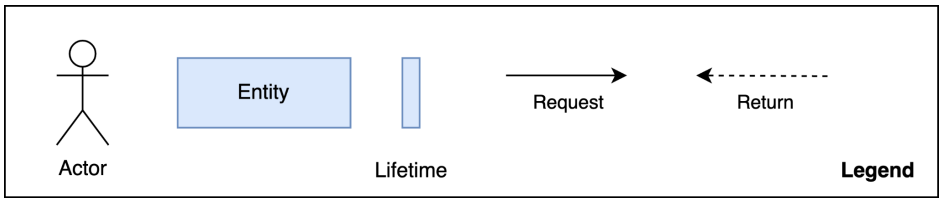


Fig. 2 and 3. Illustration of use-cases involving the WAF as a kernel module and a userland application

Stakeholders

FreeBSD Community

The FreeBSD community is the primary stakeholder for the development of this feature and any changes made to the kernel. As a kernel module, the WAF will be integrated into the FreeBSD kernel, and the community will be responsible for reviewing and testing the changes to code.

Security Professionals

Security experts of the community will play a crucial role as they are able to provide insights and recommendations in the development of a WAF. They are also able to identify potential risks and vulnerabilities regarding the implementation.

OS End Users

System administrators, web developers and the everyday users are stakeholders in the development being the end users of FreeBSD. They are responsible for configuring, maintaining and updating the WAF and can provide feedback on usability and performance.

Third-Party Vendors

Third-party vendors who develop security solutions for FreeBSD may also be stakeholders in the development. They might express interest in integrating their existing products with the WAF module and may provide feedback on how it can be improved to better support their solutions.

Chosen Approach

The chosen approach to implement the WAF in our proposal is to develop a kernel module. There are two sub-approaches to integrating the WAF module into the kernel; developing a dedicated subsystem or integrating it into an existing subsystem. We opt for the latter for the following reasons.

- Avoid major changes to the overall architecture.
- Limit the impact on other interactions and dependency relationships.
- Does not call for its own subsystem within the current style of architecture.

The objective is to provide a solution that is highly-compatible as well as easy to set up and operate. The module should be compatible with popular web servers such as Apache and Nginx out of the box. An interface should be provided for the community to develop support for any other web servers being used.

Integrating into the Network subsystem

The WAF module will be integrated into the Network subsystem in the current architecture. Since the module will be inspecting and filtering network traffic, it will benefit from being closer to the TCP/IP stack, allowing function at a lower level. This will potentially provide better performance and lower latency.

The alternative is to integrate with the Security subsystem. Although it is responsible for enforcing security policies within the kernel, we believe there will be a lack of necessary network stack knowledge to effectively filter network traffic.

Proposed Implementation

The proposed implementation of the kernel module contains four components as described below.

Core Module

The core component of the kernel module which provides a low-level interface for intercepting network traffic at the network stack level. It is responsible for capturing the traffic and passing it to the WAF engine for analysis.

WAF Engine

The WAF engine is responsible for analyzing the incoming network traffic and determining whether it is malicious or not. The engine uses a set of rules to determine whether the request will be allowed or blocked. It is highly configurable to allow for fine-grained control over the enforced rules and policies.

Rule Management

This component is responsible for managing the rules and policies used by the engine. It should allow a user-friendly interface to create, modify, enable or disable the rules.

Logging and Reporting

It is responsible for collecting and storing logs generated by the engine. The reporting system should allow users to view and analyze the results in a user-friendly format.

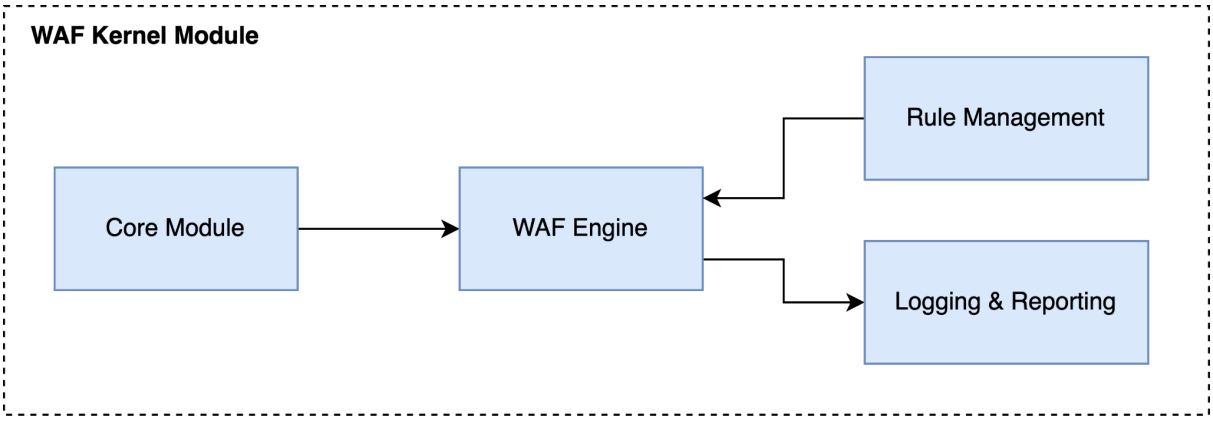
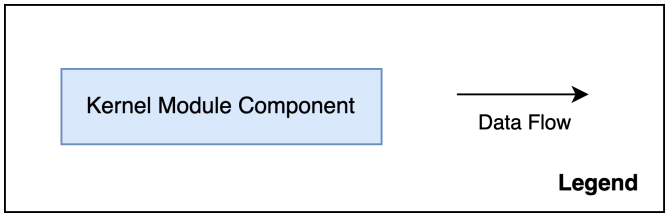


Fig. 4. Components of the proposed built-in WAF



Design Guidance

pfSense is a popular open-source firewall distribution based on FreeBSD, providing security features for a variety of network environments. It comes with a built-in WAF based on the open-source ModSecurity engine. Since pfSense is based on FreeBSD, it can serve as a reference for implementing a WAF kernel module in FreeBSD.[8] The experience gained from developing and using the WAF feature in pfSense can be leveraged to design and implement a similar feature in FreeBSD. Although design decisions may not be directly applicable, valuable insight can be gained into design considerations and potential challenges of developing such a module.

Impact

High-Level

The high-level impact involves a trade-off between improved performance, control, and security versus potential maintenance challenges and system disruptions during updates. This solution is expected to enhance resource management, offer greater control for administrators, and strengthen security by making the WAF more tamper-resistant. However, it demands advanced FreeBSD knowledge for implementation, and the update process may be more disruptive due to possible full system reboots.

Low-Level

The low-level impact of this enhancement proposal are the specific changes and interactions within various subsystems. The integration of a WAF within the FreeBSD networking subsystem requires a harmonious interaction between multiple subsystems.

Firstly, as part of the networking subsystem, the WAF must interact with TCP/IP, sockets, and network interfaces to manage web traffic. It follows that the WAF must interact with the Security subsystem, as it needs to perform access control, data protection, and policy enforcement. This will allow the WAF to safeguard web applications from attacks and unauthorized access. The WAF will also interact with the Process Management subsystem as new functions may need to be added to extend the functionality of the Networking Subsystem. This includes adding the ability to control WAF processes and threads, and efficiently allocating system resources to the WAF. Next, the Memory subsystem plays a critical role in managing the memory resources necessary for the WAF's operation. This includes managing security rules, storing temporary data, and processing web traffic. Furthermore, the memory subsystem caches user authentication data and web content. It also manages WAF buffers with web traffic data, ensuring the smooth flow of information between the WAF and web applications. Lastly, the System I/O subsystem connects the WAF to hardware-level network I/O operations. This interaction requires the Network subsystem to communicate with the System I/O subsystem to manage network hardware drivers, such as Ethernet cards or wireless adapters.

Directories and Files

New directories will be created to store the WAF source code, configuration files, and any associated modules by creating subdirectories within the kernel source tree. To effectively monitor and debug the WAF's operation, log files will be generated to help system administrators track security incidents and performance issues. Also, the WAF may require data files containing security rules, and signatures, so these files will need to be stored efficiently within the system. Lastly, integrating the WAF may require modifications to system libraries, which should be managed to maintain system stability.

Testing the Impact of Interactions

A series of tests were conducted on every subsystem that was involved in the interaction process, employing the testing methods outlined below.

Network

Network Traffic Testing

Network traffic testing can be used to evaluate how the network subsystem and the added WAF feature respond to simulated traffic, measuring packet loss, latency, throughput, and testing the compatibility, functionality, and usability of FreeBSD.

Network Load Testing

Network load testing can be used to test the capacity and scalability of the FreeBSD network subsystem and the added WAF feature under heavy traffic. It identifies bottlenecks and performance issues, and helps optimize network configuration.[3]

Protocol Testing

Network protocol testing can be used to assess compatibility, functionality, and security of the FreeBSD network subsystem and WAF feature with different protocols. It detects protocol-related issues, analyzes packet flows, and identifies vulnerabilities using various tools.[6]

Security

Penetration Testing

Penetration testing can be used to evaluate the impacts of the interaction between the FreeBSD security subsystem and the WAF feature by simulating attacks against web applications and services protected by the WAF. This can help identify any security vulnerabilities in the WAF or any potential weaknesses in the security subsystem of FreeBSD.

Event Logging and Analysis

Event logging and analysis is an effective method to test the impact of the interaction between FreeBSD's security subsystem and the WAF feature. It involves monitoring system events, generating

logs, analyzing them, and identifying security issues or anomalies to improve the system's security posture.

Process Management

Resource Usage Testing

Resource usage testing can be used to measure the impact of the interaction between the process management subsystem of FreeBSD and the WAF feature. It analyzes the system's resource usage, identifies bottlenecks or performance issues, and optimizes the system's performance using load testing tools, performance monitoring tools, and profiling tools.

Process Monitoring and Analysis

Process monitoring and analysis can be used to test the impact of the interaction between the process management subsystem of FreeBSD and the WAF feature. It involves monitoring system processes, analyzing their behavior, and identifying issues that may affect system performance or stability.

Memory

Memory Usage Testing

Memory usage testing can be used to analyze the impact of the interaction between the memory subsystem of FreeBSD and the WAF feature. Testers simulate different workloads and traffic patterns to identify any memory-related issues that may impact system performance or stability.

Memory Leak Testing

Memory leak testing can be used to analyze how the memory subsystem of FreeBSD interacts with the WAF feature. It will help to identify memory leaks that could affect performance and stability by monitoring system memory usage over time and simulating different workloads and traffic patterns. [1]

System I/O

I/O Performance Testing

I/O performance testing can be used for assessing how the interaction between the system I/O subsystem of FreeBSD and the WAF feature affects system performance. It involves measuring data transfer rates and latency times under different workloads and traffic patterns to identify potential bottlenecks or other issues.

Error Handling Testing

Error handling testing can be used to evaluate how the system I/O subsystem of FreeBSD interacts with the WAF feature and identify any issues related to error detection, reporting, and recovery. This testing involves intentionally introducing errors and faults to the system to assess how it responds.[7]

Effects

Maintainability

Since the kernel module is directly interacting with the operating system, when the WAF needs to be upgraded or patched for maintenance, it will be easier to achieve that through the operating system tools.

Evolvability

Addition of new features like network protocols or security features at the kernel level can be much easier, since WAF is directly connected to the different subsystems.

Testability

The kernel level system test can be more accurate and comprehensive, aiding to find the kernel level bugs, especially in comparison to the userland application approach mentioned above. Additionally, since it is on the kernel level, performing system level tests can be more efficient, ensuring that WAF is functioning correctly at the lowest level.

Performance

On the positive side, the WAF can be very efficient and fast on web application traffic, because it is built into the system as part of the kernel. This can help to reduce latency and overhead resource utilization, making it a good fit for high-traffic use-cases.

Risks

Performance Impact

The introduction of a WAF feature can result in a performance impact on the system due to the additional processing overhead required to inspect and filter web traffic. This could lead to slower response times for web requests and potential performance issues for high-traffic web applications.

Generate False Positives

A WAF can potentially generate false positives and block legitimate traffic, which can impact the availability of web applications and services.

Configuration Complexity

Configuring a WAF can be complex, and there is a risk that misconfiguration could result in security vulnerabilities or false negatives.

Attacker Bypass
WAFs are not foolproof, and attackers can use various techniques to bypass or evade them, such as using encrypted traffic or exploiting vulnerabilities in the WAF itself.

Limitations

Not a Substitute for Secure Coding
A WAF is not a substitute for secure coding practices and should not be relied on as the sole defense mechanism for web application security.

Limited Protection
A WAF can only provide protection against known attack signatures and may not be able to detect or prevent zero-day attacks or other unknown threats. [5]

Resource Constraints
Introducing a WAF feature to FreeBSD may require additional system resources, such as memory and processing power, which could be a limitation for systems with limited resources.

Compatibility Issues
A built-in WAF would need to be designed to work with a variety of different web servers (apart from popular choices like apache and nginx) and applications, which could present compatibility issues with some software or configurations.

Data Dictionary

Web Application Firewall (WAF): A security measure that provides an additional layer of protection for web applications running on servers against common attacks such as SQL Injection, cross-site scripting, and file inclusion. [1]

Packet Filter: Another default firewall in FreeBSD that provides network address translation (NAT), packet filtering, and other network services. Like IP Firewall, it does not offer the same functionality as a Web Application Firewall.

Kernel Module: A piece of code that can be loaded into the operating system's kernel at runtime to add or extend its features. In the context of the proposal, the WAF could be implemented as a kernel module for lower overhead and fine-grained control.

Userland Application: A software application that runs in user space, outside of the operating system's kernel. In the context of the proposal, the Web Application Firewall could be implemented as a userland application to simplify development and maintenance, though it may not offer the same performance benefits or security enhancements as a kernel module implementation.

Lessons Learned

In the process of assessing various aspects of the FreeBSD enhancement proposal, a comprehensive understanding of stakeholders and their concerns was vital. This involved considering multiple approaches to decision-making while taking into account the potential impact on subsystems, performance, and maintenance. Additionally, the evaluation of resource requirements played a crucial role in determining the feasibility of the proposed enhancements. Anticipating challenges, risks, and limitations enabled a proactive approach to addressing potential obstacles in the project. As a result, the proposal underwent revisions based on a thorough investigation of trade-offs; for instance, the initial idea of integrating Active Directory and Group Policy was replaced with the implementation of LDAP and Samba, following a consensus within the community.

Conclusion

In conclusion, web application attacks continue to be a major threat to network security, and the implementation of a web application firewall (WAF) can significantly reduce the risk of such attacks. While third-party WAF tools are available, integrating a built-in WAF into the FreeBSD kernel would provide several benefits, including improved performance, reduced latency, and better security. The proposed approach involves developing a kernel module that consists of four components: the main kernel module, WAF engine, rule management, and logging and reporting. While there are some challenges associated with integrating a WAF into the FreeBSD kernel, the potential benefits make it a worthwhile endeavor. Overall, the implementation of a built-in WAF in the FreeBSD kernel would be a significant step towards improving network security and protecting against web application attacks, increasing its adoption and usage.

References

1. Burns, B. (n.d.). *Memory leak testing in continuous integration systems*. TotalView by Perforce. Retrieved April 6, 2023, from https://totalview.io/blog/testing-memory-leaks-continuous-integration-systems

2. *Chapter 32. firewalls*. FreeBSD Documentation Portal. (n.d.). Retrieved April 6, 2023, from https://docs.freebsd.org/en/books/handbook/firewalls/

3. Frank, Makesh, Rina, Namburi, satish, Madhavilatha, Supriya, Dian, Wang, H., Pavan, Dhanish, Peter, Satham, Pierre, Mohit, Krish, Goutham, Le, H., Hashmi, H., Anamika, ... Bhatt, A. (2023, March 26). *Performance testing vs load testing vs Stress Testing (difference)*. Software Testing Help. Retrieved April 6, 2023, from

https://www.softwaretestinghelp.com/what-is-performance-testing-load-testing-stress-testing/

4. The FreeBSD Documentation Project. (2012, 01 01). *FreeBSD Architecture Handbook*. https://docs.freebsd.org/en/books/arch-handbook/book/

5. *Limitations of WAF*. Source Defense. (n.d.). Retrieved April 6, 2023, from https://sourcedefense.com/glossary/limitations-of-waf/#:~:text=Limited%20protection%3A%20A%20WAF%20only,in%20addition%20to%20a%20WAF.

6. *Network testing types*. Digital Transformation and Enterprise Software Modernization. (n.d.). Retrieved April 6, 2023, from https://www.microfocus.com/documentation/silk-test/200/en/silktestclassic-help-en/STCLASSIC-B12DC3CE-THE-KINDS-OF-NETWORK-TESTING-YOU-CAN-PERFORM.html

7. *Understanding and detecting disordered error handling*. (n.d.). Retrieved April 6, 2023, from https://www.usenix.org/system/files/sec21summer_wu-qiushi.pdf

8. *Versions of pfSense software and freebsd*. Releases - Versions of pfSense software and FreeBSD | pfSense Documentation. (n.d.). Retrieved April 6, 2023, from https://docs.netgate.com/pfsense/en/latest/releases/versions.html

9. *What is a WAF?* | *web application firewall explained* | *Cloudflare*. (n.d.). Retrieved April 6, 2023, from https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/

10. Verizon, "2021 Data Breach Investigations Report," 2021. https://www.verizon.com/business/resources/reports/2021-data-breach-investigations-report/