# **So sánh phạm vi rule và policy giữa ModSecurity-nginx và NAXSI**

1. **Giới thiệu**

Đầu tiên thì Web Application Firewall (WAF) được biết đến như là là một lớp bảo vệ được triển khai để giám sát, lọc và chặn các lưu lượng HTTP/S độc hại đến ứng dụng web. Khác với firewall truyền thống, WAF hoạt động ở tầng ứng dụng (L7 – Application Layer), có khả năng phát hiện các cuộc tấn công như SQL Injection, Cross-Site Scripting (XSS), Remote Code Execution (RCE), Path Traversal, v.v.

WAF có thể triển khai theo nhiều kiến trúc khác nhau: dạng reverse proxy, plugin trong web server, hoặc tích hợp sẵn trong firewall/UTM.

Ứng dụng web thường là mục tiêu chính của các cuộc tấn công mạng hiện nay. Vì thế WAF là một giải pháp giúp bảo vệ các lỗ hổng chưa được vá, ứng dụng lỗi thời hoặc các dịch vụ public khó kiểm soát. Đặc biệt trong môi trường DevOps/CI-CD, nơi phần mềm thay đổi thường xuyên, WAF là tuyến phòng thủ thứ hai sau các biện pháp secure coding và pentest. Ngoài ra WAF còn hỗ trợ log, giám sát và tích hợp với SIEM để cảnh báo sớm các mối đe dọa.

Và Opnsense cũng cung cấp tường lửa ứng dụng web NAXSI (WAF) để bảo vệ các máy chủ web chống lại các cuộc tấn công mạng. NAXSI WAF có thể dễ dàng được bật bằng cách cài đặt Plugin Web Server *os-nginx* trên tường lửa OpnSense.

Tiếp theo ta cần phân biệt được hai khái niệm ModSecurity và Naxsi trước khi tìm hiểu chuyên sâu hơn về nó. Nói một cách đơn giản thì:

ModSecurity là một WAF mã nguồn mở mạnh mẽ, hỗ trợ nhiều web server (Apache, Nginx, IIS), nhưng phổ biến nhất hiện nay là ModSecurity-nginx. Nó sử dụng một ngôn ngữ rule riêng gọi là SecRules, và có thể tích hợp các bộ rule như OWASP CRS (Core Rule Set) để bảo vệ ứng dụng khỏi hầu hết các attack vector phổ biến. Trong tài liệu này, ModSecurity được cài đặt thông qua: *sudo apt install nginx libnginx-mod-security* ...sau đó cấu hình trên máy chủ web nội bộ để kiểm tra khả năng phát hiện và chặn tấn công.

NAXSI(Nginx Anti XSS & SQL Injection) là một WAF đơn giản, nhẹ, dành cho Nginx, với cách tiếp cận khác biệt: scoring các ký tự nguy hiểm (ví dụ: ' , <, ; , --,...) và đánh giá tính nguy hiểm của input thay vì dùng rule logic như ModSecurity. Nó thích hợp chạy trên các thiết bị gateway/firewall. Cùng với việc triển khai trên OPNsense – một firewall mã nguồn mở mạnh mẽ – cung cấp sẵn plugin *os-naxsi* cho phép triển khai WAF ngay tại tầng gateway, hỗ trợ người quản trị dễ dàng quản lý WAF trên GUI, định nghĩa whitelist và theo dõi log tấn công.

Tài liệu này sẽ giúp người đọc hiểu rõ hơn về NAXSI WAF, các cơ chế vận hành và khả năng của nó và cung cấp hướng dẫn về việc định cấu hình WAF trên tường lửa OpnSense với mô -đun Nginx NAXSI, đồng thời phân biệt phạm vi bảo vệ, hiệu suất, khả năng phát hiện tấn công, độ tùy biến rule, cũng như tính dễ vận hành và phù hợp với các môi trường cụ thể (SMB, nội bộ, production). Hơn nữa, chúng tôi minh họa quá trình thiết lập danh sách trắng hoặc danh sách đen bằng cách sử dụng chức năng ACL dựa trên IP của máy chủ Nginx.

1. **Tổng quan về ModSecurity-nginx**
   1. **Lịch sử phát triển và đặc điểm**

ModSecurity có một tên gọi khác là Modsec, ban đầu modsecurity được phát triển như một mô đun cho máy chủ Apache HTTP. Sau đó, Mod Security đã được mở rộng để cung cấp chức năng lọc phản hồi và yêu cầu giao thức truyền văn bản, cùng với nhiều tính năng bảo mật khác.

Mod Security có khả năng hoạt động trên nhiều nền tảng khác nhau như máy chủ Apache HTTP, Microsoft IIS và Nginx. Đây là một phần mềm hoàn toàn miễn phí và được phát hành dưới giấy phép Apache 2.0.

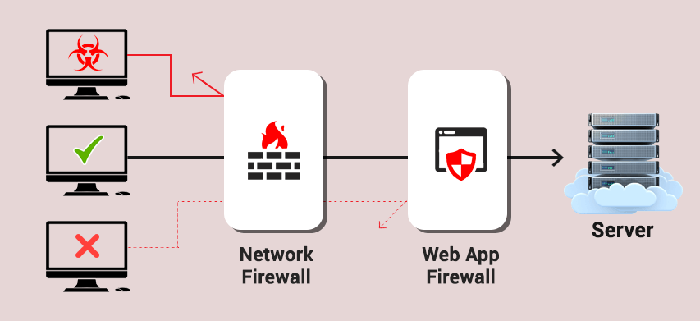
Để Modsecurity hoạt động cần có một bộ quy tắc. Bộ quy tắc này được phát triển dựa trên sự hợp tác giữa SpiderLabs và OWASP. Hai đơn vị đã cùng nhau phát triển bộ quy tắc generic được dùng cho các chức năng của Modsecurity. Để có những thông tin chi tiết về các bộ quy tắc này thì có thể tham khảo thêm các thông tin từ owasp-modsecurity-crs. Ngoài ra cũng có thể tự tạo một bộ quy tắc hoặc mua bộ quy tắc tùy chỉnh theo nhu cầu của cơ quan, tổ chức,...

Liên hệ với nginx thì đây là một máy chủ website. Nginx có thể được sử dụng làm một reverse Proxy, một bộ cân bằng tải cũng như bộ đệm HTTP. Nginx là một phần mềm mã nguồn mở hoàn toàn miễn phí được phát hành bởi Igor Sysoev vào năm 2004. Phần mềm này được phát hành theo giấy phép BSD 2. Nginx cung cấp ngôn ngữ cấu hình quy tắc là 'SecRules' dùng để theo dõi, ghi nhật ký cũng như lọc các thông tin liên lạc.

“*Nginx Modsecurity hay Modsecurity Nginx chính là Mod Security được cài đặt trên nền tảng Nginx. Nginx Modsecurity cung cấp các biện pháp bảo vệ để chống lại các lỗ hổng chung bằng cách sử dụng Bộ quy tắc OWASP ModSecurity (CRS).”* (CÔNG TY CỔ PHẦN GMO-Z.com RUNSYSTEM Thành viên của tập đoàn GMO Internet, 2025)

* 1. **Kiến trúc và cách thức hoạt động**
* *Kiến trúc tổng quan*
* *Quy trình xử lý request/response*
* *Các thành phần chính của ModSecurity*
* *Cách tích hợp với NGINX*
  + 1. **Tổng quan về Modsecurity core ruler**

Modsecurity là một tường lửa ứng dụng cho website vì vậy bản thân nó không có nhiều khả năng phòng chống các tấn công mặc định. Nó cũng không có sẵn các rule được cấu hình cẩn thận. Để tận dụng những tính năng của Mod security có sẵn, nhà phát triển lập ra một luật chặt chẽ, đầy đủ và hoàn toàn miễn phí.



Khác với các hệ thống dò tìm, phát hiện mối nguy hiểm xâm nhập khác, các core rule này có tính năng cung cấp sự bảo vệ từ những thành phần chưa được biết đến ở các ứng dụng web. Breach security khuyến cáo rằng với những core rule đã được xây dựng sẵn này thì chỉ sử dụng như một bản hướng dẫn để triển khai cho hệ thống Mod Security.

* + 1. **Nội dung của Core ruler**

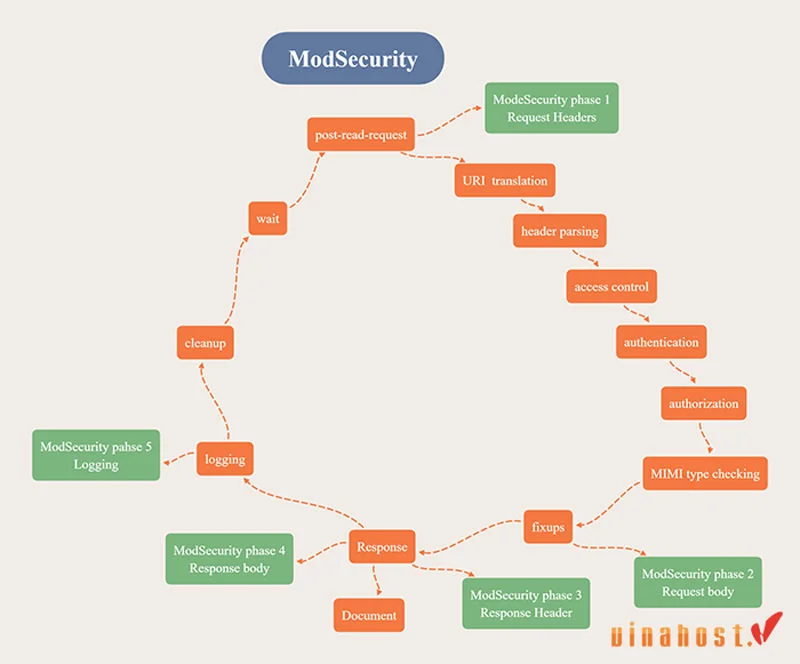
Để đảm bảo sự bảo vệ toàn diện cho website, core rule bao gồm các thành phần sau:

* Thành phần bảo vệ luồng dữ liệu HTTP: Chức năng của thành phần này là phát hiện các hành vi vi phạm giao thức HTTP và thực hiện các quy tắc xử lý đã được xác định theo core rule.
* Phòng chống các cuộc tấn công phổ biến vào Web server: Tính năng này giúp phát hiện các cuộc tấn công vào ứng dụng bảo mật của web server.
* Một số tính năng khác: Tự động phát hiện Bots, ngăn chặn và phát hiện truy cập từ Trojan horses, nhận diện lỗi tấn công và các thông báo từ web server.
  + 1. **Cấu trúc của Core ruler**

Cấu hình của Rule modsecurity bao gồm các thông tin khác nhau, yêu cầu đặc biệt về nội dung. Dưới đây là một số cấu trúc của Rule modsecurity:

* Cấu trúc chứa các chỉ thị, hàm chuyển đổi, biến và signature, cùng với các action tương ứng cho phép hoặc không cho phép.
* Thông tin về logic để phát hiện các cuộc tấn công.
* Thông tin thiết lập chính sách, quyết định hành động xử lý cụ thể khi phát hiện cuộc tấn công.

Modsecurity cho phép xử lý từng thành phần tấn công khác nhau. Core rule được xây dựng dựa trên mẫu có sẵn và có thể tự thay đổi, thêm hoặc giảm bớt các core rule.



1. **Tổng quan về NAXSI**
   1. **Lịch sử phát triển và đặc điểm**

Tương tự như Apache có ModSecurity – một module bảo mật mạnh mẽ hoạt động như một Web Application Firewall (WAF), Nginx cũng có một module bảo mật riêng là NAXSI, viết tắt của "NGINX Anti XSS and SQL Injection". Đây là một WAF mã nguồn mở, được thiết kế chuyên biệt cho Nginx, với mục tiêu bảo vệ ứng dụng web khỏi các dạng tấn công phổ biến như SQL Injection và Cross-Site Scripting (XSS).

Trên nền tảng OPNsense, NAXSI được tích hợp dưới dạng một plugin dành cho dịch vụ nginx, cho phép người dùng cấu hình và kích hoạt nhanh chóng ngay trên giao diện quản lý firewall. Khi được cài đặt và bật trong OPNsense, NAXSI hoạt động như một lớp bảo vệ ứng dụng web nằm phía sau reverse proxy Nginx, phân tích nội dung HTTP request đến và ngăn chặn các truy vấn độc hại.

Khác với nhiều WAF truyền thống sử dụng blacklist, NAXSI tuân theo triết lý "whitelist by default" – nghĩa là mặc định chặn mọi request chứa các ký tự hoặc chuỗi nghi vấn nguy hiểm, trừ khi được định nghĩa là hợp lệ trong rule. Với hệ thống scoring rules linh hoạt và hiệu năng cao, NAXSI có thể phát hiện và ngăn chặn tấn công ngay cả khi không cần tới các rule phức tạp như ModSecurity. Bên cạnh đó, người quản trị hoàn toàn có thể tùy chỉnh hoặc thêm mới các rule phù hợp với ứng dụng cụ thể của mình.

Với ưu điểm là gọn nhẹ, dễ cấu hình, tích hợp chặt với OPNsense, NAXSI là lựa chọn phù hợp cho các hệ thống cần một WAF đơn giản nhưng vẫn đảm bảo mức độ bảo vệ hiệu quả trên tầng ứng dụng.

Như quá trình phát triển Naxsi đã dừng lại và kho lưu trữ sẽ được lưu trữ vì lý do lịch sử. Điều này có nghĩa là sẽ không có bản cập nhật hoặc bản sửa lỗi mới nào được phát hành cho phiên bản.Tuy nhiên, vẫn có thể cập nhật lên phiên bản mới hơn của Naxsi nhờ kho lưu trữ mới tại https://github.com/wargio/naxsi . Kho lưu trữ này đã được bảo trì và cập nhật tích cực với các tính năng mới và bản sửa lỗi.

* 1. **Kiến trúc và cách thức hoạt động**

NAXSI (Nginx Anti XSS & SQL Injection) hoạt động dựa trên nguyên tắc phát hiện theo mẫu rule-based. Khi một HTTP request đến, NAXSI sẽ phân tích nội dung của request và so sánh với danh sách các mẫu (patterns) đã được định nghĩa sẵn để phát hiện các hành vi tấn công phổ biến như SQL Injection, Cross-site Scripting (XSS), Path Traversal, Command Injection,... Nếu một chuỗi dữ liệu trùng khớp với các mẫu nguy hiểm, NAXSI sẽ kiểm tra xem liệu có rule whitelist nào áp dụng cho trường hợp đó không. Nếu không có whitelist, hệ thống sẽ gán điểm (score) cho request đó. Khi tổng điểm vượt ngưỡng đã định trước, request sẽ bị chặn lại hoặc chuyển hướng tới trang cảnh báo.

Một tính năng nổi bật khác của NAXSI là "virtual patching" — một kỹ thuật thường thấy ở các WAF tiên tiến. Tính năng này cho phép "vá ảo" các lỗ hổng trong ứng dụng web mà không cần thay đổi mã nguồn gốc. Điều này cực kỳ hữu ích khi ứng dụng chứa lỗi bảo mật nhưng chưa thể cập nhật phiên bản hoặc sửa lỗi kịp thời. Các "virtual patch" này được biểu diễn dưới dạng rule NAXSI, có thể áp dụng lên toàn bộ request hoặc chỉ những trường dữ liệu cụ thể (field-based patching). Từ đó giúp:

* Ngăn chặn khai thác lỗ hổng trong phần mềm chưa vá.
* Giảm thiểu nguy cơ từ các zero-day exploit.
* Hạn chế bề mặt tấn công bằng cách lọc request đáng ngờ trước khi đến backend.

Đặc biệt, NAXSI áp dụng triết lý “whitelist by default”, tức mặc định sẽ từ chối mọi request không rõ ràng hoặc chưa được phép, và chỉ cho phép những gì đã được định nghĩa rõ trong whitelist. Cách tiếp cận này giúp giảm thiểu rủi ro từ các truy cập không hợp lệ hoặc bất thường.

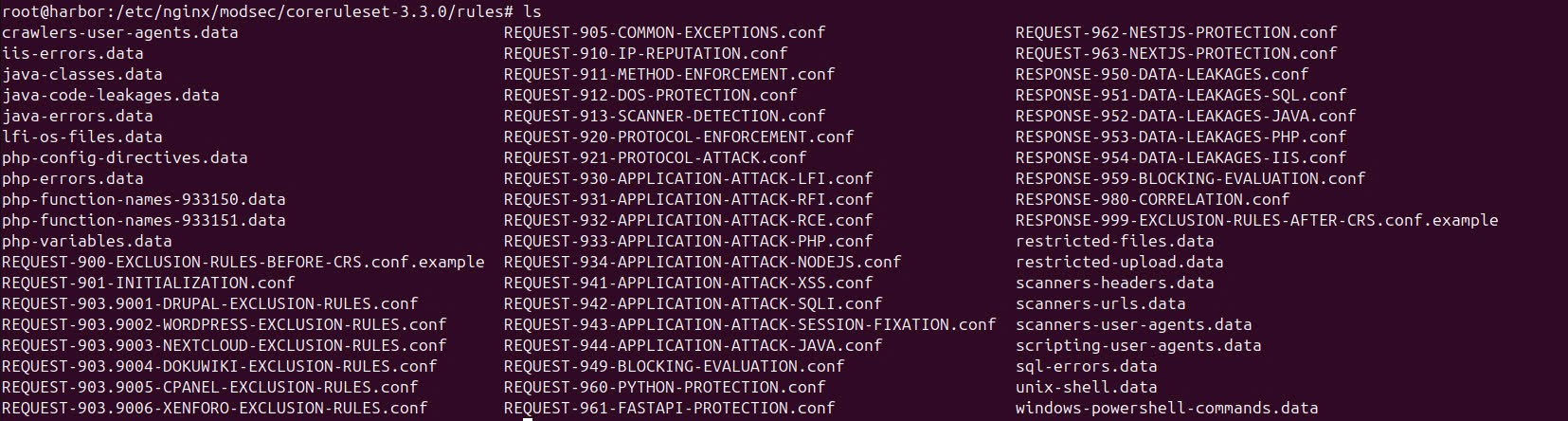
Về hệ thống rule, NAXSI sử dụng 2 loại chính:

* MainRules: gồm các rule mặc định nhằm phát hiện các hành vi tấn công phổ biến. Chúng hoạt động ở mọi context và có thể tự động chặn những payload nguy hiểm.
* BasicRules: thường sử dụng để whitelist các rule cụ thể theo từng location, hoặc bổ sung rule tùy chỉnh theo ứng dụng.

NAXSI cũng tích hợp libinjection để tăng độ chính xác trong việc phát hiện SQLi, và bạn có thể sử dụng nó ngay trong cấu hình location trên NGINX. Tuy nhiên, do vấn đề xung đột bản quyền giữa GPL và BSD, plugin NGINX trên OPNsense không thể tự động cung cấp sẵn các bộ rule như naxsi\_core.rules. Do đó, người quản trị cần tải thủ công từ GitHub (ví dụ: [naxsi\_core.rules](https://github.com/nbs-system/naxsi/blob/master/naxsi_config/naxsi_core.rules)) và nhập vào giao diện cấu hình của plugin.

1. **So sánh phạm vi rule**
   1. **Cú pháp và cấu trúc rule**

Đầu tiên phải nói đến bộ rule, khi ta tải về (deloy) thì ta sẽ nhận được một bộ rule (CRS) là một tập hợp các quy tắc phát hiện tấn công chung để sử dụng với ModSecurity hoặc Tường lửa ứng dụng web tương thích. CRS nhằm mục đích bảo vệ các ứng dụng web khỏi một loạt các cuộc tấn công, bao gồm cả Top Ten OWASP với tối thiểu các false positives.



Đối với các rule tùy chỉnh, phải tạo ID quy tắc riêng và phải là duy nhất. Các trường "ID:" chứa ID của các rules. Đối với các rule tùy chỉnh, nên sử dụng phạm vi sử dụng cục bộ (nội bộ) (xem bên dưới để biết phạm vi ID dành riêng). Không sử dụng các khu vực được chỉ định.

|  |  |
| --- | --- |
| **ID's** | **Usage** |
| **1-99.999** | Reserved for local (internal) use. Use this space at your own discretion, but not for rules distributed to others. |
| **100.000-199.999** | Oracle Rules |
| **200.000-299.999** | Rules of Comodo |
| **300.000-399.999** | Rules from gotroot.com |
| **400.000-419.999** | Unused (available for reservation) |
| **420.000-429.999** | Rules by ScallyWhack |
| **430.000-439.999** | Flameeyes rules |
| **440.000-599.999** | Unused (available for reservation) |
| **600.000-699.999** | Akamai Rules |
| **700.000-799.999** | Rules by Ivan Ristic |
| **900.000-999.999** | Rules of the OWASP ModSecurity Core Rule Set project |
| **1.000.000-1.009.999** | Rules of the Redhat Security Team |
| **1.010.000-1.999.999** | Unused (available for reservation) |
| **2.000.000-2.999.999** | Rules of Trustwave's SpiderLabs Research Team |
| **3.000.000-3.999.999** | Akamai Rules |
| **4.000.000-4.099.999** | Rules by AviNetworks |
| **4.100.000-4.199.999** | Rules by Fastly |
| **4.200.000 and more** | Unused (available for reservation) |

variables

|  |  |
| --- | --- |
| **Variable** | **Usage** |
| ARGS | Is a collection, i.e. all arguments including the POST payload. |
| ARGS\_GET | Contains only query string parameters. |
| ARGS\_POST | Contains arguments from the POST body. |
| FILES | Contains a collection of original file names. Only available for inspected multipart/form-data requests. |
| FULL\_REQUEST | Contains the full request: request line, request headers, and request body. |
| QUERY\_STRING | Contains the query string portion of a request URI. The value in QUERY\_STRING is always provided in the raw state without any URL decoding taking place. |
| REQUEST\_BODY | Contains the raw body of the request. This variable is only available if the URLENCODED request body processor was used, which is the case by default when the content type application/x-www-form-urlencoded is detected, or if the URLENCODED request body parser was forced to be used. |
| REQUEST\_HEADERS | This variable can be used either as a collection of all headers in a request or to validate selected headers. |
| REQUEST\_METHOD | This variable contains the request method used in the transaction. |
| REQUEST\_URI | This variable contains the full request URL including the query string data (e.g. /index.php?p=X). |

Operators – Toán tử so khớp

Trong ModSecurity, Operators là các phép kiểm tra được sử dụng để xác định xem một giá trị cụ thể (như giá trị của header, query string, body...) có khớp với một mẫu hoặc điều kiện cụ thể hay không. Tất cả các toán tử trong ModSecurity đều bắt đầu bằng ký tự @. Các toán tử này có thể là: @rx – So khớp với một biểu thức chính quy (regular expression); @streq – So sánh chuỗi chính xác; @pm – So khớp với một hoặc nhiều từ khóa trong danh sách (phrase match); @gt, @lt, @eq – So sánh số học; @contains – Kiểm tra xem chuỗi có chứa từ khóa; @validateByteRange, @validateUrlEncoding – Dùng để kiểm tra tính hợp lệ dữ liệu đầu vào

ModSecurity sử dụng cú pháp rule dạng ngôn ngữ kiểm tra điều kiện (SecRules), gần giống như lập trình kịch bản (scripting). Một rule của ModSecurity thường có dạng:

*SecRule ARGS "select.+from" "id:1001,phase:2,deny,status:403,msg:'SQL Injection detected'"*

Trong đó:

* ARGS là biến chứa tất cả các tham số trong request.
* "select.+from" là biểu thức chính quy để dò nội dung nguy hiểm.
* Các tham số như id, phase, deny, status, msg quy định hành động, mức xử lý và thông điệp cảnh báo.

Transformations – Hàm biến đổi dữ liệu đầu vào

Trước khi so khớp dữ liệu bằng operator, ModSecurity cho phép thực hiện biến đổi đầu vào thông qua các hàm transform. Điều này rất hữu ích vì dữ liệu đầu vào của người dùng có thể được mã hóa hoặc làm nhiễu để né tránh hệ thống bảo vệ.

Các hàm transform này không làm thay đổi dữ liệu gốc mà chỉ xử lý trên bản sao, sau đó truyền sang operator để so khớp. Một số hàm phổ biến gồm: t:urlDecode – Giải mã URL; t:lowercase – Chuyển tất cả ký tự về chữ thường; t:removeNulls – Loại bỏ ký tự NULL; t:compressWhitespace – Nén khoảng trắng; t:htmlEntityDecode – Giải mã các entity HTML

*SecRule ARGS "@rx union.+select" \*

*"id:4321,phase:2,deny,t:urlDecode,t:lowercase,msg:'Attempted SQL Injection (basic)'"*

Rule này sẽ trước tiên giải mã URL, sau đó chuyển chuỗi về chữ thường trước khi áp dụng regex để kiểm tra dấu hiệu SQL Injection.

Danh sách chi tiết các transformation functions:  
<https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-(v2.x)#transformation-functions>

Actions – Hành động

|  |  |
| --- | --- |
| **SALE** | **Usage** |
| disruptive | Được sử dụng để cho phép modsecurity thực hiện một hành động, ví dụ cho phép hoặc chặn |
| Non disruptive | Làm một cái gì đó, nhưng điều này không có tác động đến dòng xử lý quy tắc. Đặt một biến hoặc thay đổi giá trị của nó là một ví dụ về hành động không giao dịch. Các hành động không phá vỡ có thể xuất hiện trong bất kỳ quy tắc nào, bao gồm bất kỳ quy tắc nào thuộc về một chuỗi. |
| Flow | Những hành động này ảnh hưởng đến luồng quy tắc (ví dụ Skip hoặc SkipAfter). |
| Meta data | Các hành động siêu dữ liệu được sử dụng để cung cấp thêm thông tin về các quy tắc. Ví dụ là ID, REV, Mức độ nghiêm trọng và MSG. |
| Data | Đây không phải là hành động thực sự, chỉ là các container giữ dữ liệu được sử dụng bởi các hành động khác. Ví dụ: hành động trạng thái chứa trạng thái được sử dụng cho khóa (nếu nó diễn ra). |

Quy tắc viết

Dưới đây là một ví dụ về quy tắc đơn giản chặn yêu cầu nếu đường dẫn yêu cầu (sau khi chuẩn hóa thành chữ thường) bằng /index.php.

*SecRule REQUEST\_URI "@streq /index.php” “id:1,phase:1,t:lowercase,deny"*

|  |  |
| --- | --- |
| **SALE** | **Usage** |
| SecRule | Mỗi định nghĩa quy tắc bắt đầu với từ "bí mật" là sự khởi đầu của định nghĩa quy tắc. |
| REQUEST\_URI | Biến này chứa URL yêu cầu đầy đủ bao gồm dữ liệu chuỗi truy vấn (ví dụ: /index.php?p=x). Tuy nhiên, nó không bao giờ chứa một tên miền, ngay cả khi một tên được cung cấp trên dòng yêu cầu. |
| "@streq /index.php" | Thực hiện so sánh chuỗi và trả về true nếu chuỗi tham số giống hệt với chuỗi đầu vào. Trong trường hợp của chúng tôi /index.php. |
| "id:1,phase:1,t:lowercase,deny" | Chuyển đổi tất cả các ký tự thành chữ thường, dừng xử lý quy tắc và chặn giao dịch. |

Trong thực tế, hầu hết các quy tắc (regulations/rules) không đơn giản như vậy. Dưới đây là một ví dụ về một rule hiện đang được sử dụng trong CRS

*SecRule REQUEST\_COOKIES|!REQUEST\_COOKIES:/\_\_utm/|REQUEST\_COOKIES\_NAMES|ARGS\_NAMES|ARGS|XML:/\* "@rx (?i)union.\*?select.\*?from" \*

*"id:942270,\*

*phase:2,\*

*block,\*

*capture,\*

*t:none,t:urlDecodeUni,\*

*msg:'Looking for basic sql injection. Common attack string for mysql, oracle and others',\*

*logdata:'Matched Data: %{TX.0} found within %{MATCHED\_VAR\_NAME}: %{MATCHED\_VAR}',\*

*tag:'application-multi',\*

*tag:'language-multi',\*

*tag:'platform-multi',\*

*tag:'attack-sqli',\*

*tag:'paranoia-level/1',\*

*tag:'OWASP\_CRS',\*

*tag:'capec/1000/152/248/66',\*

*tag:'PCI/6.5.2',\*

*ver:'OWASP\_CRS/3.3.0',\*

*severity:'CRITICAL',\*

*setvar:'tx.sql\_injection\_score=+%{tx.critical\_anomaly\_score}',\*

*setvar:’tx.anomaly\_score\_pl1=+%{tx.critical\_anomaly\_score}'"*

Bạn có thể đã để ý thấy phần *operator* (toán tử) trong rule trên là @rx, theo sau là một chuỗi kỳ lạ trông giống như ký tự loằng ngoằng. Đó chính là regular expression (regex) – một biểu thức chính quy.

*@rx (?i)union.\*?select.\*?from*

Chuỗi ký tự lạ kia chính là biểu thức chính quy (regular expression – viết tắt là regex), một công cụ rất mạnh để dò tìm mẫu (pattern matching) trong văn bản.

Regex này kiểm tra xem chuỗi có chứa:

* từ union (không phân biệt hoa thường – do có (?i)),
* theo sau bởi bất kỳ ký tự nào (.\*?),
* sau đó là từ select,
* rồi lại theo sau bởi bất kỳ ký tự nào,
* và cuối cùng là từ from.

Đây là mẫu tấn công SQL Injection kinh điển: union select ... from ...

Các từ khóa SQL này, khi xuất hiện theo đúng thứ tự như vậy, có thể là dấu hiệu cho thấy một hacker đang cố gắng khai thác lỗ hổng SQL Injection.

ModSecurity hỗ trợ rất nhiều biến (ARGS, REQUEST\_HEADERS, REQUEST\_COOKIES, FILES,...) và điều kiện logic (AND/OR), cho phép người quản trị xây dựng rule phức tạp, kết hợp nhiều điều kiện, kiểm soát sâu các layer của HTTP request. Điều này khiến ModSecurity có tính tùy biến cao và phù hợp với các hệ thống cần bảo vệ nhiều lớp ứng dụng.

Ngược lại, NAXSI sử dụng cú pháp rule đơn giản hơn nhiều. Một rule trong NAXSI thường có dạng:

*MainRule "rx:select.+from" "msg:sql keyword" "mz:BODY|URL|ARGS" "s:$SQL" "id:1001"*

Trong đó:

* "rx:..." là biểu thức regex để bắt nội dung đáng ngờ.
* "mz" (match zone) quy định vùng kiểm tra như BODY, URL, ARGS,...
* "s:$SQL" gán điểm số cho nhóm tấn công SQLi.
* "id" là định danh của rule.

NAXSI không trực tiếp chặn request ngay khi match rule, mà thay vào đó, mỗi rule tăng điểm cho loại tấn công tương ứng. Sau đó, nếu điểm vượt ngưỡng định sẵn (checkrules), request mới bị chặn. Cách làm này giúp tránh false positive và linh hoạt trong xử lý hành vi phức tạp.

NAXSI hoạt động dựa trên ba loại quy tắc chính: MainRule – Quy tắc phát hiện mẫu tấn công; BasicRule – Quy tắc whitelist, loại trừ các trường hợp hợp lệ khỏi bị chặn; CheckRule – Quy tắc hành động, xác định khi nào một request nên bị chặn dựa trên điểm số tích lũy.

Và cũng như ID của rule modsecurity thì Naxsi cũng có được phân chia rõ ràng

|  |  |
| --- | --- |
| **Rule** | **ID** |
| INTERNAL RULES IDS  *Quy tắc nội bộ, xử lý các lỗi định dạng hoặc yêu cầu bất thường.* | 1-999 |
| QL Injections IDs *Phát hiện SQL Injection.* | 1000-1099 |
| OBVIOUS RFI Ids *Phát hiện Remote File Inclusion (RFI).* | 1100-1199 |
| Directory traversal IDs *Phát hiện Directory Traversal.* | 1200-1299 |
| Cross Site Scripting IDs *Phát hiện Cross-Site Scripting (XSS).* | 1300-1399 |
| Evading tricks IDs  *Phát hiện các kỹ thuật né tránh (Evasion).* | 1400-1500 |
| File uploads  *Phát hiện các tệp tải lên nguy hiểm.* | 1500-1600 |

Đặc biệt bộ rule internal không hiện ra khi xem trên trên firewall mà phải xem trong file /etc/nginx/naxsi\_core.rules . Còn rule tự viết sẽ nằm ở /etc/nginx/naxsi\_custom.rules

Với mỗi bộ quy tắc sẽ có một cấu trúc riêng:

* 1. MainRule – Phát hiện mẫu tấn công

Cú pháp:

*MainRule "str:pattern" "msg:description" "mz:match\_zone" "s:$ATTACK\_TYPE:score" id:ID;*

Một ví dụ về mainrule:

*MainRule "str:select" "msg:SQL keyword detected" "mz:ARGS|BODY" "s:$SQL:5" id:1001;*

Giải thích:

* str:select: Tìm chuỗi "select" trong dữ liệu.
* msg: Mô tả quy tắc.
* mz: Vùng dữ liệu kiểm tra, như ARGS, BODY, URL, v.v.
* s:$SQL:5: Nếu khớp, tăng điểm SQL lên 5.
* id:1001: ID duy nhất của quy tắc.
  1. BasicRule – Whitelist

Cú pháp tương ứng:

*BasicRule wl:ID "mz:match\_zone" id:ID;*

Ví dụ:

*BasicRule wl:1001 "mz:ARGS\_VAR:username" id:2001;*

Giải thích:

* wl:1001: Whitelist quy tắc có ID 1001.
* mz:ARGS\_VAR:username: Áp dụng whitelist cho biến ARGS có tên "username".
* id:2001: ID duy nhất của quy tắc whitelist.
  1. CheckRule – Hành động dựa trên điểm số

Tương tự thì phần này cũng có cú pháp riêng:

*CheckRule "$ATTACK\_TYPE >= threshold" BLOCK;*

Ví dụ:

*CheckRule "$SQL >= 8" BLOCK;*

Giải thích: Nếu tổng điểm của loại tấn công SQL lớn hơn hoặc bằng 8, request sẽ bị chặn.

Về cách cấu hình NAXSI trong OPNsense

Trên OPNsense, bạn có thể cấu hình NAXSI thông qua giao diện web hoặc chỉnh sửa trực tiếp các tệp cấu hình, sau khi kích hoạt NAXSI trong phần cấu hình NGINX. Ta có thể chỉ định các tệp quy tắc trong cấu hình NGINX:

*include /usr/local/etc/nginx/naxsi\_core.rules;*

*include /usr/local/etc/nginx/naxsi.rules;*

Sau đó thiết lập các quy tắc hành động trong naxsi.rules:

*SecRulesEnabled;*

*DeniedUrl "/error.html";*

*CheckRule "$SQL >= 8" BLOCK;*

*CheckRule "$XSS >= 8" BLOCK;*

Tuy nhiên, NAXSI không có các điều kiện logic phức tạp như AND/OR, không thể kiểm tra cookie hoặc multipart upload chi tiết như ModSecurity. Thay vào đó, NAXSI thiên về hướng “nhẹ”, dễ triển khai, đơn giản, ưu tiên tốc độ xử lý cao, phù hợp với OPNsense hoặc các môi trường resource thấp.

Đây là một bảng so sánh độ phức tạp và tính linh hoạt của chúng

|  |  |  |
| --- | --- | --- |
| Tiêu chí | ModSecurity | NAXSI |
| Mức độ phức tạp | Cao – cần kiến thức về rule syntax, logic | Thấp – dễ học, dễ viết |
| Độ linh hoạt | Rất cao – hỗ trợ regex nâng cao, logic nhiều lớp | Vừa phải – chỉ rule đơn giản, dựa trên điểm số |
| Vùng kiểm tra (match zone) | Rất đa dạng – header, body, cookie, file, multipart,... | Giới hạn hơn – chủ yếu là URL, ARGS, BODY |
| Hiệu suất xử lý | Trung bình – tốn tài nguyên hơn | Cao – phù hợp cho firewall/embedded như OPNsense |
| Triết lý mặc định | Cho phép -> lọc theo blacklist (phổ biến) | Chặn tất cả -> cho phép theo whitelist (an toàn hơn) |
| Rule mẫu sẵn có | Rất nhiều (OWASP CRS) | Ít hơn – có thể tự bổ sung |

* 1. **Khả năng biểu diễn logic**

Logic phức tạp trong ModSecurity:

ModSecurity cung cấp khả năng biểu diễn logic phức tạp thông qua cú pháp SecRule của nó. Hệ thống này cho phép kết hợp nhiều biến, toán tử và hành động trong một rule, tạo nên một ngôn ngữ rule mạnh mẽ. ModSecurity cho phép:

* Sử dụng các biểu thức chính quy (regex) phức tạp
* Kết hợp nhiều điều kiện trong một rule thông qua các toán tử logic (AND, OR)
* Tạo các biến tạm thời để lưu trữ kết quả trung gian
* Thực hiện các hành động có điều kiện dựa trên nhiều yếu tố khác nhau
* Kiểm tra nhiều phần của HTTP request/response bao gồm headers, cookies, body,…

Ví dụ về một rule ModSecurity phức tạp:

*SecRule REQUEST\_HEADERS:User-Agent "@rx (?i:nikto|sqlmap|acunetix)" \*

*"id:1000,\*

*phase:1,\*

*deny,\*

*log,\*

*msg:'Phát hiện công cụ quét lỗ hổng bảo mật'"*

Logic đơn giản hóa trong NAXSI:

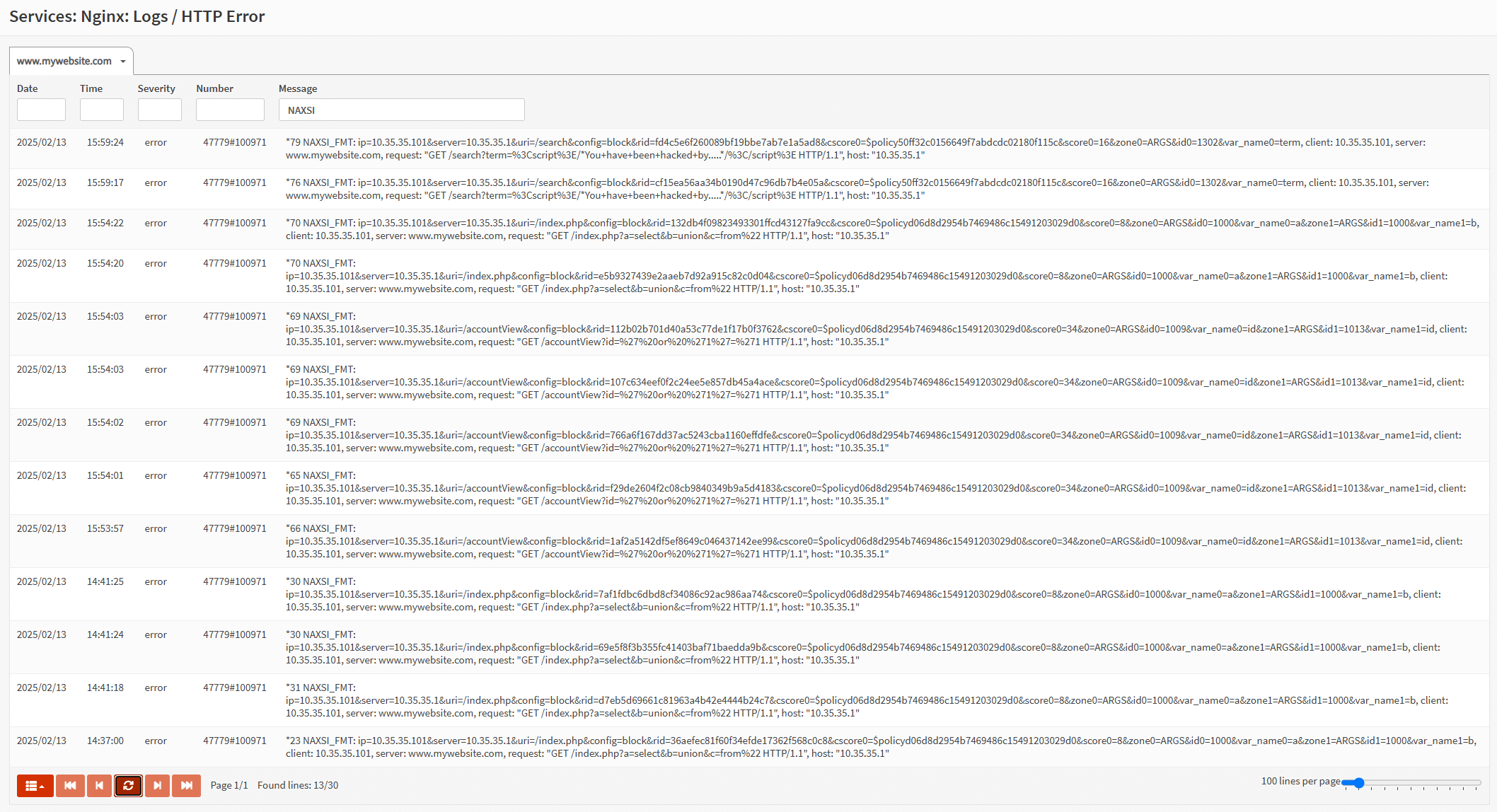
NAXSI sử dụng cách tiếp cận đơn giản hóa với triết lý "whitelist by default". Phương pháp này dựa trên một hệ thống tính điểm đơn giản và chỉ tập trung vào việc phát hiện các mẫu nguy hiểm phổ biến. NAXSI:

* Sử dụng cú pháp đơn giản hơn nhiều so với ModSecurity
* Dựa vào hệ thống scoring để đánh giá mức độ nguy hiểm
* Tập trung chủ yếu vào việc phát hiện XSS và SQL Injection
* Quyết định chặn dựa trên ngưỡng điểm tổng hợp

Theo tài liệu từ OPNsense, NAXSI "chỉ xem xét một số tiêu chí đơn giản, chẳng hạn như 99% các mẫu đã biết liên quan đến các lỗ hổng trang web". Ví dụ, theo nguyên tắc đơn giản, các ký tự như < , | hoặc từ khóa drop không nên xuất hiện trong URI.

Ví dụ về rule NAXSI:

*MainRule negative "rx:multipart/form-data|application/x-www-form-urlencoded" "msg:Content-Type không phải dạng form được chấp nhận" "mz:$HEADERS\_VAR:Content-Type" "s:$EVADE:4" id:1402;*



Ưu nhược điểm của mỗi cách tiếp cận:

ModSecurity là một Web Application Firewall mạnh mẽ với khả năng phân tích và xử lý các request phức tạp nhờ hệ thống rule linh hoạt. Ưu điểm lớn nhất của ModSecurity là khả năng tùy biến cao, cho phép xây dựng các rule phức tạp với logic điều kiện nhiều lớp, sử dụng đa dạng biến và toán tử. Nhờ đó, ModSecurity có thể bảo vệ ứng dụng web trước nhiều loại tấn công, kể cả các kỹ thuật khai thác nâng cao. Ngoài ra, cộng đồng hỗ trợ lớn và bộ rule OWASP CRS cũng giúp việc triển khai dễ dàng hơn. Tuy nhiên, nhược điểm của ModSecurity nằm ở chính sự phức tạp của nó – người dùng cần có kiến thức sâu về bảo mật web và cú pháp rule để cấu hình hiệu quả. Nếu không được tinh chỉnh cẩn thận, ModSecurity có thể gây ra false positive và ảnh hưởng đến hiệu suất hệ thống do phải xử lý hàng trăm rule phức tạp mỗi request.

NAXSI, ngược lại, theo đuổi triết lý thiết kế đơn giản hơn với cấu trúc rule nhẹ và dễ hiểu. Ưu điểm của NAXSI là dễ học, dễ triển khai và đặc biệt phù hợp cho các hệ thống như OPNsense nhờ tốc độ xử lý nhanh. Với cơ chế mặc định chặn (default DROP) và whitelist chủ động, NAXSI giúp giảm thiểu bề mặt tấn công ngay từ đầu mà không cần quá nhiều rule phức tạp. Tuy nhiên, tính linh hoạt của NAXSI không cao – nó không hỗ trợ logic phức tạp, thiếu các biến nâng cao và phụ thuộc nhiều vào whitelist để tránh false positive. Điều này khiến NAXSI ít phù hợp với các môi trường web có nhiều hành vi động hoặc yêu cầu bảo mật đặc thù. Ngoài ra, quá trình xây dựng whitelist ban đầu cũng cần theo dõi sát sao request thực tế để tối ưu hóa.

* 1. **Phạm vi bảo vệ**

**Phạm vi bảo vệ**

**Loại tấn công được phát hiện bởi ModSecurity**

ModSecurity có khả năng phát hiện và ngăn chặn một phạm vi rộng các loại tấn công web do cấu trúc rule phức tạp và hỗ trợ OWASP Core Rule Set. Các loại tấn công chính được phát hiện bao gồm:

1. **Tấn công injection**:
   * SQL Injection
   * NoSQL Injection
   * Command Injection
   * LDAP Injection
   * XML Injection
2. **Tấn công Cross-site**:
   * Cross-site Scripting (XSS)
   * Cross-site Request Forgery (CSRF)
3. **Các tấn công khác**:
   * Local/Remote File Inclusion (LFI/RFI)
   * Path Traversal
   * Server-side Request Forgery (SSRF)
   * HTTP Protocol Violations
   * Session Fixation
   * Botnet và công cụ tự động (scanner, fuzzers)
   * Request Flooding
   * Shell Injection

ModSecurity có khả năng kiểm soát truy cập dựa trên nhiều yếu tố như IP, User-Agent, cookie và các thông tin session, giúp phòng chống rất nhiều loại tấn công phức tạp.

**Phạm vi bảo vệ của NAXSI**

NAXSI có phạm vi bảo vệ hẹp hơn, tập trung chủ yếu vào các tấn công phổ biến nhất. Theo tên gọi "Nginx Anti XSS & SQL Injection", NAXSI chủ yếu tập trung vào:

1. **Tấn công chính**:
   * SQL Injection (SQLi)
   * Cross-site Scripting (XSS)
2. **Tấn công phổ biến khác**:
   * Command Injection cơ bản
   * Một số dạng Path Traversal đơn giản
   * Remote File Inclusion (RFI) thông qua URL

NAXSI sử dụng libinjection, một thư viện phát hiện SQLi và XSS hiệu quả, tuy nhiên không có khả năng phát hiện nhiều loại tấn công phức tạp như ModSecurity. Sức mạnh của NAXSI đến từ triết lý "default DROP", nghĩa là mọi request đều bị từ chối trừ khi được whitelist cụ thể.

**Độ phủ theo OWASP Top 10**

|  |  |  |
| --- | --- | --- |
| **OWASP Top 10** | **ModSecurity (với CRS)** | **NAXSI** |
| A01 – Broken Access Control | Cao | Thấp |
| A02 – Cryptographic Failures | Trung bình | Rất thấp |
| A03 – Injection | Rất cao | Cao (chủ yếu SQLi, XSS) |
| A04 – Insecure Design | Thấp (tùy thiết kế) | Rất thấp |
| A05 – Security Misconfiguration | Trung bình | Thấp |
| A06 – Vulnerable and Outdated Components | Thấp | Rất thấp |
| A07 – Identification and Authentication Failures | Trung bình | Thấp |
| A08 – Software and Data Integrity Failures | Trung bình | Thấp |
| A09 – Security Logging and Monitoring Failures | Cao (log chi tiết) | Trung bình |
| A10 – Server-Side Request Forgery (SSRF) | Cao | Trung bình |

* 1. ***Khả năng tùy chỉnh***
* *Khả năng tạo rule tùy chỉnh trong ModSecurity*
* *Khả năng tạo rule tùy chỉnh trong NAXSI*

Trong quá trình triển khai các hệ thống tường lửa ứng dụng web (WAF), khả năng tạo rule tùy chỉnh là một yếu tố cực kỳ quan trọng, giúp người quản trị có thể thích ứng với từng môi trường thực tế. Cả ModSecurity và NAXSI đều hỗ trợ viết rule riêng, nhưng cách tiếp cận và mức độ linh hoạt của mỗi công cụ có sự khác biệt rõ rệt.

“*ModSecurity – Viết rule như lập trình viên viết code*”

ModSecurity nổi bật bởi khả năng viết rule cực kỳ linh hoạt, có thể ví như một ngôn ngữ lập trình thu nhỏ dành riêng cho việc xử lý request và response HTTP. Người dùng có thể viết rule bằng cú pháp SecRule, kết hợp nhiều điều kiện, toán tử và hành động khác nhau.

Ví dụ, có thể viết một rule để chặn người dùng từ Việt Nam truy cập vào hệ thống chỉ bằng vài dòng:

*SecRule GEO:COUNTRY\_CODE "@streq VN" \*

*"id:4000, phase:1, deny, status:403, log, msg:'Blocked access from Vietnam'"*

ModSecurity xử lý dữ liệu theo từng giai đoạn – từ khi nhận được header của request cho đến khi xử lý xong response. Điều này giúp rule có thể can thiệp ở từng bước cụ thể, như kiểm tra nội dung tải lên, lọc nội dung phản hồi từ server, hoặc theo dõi hành vi người dùng xuyên suốt các phiên.

Ngoài ra, ModSecurity còn hỗ trợ tạo biến tùy chỉnh và bộ sưu tập lưu trữ dữ liệu, giúp theo dõi IP đáng ngờ, đếm số lần request từ một địa chỉ nhất định, hoặc lưu lại trạng thái người dùng. Đặc biệt, nó còn hỗ trợ các toán tử chuyên dụng như kiểm tra thẻ tín dụng, kiểm tra định dạng file, hoặc xác thực chuỗi byte hợp lệ trong request.

Tuy nhiên, do tính linh hoạt cao, việc viết và quản lý rule trong ModSecurity có thể khá phức tạp, đòi hỏi người triển khai phải hiểu rõ cấu trúc HTTP và có kiến thức nhất định về biểu thức chính quy cũng như logic rule.

“*NAXSI – Cấu hình đơn giản, dễ hiểu, phù hợp hệ thống nhẹ*”

Trái ngược với ModSecurity, NAXSI thiết kế rule theo hướng đơn giản hơn, gần giống như một bộ lọc chữ ký (signature-based), với nguyên tắc hoạt động dựa trên điểm số. Các rule được định nghĩa trước để phát hiện các mẫu nguy hiểm (như ký tự SQL, mã JavaScript...) trong request. Mỗi khi một request chứa chuỗi đáng ngờ, NAXSI sẽ cộng điểm cho request đó. Nếu tổng điểm vượt ngưỡng, request sẽ bị chặn.

Một điểm thú vị của NAXSI là có thể **whitelist** rule theo từng địa chỉ URL hoặc tham số cụ thể, giúp giảm cảnh báo sai (false positive) mà không ảnh hưởng đến các khu vực khác của ứng dụng.

Ví dụ:

*MainRule "str:admin" "msg:URL chứa từ khóa admin" "mz:URL" "s:DROP" id:4242;*

*BasicRule wl:1001,1010,1011 "mz:$URL:/login.php|ARGS";*

Trong ví dụ trên, rule đầu tiên chặn bất kỳ request nào chứa từ "admin" trong URL, còn rule thứ hai là whitelist, bỏ qua một số rule cụ thể khi người dùng gửi dữ liệu lên trang /login.php.

NAXSI cũng hỗ trợ Learning Mode – chế độ học tự động giúp thu thập dữ liệu và đề xuất whitelist phù hợp dựa trên lưu lượng hợp lệ, giúp đơn giản hóa quá trình tinh chỉnh rule.

1. **So sánh các policy**
   1. **Default Policy**

Trong các hệ thống WAF (Web Application Firewall), chính sách mặc định đóng vai trò rất quan trọng – đặc biệt trong giai đoạn ban đầu triển khai. Việc hiểu rõ cách mỗi công cụ phản ứng với lưu lượng truy cập ngay từ đầu giúp người quản trị có thể lựa chọn phù hợp với nhu cầu hệ thống, cũng như chuẩn bị các cấu hình cần thiết để đạt hiệu quả bảo vệ tốt nhất. Cả ModSecurity và NAXSI đều có triết lý riêng khi nói đến chính sách mặc định.

ModSecurity – “Cảnh báo chứ chưa chặn”

Ngay sau khi cài đặt, ModSecurity không thực hiện bất kỳ hành động chặn nào. Mặc định, nó hoạt động ở chế độ “Detection Only” – tức là chỉ giám sát và ghi log các hành vi đáng ngờ mà không có bất kỳ phản ứng nào đối với người dùng.

Một điểm đáng lưu ý là: ModSecurity không đi kèm bộ rule mặc định nào được bật sẵn. Tức là, nếu chỉ cài đặt mà không cấu hình thêm, hệ thống sẽ hầu như không có khả năng phát hiện hoặc ngăn chặn tấn công. Để hoạt động hiệu quả, người dùng cần tích hợp thêm OWASP Core Rule Set (CRS) hoặc tự viết rule tùy chỉnh.

Ví dụ về phần cấu hình mặc định của ModSecurity thường thấy:

*SecRuleEngine On*

*SecRequestBodyAccess On*

*SecResponseBodyAccess On*

*SecRequestBodyLimit 13107200*

*SecRequestBodyNoFilesLimit 131072*

*SecResponseBodyLimit 524288*

Như vậy, mặc dù cho phép tùy biến rất sâu, nhưng ModSecurity yêu cầu người triển khai phải chủ động cấu hình để hệ thống bắt đầu phát huy hiệu quả thực sự.

NAXSI – “Chặn trước, cho sau”

Khác hoàn toàn với ModSecurity, NAXSI được thiết kế theo triết lý ngược lại – “Default DROP”. Điều đó có nghĩa là mọi request có dấu hiệu nguy hiểm sẽ bị chặn ngay lập tức, trừ khi đã được cấu hình whitelist rõ ràng. Đây là một chiến lược bảo vệ có phần “cứng rắn” hơn, nhưng cũng giúp hệ thống giảm thiểu nguy cơ bị tấn công từ đầu.

Ngay sau khi cài đặt, NAXSI đi kèm với một bộ rule phát hiện sẵn, tập trung chủ yếu vào các loại tấn công phổ biến như SQL Injection và Cross-site Scripting (XSS). Các rule này sẽ dò tìm các chuỗi mẫu nguy hiểm trong URL, query string, hoặc nội dung của request và áp dụng hệ thống tính điểm. Nếu tổng điểm vượt ngưỡng định sẵn, request sẽ bị từ chối.

Tuy nhiên, vì triết lý bảo vệ “mạnh tay”, NAXSI thường gây ra false positive (chặn nhầm), nhất là khi mới triển khai với các ứng dụng có logic phức tạp. Để giải quyết vấn đề này, NAXSI cung cấp Learning Mode – một chế độ học tự động cho phép ghi nhận các request hợp lệ để người quản trị tạo whitelist phù hợp. Điều này giúp giảm lỗi chặn nhầm mà vẫn giữ được mức bảo vệ cao.

Trích từ tài liệu chính thức của OPNsense:  
“*NAXSI hoạt động như một firewall DROP mặc định, đòi hỏi việc thêm các rule ACCEPT cụ thể để trang web mục tiêu hoạt động đúng.”*

* 1. **OWASP Core Rule Set (CRS)**

ModSecurity được xem là một trong những WAF mã nguồn mở mạnh nhất hiện nay, và một trong những điểm mạnh nổi bật chính là khả năng tích hợp sâu với bộ quy tắc OWASP Core Rule Set (CRS). Đây là một tập hợp rule do cộng đồng OWASP phát triển nhằm phát hiện và ngăn chặn các loại tấn công phổ biến vào ứng dụng web. Việc tích hợp CRS vào ModSecurity gần như là một thao tác mặc định, vì hai bên được thiết kế để hoạt động tương thích với nhau hoàn toàn.

CRS trong ModSecurity được tổ chức theo từng nhóm cụ thể, mỗi nhóm đại diện cho một loại hình tấn công như SQL Injection, XSS, LFI, RCE,... Ngoài ra, CRS còn phân cấp độ bảo vệ theo hệ thống gọi là “Paranoia Level” (từ mức 1 đến 4), cho phép người quản trị dễ dàng điều chỉnh độ "nhạy" của WAF tùy theo nhu cầu thực tế: mức thấp để giảm false positive, mức cao để tăng cường bảo vệ chuyên sâu. Hơn nữa, CRS sử dụng cơ chế tính điểm (scoring system) để đánh giá mức độ nghiêm trọng của request trước khi quyết định có nên chặn hay không. Đây là một hệ thống tinh vi, phù hợp với những ứng dụng phức tạp cần bảo vệ nhiều lớp.

Một điểm cộng lớn khác là CRS được cập nhật liên tục bởi cộng đồng OWASP, giúp đảm bảo khả năng phát hiện các kiểu tấn công mới. Người dùng cũng dễ dàng tùy chỉnh rule, bật hoặc tắt từng nhóm, thay đổi ngưỡng điểm chặn hoặc thêm các ngoại lệ nhằm tránh false positive khi triển khai thực tế. Việc tích hợp CRS chỉ cần một vài dòng cấu hình, ví dụ như *Include /path/to/crs/crs-setup.conf*, rất đơn giản và phổ biến trong các hệ thống sử dụng ModSecurity.

*# Kích hoạt CRS trong ModSecurity*

*Include /usr/local/owasp-crs/crs-setup.conf*

*Include /usr/local/owasp-crs/rules/\*.conf*

Ngược lại, NAXSI không được thiết kế để tích hợp trực tiếp với OWASP CRS. Lý do đến từ sự khác biệt căn bản trong cách viết rule và triết lý hoạt động. NAXSI sử dụng cú pháp rule riêng, đơn giản hơn nhiều so với ModSecurity nhưng cũng kém linh hoạt hơn. Do đó, nếu muốn sử dụng ý tưởng hoặc logic từ CRS, người quản trị buộc phải chuyển đổi thủ công các rule sang cú pháp tương thích với NAXSI, điều này không chỉ tốn công mà còn dễ xảy ra sai sót nếu không cẩn thận.

Thay vì CRS, NAXSI đi kèm với bộ quy tắc mặc định có tên là naxsi\_core.rules. Bộ quy tắc này chủ yếu tập trung vào việc phát hiện các mẫu nguy hiểm phổ biến như XSS và SQL Injection, nhưng lại không bao phủ được toàn bộ phổ tấn công như CRS. Việc cập nhật rule cũng không thường xuyên như OWASP CRS và phạm vi bảo vệ cũng bị giới hạn đáng kể.

Theo tài liệu chính thức từ OPNsense, do xung đột giữa giấy phép sử dụng (GPL của CRS và BSD của OPNsense), người dùng không thể cài trực tiếp bộ rule CRS vào NAXSI thông qua plugin mặc định. Tuy nhiên, vẫn có thể import thủ công nếu cần thiết, nhưng việc này yêu cầu kỹ năng và sự hiểu biết nhất định.

Về khả năng bảo vệ theo OWASP Top 10, có thể thấy rõ sự khác biệt giữa hai công cụ. ModSecurity khi tích hợp với CRS có khả năng bao phủ rất rộng, đặc biệt ở các hạng mục như Injection, Broken Access Control, SSRF hay Logging/Monitoring. Trong khi đó, NAXSI tỏ ra yếu hơn về mặt tổng thể, tuy vẫn làm tốt ở nhóm tấn công cụ thể như SQLi và XSS. Điều này cho thấy ModSecurity thích hợp hơn với các hệ thống phức tạp cần bảo vệ toàn diện, còn NAXSI phù hợp cho những trường hợp đơn giản, dễ kiểm soát.

* 1. **Các bộ rule phổ biến**

Một trong những yếu tố then chốt quyết định hiệu quả và tính thực tiễn của một giải pháp WAF chính là hệ sinh thái bộ quy tắc (rule set) mà nó hỗ trợ, cũng như mức độ hoạt động của cộng đồng phát triển đứng sau. Về mặt này, ModSecurity và NAXSI có sự khác biệt rõ rệt.

Đối với ModSecurity: ModSecurity sở hữu một hệ sinh thái cực kỳ phong phú với nhiều bộ rule được phát triển không chỉ bởi cộng đồng mã nguồn mở mà còn bởi các công ty bảo mật hàng đầu thế giới. Trong số đó, OWASP Core Rule Set (CRS) là bộ quy tắc phổ biến nhất và gần như được xem là "tiêu chuẩn vàng" khi triển khai ModSecurity. CRS cung cấp khả năng phát hiện một loạt các kiểu tấn công phổ biến như SQL Injection, XSS, Local File Inclusion, Command Injection, và nhiều loại tấn công ứng dụng web khác. Bộ quy tắc này được duy trì và cập nhật thường xuyên bởi cộng đồng OWASP trên GitHub (<https://github.com/coreruleset/coreruleset>), đảm bảo khả năng ứng phó kịp thời với các mối đe dọa mới.

Ngoài OWASP CRS, ModSecurity còn hỗ trợ nhiều bộ rule khác như:

* Trustwave SpiderLabs Rules: Đây là bộ quy tắc được phát triển bởi chính đội ngũ đứng sau ModSecurity. Nó tập trung vào các mối đe dọa mới và các kỹ thuật tấn công tiên tiến, mặc dù một số rule yêu cầu đăng ký sử dụng.
* Comodo WAF Rules: Do Comodo phát triển, bộ quy tắc này nhắm vào việc bảo vệ các ứng dụng web phổ biến và được tối ưu cho các tình huống tấn công thực tế trong doanh nghiệp.
* Atomicorp Rules (ASSP): Là một giải pháp thương mại, bộ rule của Atomicorp được đánh giá cao về chất lượng, độ bao phủ và tần suất cập nhật (thường là hàng ngày), rất phù hợp với các tổ chức có yêu cầu bảo mật cao.
* AQTRONIX WebKnight Rules: Tập trung chủ yếu vào môi trường Microsoft, bộ quy tắc này dễ dàng tích hợp với ModSecurity trong các hệ thống chạy IIS hoặc liên quan đến công nghệ của Microsoft.

Bên cạnh các bộ quy tắc lớn trên, ModSecurity còn có các rule chuyên biệt được thiết kế riêng cho các CMS như WordPress, Joomla, Drupal... giúp bảo vệ tối ưu trong các tình huống cụ thể.

Đối với NAXSI: So với ModSecurity, hệ sinh thái rule của NAXSI có phần khiêm tốn hơn. Bộ rule mặc định là naxsi\_core.rules, được phát triển bởi chính dự án NAXSI, chủ yếu tập trung vào việc phát hiện các mẫu tấn công XSS và SQL Injection. Dù có thể mở rộng, nhưng số lượng và tính toàn diện của các rule vẫn chưa thể so sánh với CRS của ModSecurity. Rule này hiện vẫn được lưu trữ công khai trên GitHub tại địa chỉ: <https://github.com/nbs-system/naxsi/blob/master/naxsi_config/naxsi_core.rules>.

Ngoài ra, một số bộ rule khác dành cho NAXSI cũng đáng chú ý như:

* NBS System Rules: Do chính nhóm phát triển NAXSI ban đầu cung cấp, mở rộng phạm vi bảo vệ cho các ứng dụng web phổ biến “https://github.com/nbs-system/naxsi-rules”.
* OPNsense Rules: Đây là bộ rule được tích hợp sẵn trong plugin NAXSI của OPNsense, được tinh chỉnh để phù hợp với môi trường hệ thống OPNsense cụ thể. Bao gồm default rule từ 1-999 và các rule tích hợp từ ngoài.
* Whitelist Rules: Các bộ quy tắc dạng whitelist được phát triển để giảm false positive khi triển khai cho các CMS như WordPress, Drupal... giúp WAF "hiểu" các hành vi hợp lệ của ứng dụng để không vô tình chặn và có sẵn khi cài về để vào tùy chỉnh.
* Libinjection Integration: NAXSI cũng hỗ trợ thư viện libinjection – một công cụ mạnh mẽ để phát hiện các payload SQL Injection và XSS thông qua phân tích cú pháp sâu, giúp cải thiện đáng kể độ chính xác khi phát hiện các loại tấn công này “https://github.com/nbs-system/naxsi/wiki/libinjection-integration”.

Tuy nhiên, theo tài liệu chính thức từ OPNsense, dự án NAXSI gốc đã bị bỏ rơi kể từ ngày 8 tháng 11 năm 2023. Hiện nay, chỉ còn một nhà phát triển duy nhất là Giovanni, một người đam mê cryptography và kỹ thuật reverse tiếp tục duy trì và cập nhật phần mềm như một dự án cá nhân. Điều này khiến cộng đồng hỗ trợ NAXSI hiện tại khá nhỏ và không còn hoạt động sôi nổi như trước.

Nếu so sánh cộng đồng hỗ trợ thì ModSecurity sở hữu một cộng đồng lớn, có mặt khắp các nền tảng như GitHub, StackOverflow, các diễn đàn chuyên biệt và cả các tài liệu chuyên sâu do các chuyên gia bảo mật viết. Dự án cũng được hỗ trợ bởi các tổ chức lớn như Trustwave và OWASP nên luôn được cập nhật thường xuyên và có tính thực tiễn cao. Ngược lại, cộng đồng phát triển NAXSI nhỏ hơn rất nhiều. Tài liệu hướng dẫn ít và chủ yếu mang tính cơ bản. Các thảo luận hỗ trợ chủ yếu diễn ra trên GitHub Issues, với tần suất cập nhật không đều. Mặc dù NAXSI vẫn là một lựa chọn WAF nhẹ, đơn giản và dễ triển khai, nhưng sự hạn chế về tài nguyên cộng đồng và các bộ rule đi kèm khiến nó kém phù hợp với các hệ thống lớn hoặc có nhu cầu bảo vệ toàn diện.

* 1. **Khả năng tích hợp policy từ bên thứ ba**

Khả năng tích hợp rule từ bên thứ ba đóng vai trò quan trọng trong việc mở rộng và cập nhật nhanh chóng khả năng phát hiện – phòng thủ của các hệ thống WAF, nhất là khi đối mặt với các mối đe dọa mới hoặc các dạng tấn công zero-day chưa được xử lý trong rule mặc định.

ModSecurity thể hiện khả năng tích hợp rule từ bên thứ ba một cách rất linh hoạt. Nhờ có thiết kế mở và hệ sinh thái mạnh mẽ, ModSecurity hỗ trợ dễ dàng việc nhập rule từ các nguồn ngoài như OWASP CRS, Comodo, Trustwave SpiderLabs hay Atomicorp. Hầu hết các bộ rule này có thể được tải xuống, cài đặt và cập nhật thông qua các script hoặc quản lý tập trung qua các nền tảng như ModSecurity Rule Management Tools (ví dụ: CRS Updater).

Đặc biệt, người dùng có thể tạo rule tùy chỉnh hoặc sử dụng các rule được chia sẻ trong cộng đồng để xử lý các tình huống đặc thù, như bảo vệ CMS (WordPress, Joomla), API gateway hoặc các ứng dụng microservice phức tạp. Ngoài ra, nhiều giải pháp thương mại cũng cung cấp giao diện quản trị rule tập trung, tích hợp ModSecurity với các nền tảng như SIEM, Threat Intelligence Platform để tự động cập nhật rule dựa trên IOC (Indicator of Compromise) thu thập từ thực địa.

Tóm lại, với ModSecurity, việc tích hợp rule bên ngoài không chỉ dễ thực hiện mà còn được hỗ trợ bởi một hệ sinh thái phát triển mạnh và liên tục mở rộng.

Ngược lại, NAXSI có khả năng tích hợp rule từ bên thứ ba khá hạn chế. Dù vẫn hỗ trợ việc thêm rule tùy chỉnh hoặc nhập từ repository như naxsi\_core.rules, nhưng quy trình này phần lớn vẫn mang tính thủ công. Việc cập nhật rule từ cộng đồng hoặc các bên thứ ba yêu cầu người dùng hiểu rõ cấu trúc rule của NAXSI – vốn có cú pháp và logic khác biệt so với ModSecurity (sử dụng whitelist model thay vì blacklist).

Ngoài bộ rule chính thức, một số nguồn như NBS System hoặc rule tích hợp sẵn trong OPNsense có thể được sử dụng, tuy nhiên số lượng rất hạn chế và không cập nhật thường xuyên. Một số rule whitelist dành cho CMS cũng có sẵn, nhưng thường cần điều chỉnh phù hợp với từng hệ thống cụ thể để tránh false positive.

Bên cạnh đó, do NAXSI đã không còn được duy trì bởi đội ngũ phát triển chính (từ tháng 11/2023), cộng đồng và các nguồn rule bên ngoài cũng trở nên rời rạc và khó tích hợp hơn theo cách tự động hóa hoặc có hỗ trợ quản lý tập trung.

1. **Quản lý và giám sát**

Việc quản lý và giám sát hiệu quả là yếu tố quan trọng giúp đảm bảo rằng các hệ thống WAF không chỉ phát hiện và ngăn chặn tấn công, mà còn cung cấp đủ thông tin để phân tích, phản ứng và tối ưu bảo mật một cách liên tục. Trong phần này, ta đánh giá ba khía cạnh chính: khả năng ghi log và giám sát (logging & monitoring), báo cáo và phân tích (reporting & analysis), cũng như năng lực quản lý tập trung trên quy mô lớn.

1. **Kết luận và đề xuất**
   1. **Tóm tắt so sánh**

Qua quá trình phân tích và đánh giá hai giải pháp WAF mã nguồn mở là ModSecurity và NAXSI, có thể rút ra một số nhận định chính:

* ModSecurity là một giải pháp WAF trưởng thành hơn, hỗ trợ đa nền tảng (Apache, Nginx, IIS) và có khả năng phân tích nội dung HTTP phức tạp thông qua rule dạng chữ ký (signature-based). Nó phù hợp cho các hệ thống cần bảo vệ toàn diện, nhiều tính năng nâng cao, tích hợp tốt với hệ sinh thái bảo mật như SIEM, IDS, và các công cụ phân tích log.
* NAXSI thiên về mô hình whitelist và đánh điểm (scoring), đơn giản hơn trong cấu hình, hiệu năng nhẹ, phù hợp với môi trường giới hạn tài nguyên như các hệ thống firewall nhúng hoặc cổng mạng trên OPNsense. Tuy nhiên, khả năng phát hiện các tấn công tinh vi còn hạn chế, đặc biệt với các kỹ thuật né tránh (evasion).

Từ đó cho thấy mỗi giải pháp có thế mạnh riêng, và không có công cụ nào “tốt nhất tuyệt đối” mà chỉ có “phù hợp nhất theo tình huống”.

* 1. **Đề xuất dựa trên tình huống**

Nếu triển khai WAF trong các hệ thống phức tạp, cần phát hiện nhiều loại tấn công khác nhau (SQLi, XSS, LFI, RFI...), cần tích hợp sâu với hệ thống log, SIEM hoặc các hệ thống giám sát thì ModSecurity là lựa chọn phù hợp hơn. Đặc biệt là khi có đội ngũ vận hành đủ kỹ thuật để tinh chỉnh rule và xử lý false positive.

Nếu triển khai trong môi trường sử dụng OPNsense làm firewall, đặc biệt với các dịch vụ web đơn giản hoặc yêu cầu bảo vệ mức cơ bản, muốn tiết kiệm tài nguyên, dễ quản lý, không cần tích hợp quá sâu thì NAXSI là lựa chọn hợp lý hơn. Nó cũng phù hợp với các hệ thống cần bảo mật web nội bộ mà không có quá nhiều nguy cơ tấn công tinh vi từ bên ngoài.

Trường hợp cần sử dụng cả OPNsense nhưng vẫn cần khả năng bảo vệ nâng cao, có thể kết hợp OPNsense dùng NAXSI làm WAF nhẹ ở lớp edge, và ModSecurity ở reverse proxy phía sau như Nginx hoặc Apache – giúp tận dụng ưu điểm của cả hai.

* 1. **Xu hướng phát triển trong tương lai**

ModSecurity vẫn đang phát triển với phiên bản v3, đặc biệt trên nền Nginx, đồng thời có cộng đồng mạnh duy trì bộ rule OWASP CRS. Các nền tảng thương mại như F5, Cloudflare vẫn tích hợp ModSecurity như một phần cốt lõi – cho thấy tính ổn định và khả năng mở rộng của nó trong tương lai.

NAXSI dù ít nổi bật nhưng vẫn được duy trì để phục vụ các trường hợp đặc thù như trên OPNsense. Hướng phát triển của NAXSI phụ thuộc vào cộng đồng firewall mã nguồn mở, và có thể sẽ tích hợp sâu hơn vào các bản phân phối như OPNsense thay vì phát triển độc lập.

Về tổng thể, công nghệ WAF đang dịch chuyển về hướng thông minh hơn, ứng dụng AI/ML trong việc phân tích lưu lượng và phát hiện tấn công. Tuy nhiên, trong môi trường on-premise hoặc hybrid cloud, các giải pháp mã nguồn mở như ModSecurity và NAXSI vẫn có vai trò nhất định, đặc biệt khi cần kiểm soát triệt để hoặc tuân thủ các yêu cầu bảo mật nội bộ.

**Phụ lục**

**A. Bảng tham khảo nhanh các rule phổ biến**

* Bảng rule phổ biến của ModSecurity
* Bảng rule phổ biến của NAXSI

**B. So sánh chi tiết cú pháp rule**

* Ví dụ và giải thích chi tiết cú pháp

**C. Tài liệu tham khảo**

* Danh sách tài liệu tham khảo
* Nguồn tài nguyên học tập