OWASP

The Open Web Application Security Project

Defending Layer 7: A look inside Web Application Firewalls

OWASP Aguascalientes - Chapter Meeting September – September 1st 2016

About us

{David Garcia}

- More than 8 years in Information Security
- Experience with Application Security, Vulnerability Management, Third Party Compliance.
- Expert in Pentesting
- Rubik’s cubes, soccer, Necaxa

We are not in social networks, we just talk by phone

{Alejandro Jalomo, MSc, CISSP, CRISC, CISA, ISO 27001 LA}

- 15 years in TI, 6 years in Information Security
- Experience with ISO 27001, HIPAA and PCI Compliance, Audit, Risk management.
- Expert in data protection solutions
- Drummer, black & gray tattoos, concerts

Agenda

- What is a Web Application Firewall (WAF)?
- When to use a WAF?
- WAF Architecture
- Key Market Players
- Top Ten Open Source WAFs
- Typical WAF Architecture
- Difference between IPS and WAF
- What is ModSecurity?
- What ModSecurity can do?
- Deployment Options
- Main Areas of Functionality
- What Rules Look Like
- Transaction Lifecycle
- Useful Rules
- Transaction Example
- Other projects
- What is WebKnight?
- Questions
- Demos

WebKnight
Open Source Web Application Firewall for IIS

ModSecurity
Open Source Web Application Firewall

What is a Web Application Firewall (WAF)?

- A web application firewall (WAF) is an appliance, server plugin, or filter that applies **a set of rules to an HTTP conversation.**

- Generally, these rules cover common attacks such as **cross-site scripting (XSS)** and **SQL injection**.

- By customizing the rules to your application, many attacks can be identified and blocked. The effort to perform this customization can be **significant** and needs to be maintained as the application is modified.

```
ead>
<meta http-equiv="Content-Type" content="
<title>CSS3</title>
<link href="style.css" rel="stylesheet" t
</head>
<body>
        container"> <!-- Main Container
                    <!-- Menu here -->
                      <h2>CSS3 Sample</h2>
```

What is a Web Application Firewall (WAF)?

- **Software or appliances used to filter unwanted TCP port 80/443 traffic from connecting to a web server**
- Web Application Firewalls:
    - Examine within the data payload, beyond simply the IP or TCP headers
    - Perform "Deep packet inspection"
    - Detect and respond to signatures for known application vulnerabilities
    - Do not require modifications to existing application code

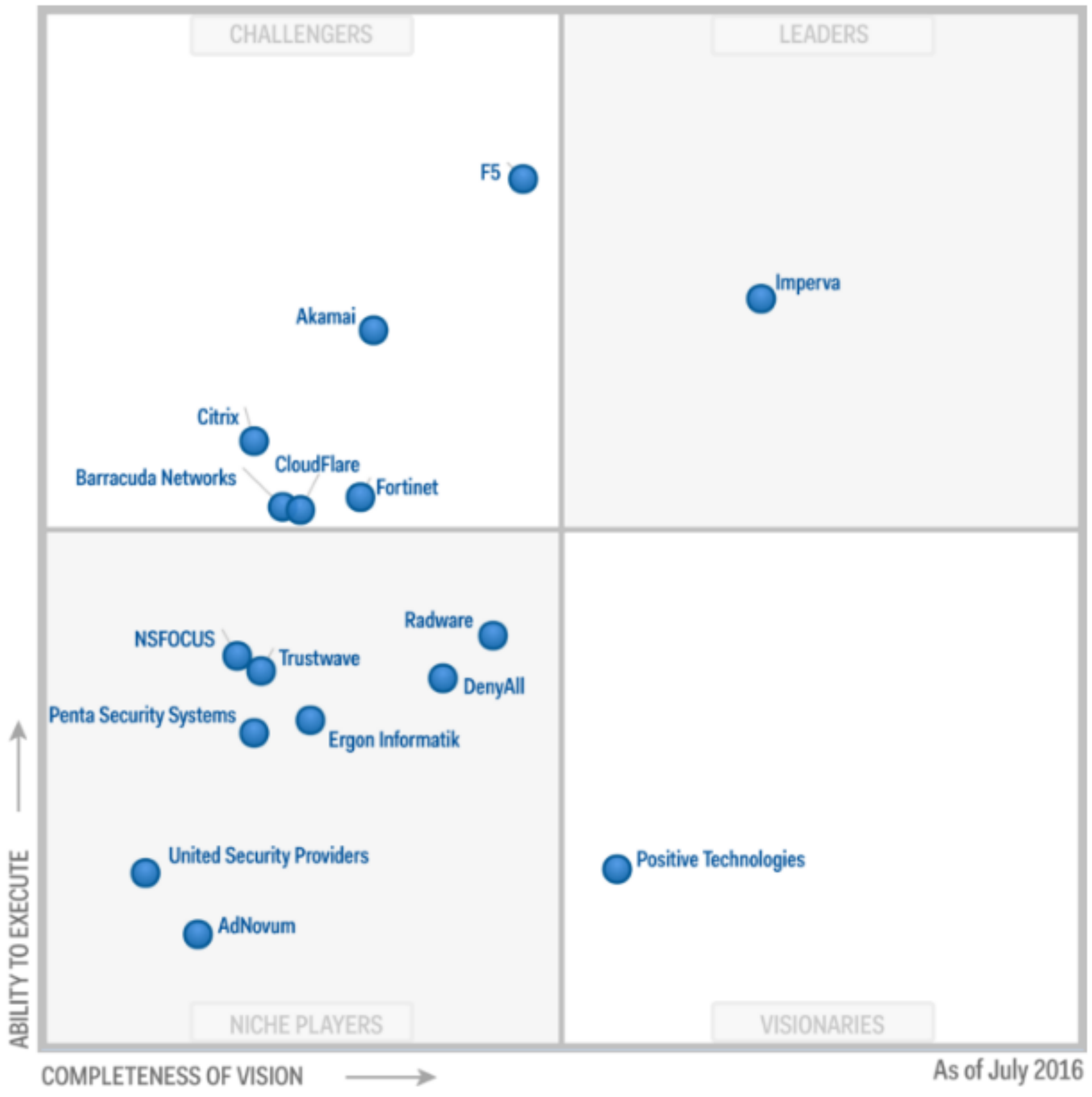
When to use a WAF?

I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION
memegenerator.net

Gartner 2016 Magic Quadrant for Web Application Firewalls

Gartner

Key Market Players

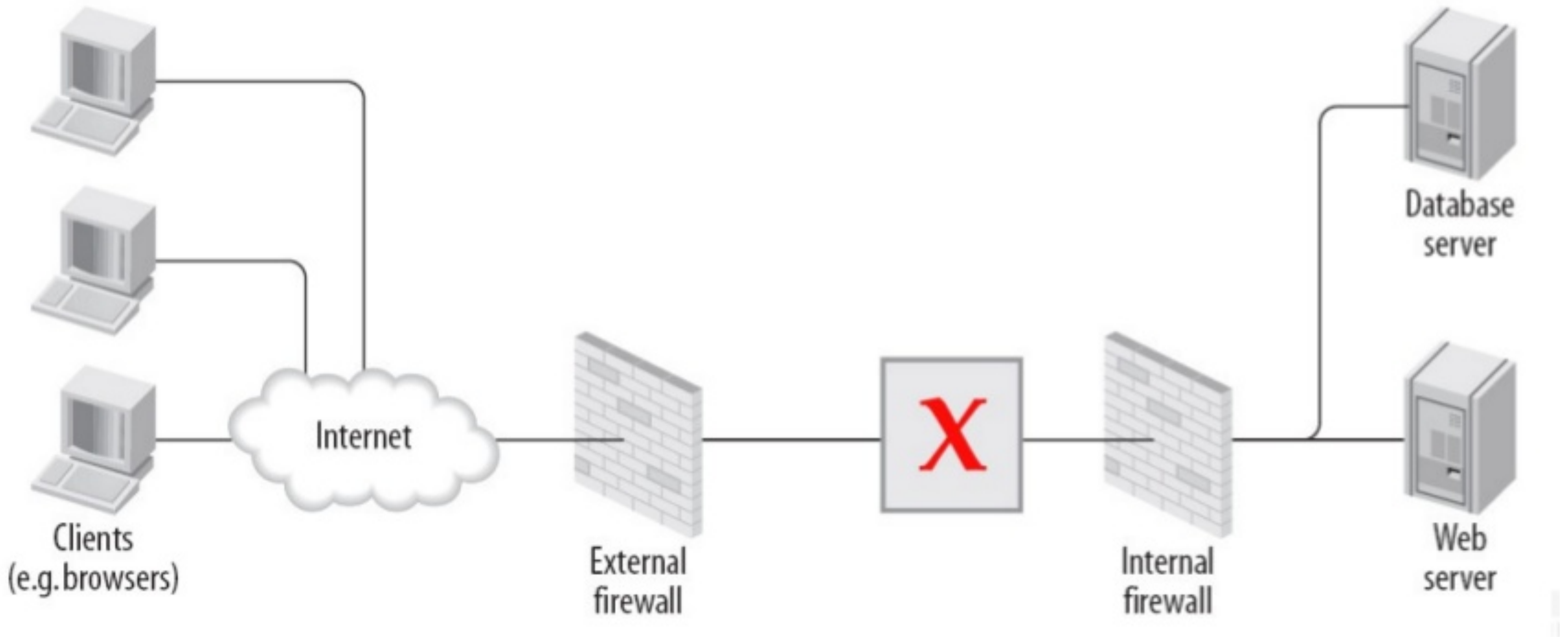


Magic Quadrant for Web Application Firewalls,Published: 19 July 2016

Top 10 Open Source WAFs

1. ModSecurity (Trustwave SpiderLabs)
2. AQTRONIX WebKnight
3. ESAPI WAF
4. WebCastellum
5. Binarysec
6. Guardian@JUMPERZ.NET
7. OpenWAF
8. Ironbee
9. Profense
10. Smoothwall

Typical WAF Architecture



Difference between IPS and WAF

W

ModSecurity
Open Source Web Application Firewall

ı are able to analyze it in real time,
ts

:ation—even if you can’t access the

toolkit for real-time web application monitoring, logging, and access control.

What ModSecurity can do?

ModSecurity
Open Source Web Application Firewall

- Real-time application security monitoring and access control

  At its core, ModSecurity gives you access to the HTTP traffic stream, in real-time, along with the ability to inspect it.

- Virtual patching

  Virtual patching is a concept of vulnerability mitigation in a separate layer, where you get to fix problems in applications without having to touch the applications themselves.

  ModSecurity excels at virtual patching because of its reliable blocking capabilities and the flexible rule language that can be adapted to any need.

- Full HTTP traffic logging

  ModSecurity gives you that ability to log anything you need, including raw transaction data, which is essential for forensics.

ModSecurity
Open Source Web Application Firewall

- Web application hardening

  ModSecurity is attack surface reduction, in which you selectively narrow down the HTTP features you are willing to accept (e.g., request methods, request headers, content types, etc.).

Deployment Options

ModSecurity
Open Source Web Application Firewall

- Embedded

  Because ModSecurity is an Apache module, you can add it to any compatible version of Apache.

  The embedded option is a great choice for those who already have their architecture laid out and don’t want to change it.

- Reverse proxy

  Reverse proxies are effectively HTTP routers, designed to stand between web servers and their clients.

  You can use it to protect any number of web servers on the same network.

Main Areas of Functionality

- Parsing

  The supported data formats are backed by security-conscious parsers that extract bits of data and store them for use in the rules.

- Buffering

  Both request and response bodies will be buffered. This means that ModSecurity usually sees complete requests before they are passed to the application for processing, and complete responses before they are sent to clients.

Main Areas of Functionality

- Logging

  This feature allows you to record complete HTTP traffic. Request headers, request body, response header, response body will be available

- Rule engine

  The rule engine builds on the work performed by all other components. By the time the rule engine starts operating, the various bits and pieces of data it requires will all be prepared and ready for inspection.

  At that point, the rules will take over to assess the transaction and take actions as necessary.

What Rules Look Like

configuration tells ModSecurity how to process the data it sees;
the rules decide what to do with the processed data.

```
SecRule ARGS "<script>" log,deny,status:404
```

```
SecRule VARIABLES OPERATOR ACTIONS
```

The three parts have the following meanings:

1. The VARIABLES part tells ModSecurity where to look. The ARGS variable, used in the example, means all request parameters.

2. The OPERATOR part tells ModSecurity how to look. In the example, we have a regular expression pattern, which will be matched against ARGS.

3. The ACTIONS part tells ModSecurity what to do on a match. The rule in the example gives three instructions: log problem, deny transaction and use the status 404 for the denial (status:404).

TRIPTANK
TRIPTANK
maint ance
W.A.N.G.

Transaction Lifecycle

ModSecurity

Open Source Web Application Firewall

is the **main request analysis** phase and takes place immediately after a complete request body has been received and processed.

Request Body (2)

Transaction Lifecycle

takes place after response headers become available, but before a response body is read.

Response Headers (3)

The rules that need to decide whether to inspect a response body should run in this phase.

Transaction Lifecycle

ModSecurity
Open Source Web Application Firewall

is the main response analysis phase. The response body will have been read, with all its data available for the rules to make their decisions.

Response Body (4)

Transaction Lifecycle

ModSecurity
Open Source Web Application Firewall

It’s the only phase from which you cannot block.

By the time this phase runs, the transaction will have finished, so there’s little you can do **but record** the fact that it happened.

Logging (5)

Useful Rules

AV Integration

```
SecRule FILES_TMPNAMES "@inspectFile /opt/modsecurity/bin/file-inspect.pl“ \
    phase:2,t:none,log,block
```

Drop for Brute Force

```
SecAction phase:1,initcol:ip=%{REMOTE_ADDR},nolog
SecRule ARGS:login "!^$" \
      nolog,phase:1,setvar:ip.auth_attempt=+1,deprecatevar:ip.auth_attempt=20/120
SecRule IP:AUTH_ATTEMPT "@gt 25" \
     "log,drop,phase:1,msg:'Possible Brute Force Attack'"
```

Transaction Example

response

```
HTTP/1.1 200 OK
Date: Sun, 17 Jan 2010 00:13:44 GMT
Server: Apache
Content-Length: 12 Connection: close
Content-Type: text/html

Hello World!
```

ModSecurity is first invoked by Apache after request headers become available, but before a request body (if any) is read.

First comes the initialization message, which contains the unique transaction ID generated by mod_unique_id.

Using this information, you should be able to pair the information in the debug log with the information in your access and audit logs.

At this point, ModSecurity will parse the information on the request line and in the request headers

Transaction Example

In this example, the query string part contains a single parameter (a), so you will see a message documenting its discovery. ModSecurity will then create a transaction context and invoke the REQUEST_HEADERS phase:

```
[4] Initialising transaction (txid SopXW38EAAE9YbLQ).
[5] Adding request argument (QUERY_STRING): name "a", value "test"
[4] Transaction context created (dcfg 8121800).
[4] Starting phase REQUEST_HEADERS.
```

Assuming that a rule didn’t block the transaction, ModSecurity will now return control to Apache, allowing other modules to process the request before control is given back to it.

Transaction Example

In the second phase, ModSecurity will first read and process the request body, if it is present.

In the following example, you can see three messages from the input filter, which tell you what was read.

The fourth message tells you that one parameter was extracted from the request body. The content type used in this request (application/x-www-form-urlencoded) is one of the types ModSecurity recognizes and parses automatically. Once the request body is processed, the REQUEST_BODY rules are processed.

```
[4] Second phase starting (dcfg 8121800).
[4] Input filter: Reading request body.
[9] Input filter: Bucket type HEAP contains 6 bytes.
[9] Input filter: Bucket type EOS contains 0 bytes.
[5] Adding request argument (BODY): name "b", value "test"
[4] Input filter: Completed receiving request body (length 6).
[4] Starting phase REQUEST_BODY.
```

Transaction Example

Shortly thereafter, the output filter will start receiving data, at which point the RESPONSE_HEADERS rules will be invoked:

```
[9] Output filter: Receiving output (f 81d2258, r 81d0588).
[4] Starting phase RESPONSE_HEADERS.
```

Transaction Example

ModSecurity

Open Source Web Application Firewall

Once all the rules have run, ModSecurity will continue to store the response body in its buffers, after which it will run the RESPONSE_BODY rules:

```
[9] Output filter: Bucket type MMAP contains 12 bytes.
[9] Output filter: Bucket type EOS contains 0 bytes.
[4] Output filter: Completed receiving response body (buffered full - 12 bytes).
[4] Starting phase RESPONSE_BODY.
```

Transaction Example

Finally, the logging phase will commence. The LOGGING rules will be run first to allow them to influence logging, after which the audit logging subsystem will be invoked to log the transaction if necessary. A message from the audit logging subsystem will be the last transaction message in the logs. In this example, ModSecurity tells us that it didn’t find anything of interest in the transaction and that it sees no reason to log it:

```
[4] Initialising logging.
[4] Starting phase LOGGING.
[4] Audit log: Ignoring a non-relevant request.
```

Transaction Example

Again, assuming that none of the rules blocked, the accumulated response body will be forwarded to the client:

```
[4] Output filter: Output forwarding complete.
```

Other projects

1. http://waf-fle.org/
2. https://splunkbase.splunk.com/app/880/
3. https://www.owasp.org/index.php/Category:OWASP_WeBekci_Project
4. http://www.root25.com/2013/02/mod-security-log-auditor-application-in-PHP-free-analyse-draw-chart-from-modsecurity-log.html
5. https://www.netsparker.com/blog/docs-and-faqs/generate-modsecurity-web-application-firewall-rules/

WebKnight

Open Source Web Application Firewall for IIS

AQTRONIX WebKnight is an application firewall for IIS and other web servers and is released under the GNU General Public License.

More particularly it is an ISAPI filter that secures your web server by blocking certain requests.

If an alert is triggered WebKnight will take over and protect the web server.

It does this by scanning all requests and processing them based on filter rules, set by the administrator.

These rules are not based on a database of attack signatures that require regular updates. Instead WebKnight uses security filters as buffer overflow, SQL injection, directory traversal, character encoding and other attacks.

This way WebKnight can protect your server against all known and unknown attacks.

Because WebKnight is an ISAPI filter it has the advantage of working closely with the web server, this way it can do more than other firewalls and intrusion detection systems, like scanning encrypted traffic.

DEMOS