ApacheCon Europe '08

ModSecurity

The Open Source Web Application Firewall

Ivan Ristić

Chief Evangelist

Breach Security

BREACH

Introduction

Ivan Ristic

- **Web application security** and **web application firewall** specialist.
- Author of **Apache Security**.
- Author of **ModSecurity**.
- **OWASP London Chapter** leader.
- Officer of the **Web Application Security Consortium**.
- Employed by **Breach Security**.

modsecurity

BREACH™

Part 1

What are Web Application Firewalls?

Problems with web applications

How did it all start?

- HTTP and browsers designed for document exchange.
- Web applications built using a number of loosely integrated technologies.
- No one thought about security at the time.

Where are we today?

- Most web applications suffer from one type of problem or another. It is very difficult to develop a reasonably secure web application.
- Not possible to achieve 100% security.

How can we improve the situation?

Education & good development practices.

- We have been working hard on this since 2000.
- Much better than it used to be, but still not good enough.
- Secure web programming too difficult and time consuming for your average programmer.

Design & code reviews.

- Slow and expensive.

Scanning & penetration testing.

- Not conclusive.
- Slow and expensive.

Why use web application firewalls?

It's a cost-effective technology that **works**.

It can be deployed **straight away**.

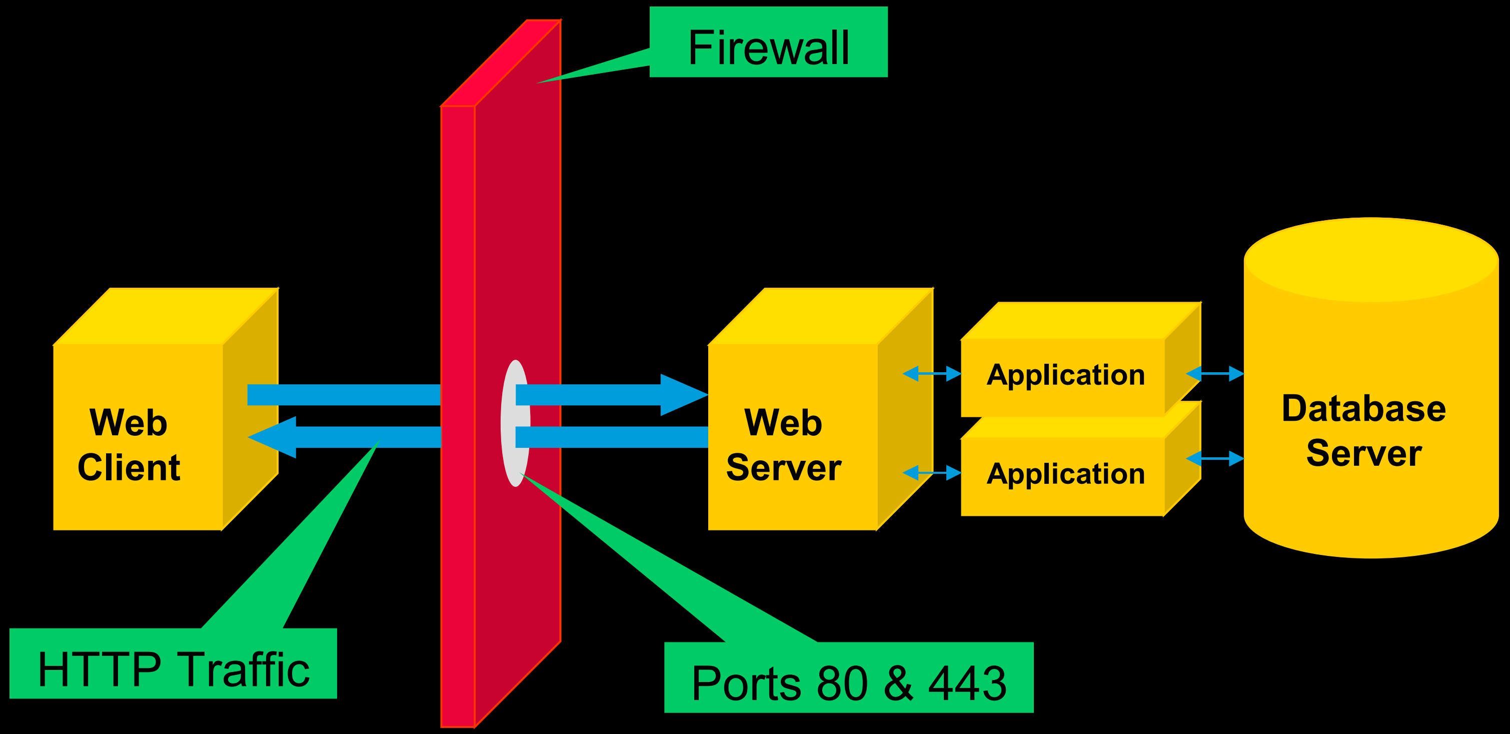
Gives **instant visibility** of the systems it protects.

Can provide instant **protection**.

In some of its forms (reverse proxies) it is actually an **essential building block of HTTP networks**.

Good example of **defence-in-depth**.

Network firewalls do not work



Neither do IDS/IPS solutions.

WAF identity problem: Naming

There is a long-standing WAF identity problem.

With the **name**, first of all:

- Adaptive Firewall
- Adaptive Proxy
- Adaptive Gateway
- Application Firewall
- Application-level Firewall
- Application-layer Firewall
- Application-level Security Gateway
- Application Level Gateway
- Application Security Device
- Application Security Gateway
- Stateful Multilayer Inspection Firewall
- Web Adaptive Firewall
- Web Application Controller
- **Web Application Firewall**
- Web Application Security Device
- Web Application Proxy
- Web Application Shield
- Web Shield
- Web Security Firewall
- Web Security Gateway
- Web Security Proxy
- Web Intrusion Detection System
- Web Intrusion Prevention System

WAF identity problem: Purpose

There are four main aspects to consider:

1. **Auditing/monitoring device**
   - **Attacks and client activity**
   - **Passive defect/vulnerability discovery**
2. **Access control device**
3. **Layer 7 router/switch (reverse proxy)**
4. **Web application hardening tool**

The name (WAF) is *overloaded*. How about:

- **Web Intrusion Detection System?**
- **HTTP Security Monitoring?**

WAFEC

Short for Web Application Firewall Evaluation Criteria.

Project of the Web Application Security Consortium (webappsec.org).

It's an open project.

Virtually all WAF vendors on board (not enough users though).

WAFEC v1.0 released in 2006.

- Version 2 coming in May 2008.

Part 2

ModSecurity

What is ModSecurity?

It is an **open source web application firewall**.

- **Most widely deployed web application firewall** according to Forrester Research.

That's not surprising because it is:

- Free.
- Full-featured.
- Stable and reliable.
- Well documented.
- Does what it says on the box.
- Been around for years.

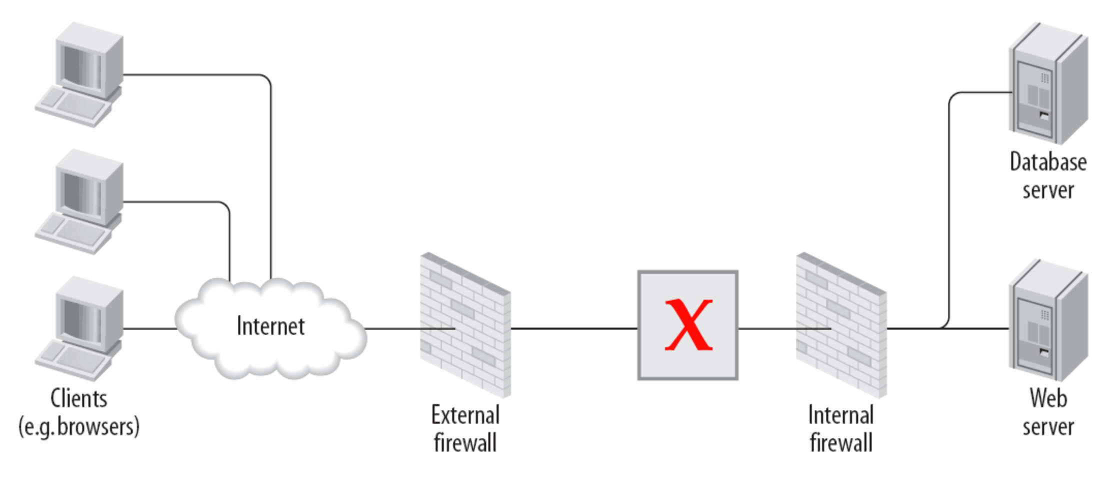
modsecurity

History of ModSecurity

- Project started in 2002:
  - **“Wouldn’t it be nice if I had something to monitor what’s going on in my applications?”**
- Commercial support through Thinking Stone since 2004.
- Acquired by Breach Security in 2006.
  - **Breach Security pledges to support the open source nature of the project; adds resources.**
  - **Still going strong.**

Deployment architectures

- **Embed** into your existing web servers.
- Deploy as a **network gateway** combining **Apache** working as reverse proxy with ModSecurity.



ModSecurity philosophy

- **Empower the users to do what they need.**
- Don't do anything implicitly.
  - Everything is in the configuration.
- Be passive (if you can).
  - Errors raise flags, which need to be handled in the rules.
- Offer no surprises.
  - Document everything and tell it like it is.

Request lifecycle (I)

- Run phase 1 as early as possible (REQUEST HEADERS).
  - Reuse information available from Apache.
  - Perform additional parsing as necessary.
- Buffer and parse the request body (optional).
  - Look for protocol-level evasion.
- Run phase 2 (REQUEST BODY).
- Allow the request to be processed.

Request lifecycle (II)

- Run phase 3 before headers are sent (RESPONSE HEADERS).
- Buffer response body (optional).
  - Depending on the response MIME type and custom instructions.
- Run phase 4 (RESPONSE BODY).
- Run phase 5 (LOGGING).
- Log transaction (optional).

Transaction logging

- ModSecurity will log complete transaction data on demand. Using the rules you can:
  - **Choose whether to log a transaction.**
  - **Choose which parts to log.**
- Transactions are recorded in two formats:
  - ***Serial* – single file; convenient but limited.**
  - ***Concurrent* – file per transaction; scalable but not suitable for manual handling.**
- ***Mlogc*** picks up transactions as they are recorded and sends them to the central logging server.
  - **It's fast, secure (SSL), reliable, and uses buffering.**

ModSecurity Rule Language

- It's a simple event-based programming language, which lives within the Apache configuration syntax.
  - Look at any part of the transaction.
  - Transform data to counter evasion.
  - Perform various actions.
  - Combine rules to form complex logic.
- Common tasks are easy, complex tasks are possible.

Rules

Generic syntax:

```
SecRule TARGETS OPERATOR [ACTIONS]
```

For example:

```
SecRule ARGS|REQUEST_HEADERS "<script" \
    "id:1001,msg:'XSS Attack', \
    severity:ERROR,deny,status:404"
```

With rule chaining:

```
SecRule ARGS:username "@streq admin" \
    chain,deny
SecRule REMOTE_ADDR "!@streq 192.168.1.1"
```

Target Variables

- Using variables you tell ModSecurity *where to look*.
  - There are 78 variables in the latest version.
- For example:
  - » `ARGS`
  - » `ARGS_COMBINED_SIZE`
  - » `ARGS_NAMES`
  - » `ARGS_GET`
  - » `ARGS_POST`
  - » …

Operators

- Operators tell ModSecurity *how to process* request data.
  - There are 22 operators in the latest version.
- For example:
  - **Strings** (rx, pm, beginsWith, contains, endsWith, streq, within).
  - **Numerical** (eq, ge, gt, le, lt).
  - **XML** (validateDTD, validateSchema).
  - **Other** (rbl, geoLookup, inspectFile, verifyCC).

Actions

- Actions tell ModSecurity *what to do* when a match occurs.
  - There are 42 actions in the latest version.
- Possible use of actions:
  - **Block transaction** (block, drop, deny, proxy).
  - **Influence logging** (log, auditlog, sanitiseArg).
  - **Set/change/delete variables** (setvar, setenv).
  - **Access persistent storage** (initcol).
  - …

Part 3

Real-life Examples

Ignore static content

In most cases you don't want to waste CPU cycles analysing requests for static resources.

```
<Location /g/>
    SecRuleEngine Off
</Location>
```

You can also do this (works best in embedded mode):

```
SecRule REQUEST_METHOD "^(GET|HEAD)$" \
    chain,allow,nolog
SecRule REQUEST_BASENAME "\.(jpg|gif|png)$" chain
SecRule &ARGS "@eq 0"
```

Virtual patching example

Virtual patching example using the *positive security* approach:

```
<Location /apps/script.php>
    SecRule &ARGS "!@eq 1"
    SecRule ARGS_NAMES "!^statid$"
    SecRule ARGS:statID "!^\d{1,3}$"
</Location>
```

White-list IP address or IP range

A frequent request is to create an exception to not process requests coming from an IP address:

```
SecRule REMOTE_ADDR "@streq 192.168.254.1" \
    allow,phase:1,nolog

SecRule REMOTE_ADDR "@beginsWith 192.168.254." \
    allow,phase:1,nolog

SecRule REMOTE_ADDR "@rx ^192\.168\.254\.(1|2|5)$" \
    allow,phase:1,nolog
```

In a future version we will probably introduce a new operator, `@ipMatch`, to make working with network segments easier.

Track activity per IP address

Initialise IP address collection:

```
SecAction \
phase:1,initcol:ip=%{REMOTE_ADDR},nolog,pass
```

Deny IP addresses whose scores are too high:

```
SecRule IP:score "@gt 20" phase:1,log,deny
```

Increment score on rule match:

```
SecRule ARGS pattern phase:2,pass,setvar:ip.score=+1
```

Associate session with request

ModSecurity has support for sessions, but you need to help it by extracting the session ID from request:

```
SecRule REQUEST_COOKIES:PHPSESSID !^$ \
    "chain,phase:2,nolog,pass, \
    setsid:%{REQUEST_COOKIES.PHPSESSID}"
```

Collection SESSION will be available from this moment on to store per-session data, persistent across requests.

Transaction will be tagged with the session ID in the transaction log.

Sanitise data before logging

If you know the sensitive parameter names in advance:

```
SecAction "phase:5,nolog,pass, \
    sanitiseArg:password, \
    sanitiseArg:password_again, \
    sanitiseArg:oldPassword"
```

For any parameter name that sounds like a password:

```
SecAction ARGS_NAMES password \
    phase:5,nolog,pass,sanitiseMatched
```

Or based on content:

```
SecRule ARGS "@verifyCC CCREGEX" \
    phase:5,nolog,pass,sanitiseMatched
```

Dealing with evasion

Writing rules to deal with all possible combinations of evasion methods is not only time consuming, it's impossible. A few evasion examples:

```
drop table
dRoP\ntaBle
DROP/*sdfjsdlkfj*/TABLE
```

ModSecurity uses a concept of transformation functions to deal with this problem:

```
SecRule ARGS "drop table" \
    t:lowercase,t:replaceComments, \
    t:compressWhitespace"
```

Decisions based on client location

You can take the geographic location of the client into account when making decisions.

First you configure the GeoIP database (download free from maxmind.com):

```
SecGeoLookupDb /path/to/geo.db
```

Then look the IP address up:

```
SecRule REMOTE_ADDR @geoLookup \
    "phase:1,chain,drop,msg:'Non-UK IP address'"
SecRule GEO:COUNTRY_CODE "!@streq UK"
```

Capture and transform data

Sometimes you need to transform input data before you can look at it.

The HTTP Basic authentication, for example:

```
Authorization: Basic bm9ib2R5OnRlc3Q=
```

The following rules would do the trick:

```
SecRule REQUEST_HEADERS:Authorization \
    "^Basic ([a-zA-Z0-9]+=*)$" \
    phase:1,capture,chain
SecRule TX:1 ^(\w+): t:base64Decode,capture,chain
SecRule TX:1 ^(admin|root|backup)$ logdata:%{TX.1}
```

Write rule in Lua (experimental)

As of 2.5 you can write rules in Lua:

```
SecRuleScript /path/to/file.lua
```

And the script:

```
function main()
    m.log(1, "Hello world!");

    local var1 = m.getvar("REMOTE_ADDR");
    local var2 = m.getvar("REQUEST_URI",
        "normalisePath");
    local var3 = m.getvar("ARGS.p", { "lowercase",
        "compressWhitespace" } );

    return "Variable ARGS:p looks suspicious!";
end
```

Part 4

Roadmap (2008)

Portability

- Limited by being too close to Apache:
  - **Need to reload configuration without affecting the web server.**
  - **Need freedom to expand rule syntax.**
- Work embedded in any web server.
  - **Port to IIS and ISA.**
  - **Help the community port to other web servers.**
- Other deployment modes:
  - **Passive/sniffer.**
  - **Command line / batch processing.**

Learning

- Better support for positive security.
  - **We have good support for virtual patching but writing complex positive security rules is difficult.**
- Create positive security models automatically using traffic profiling.
- Make it easier to interact with the contextual information.
  - **Customise policy based on the target system.**

Modularity

- Formal component boundaries to allow for a mix-and-match deployment of modules. For example:
  - **Deploy with a different persistence backend.**
- Formats for data exchange.
- Handle complex requirements better:
  - **Write rules in C.**
  - **Write rules in Lua (already in 2.5).**

Part 5

Related Projects

ModSecurity Core Rules

Coherent set of rules designed to detect generic web application security attacks.

- Bundled with ModSecurity, but with a separate release cycle.
- Lead by Ofer Shezaf.

Design goals:

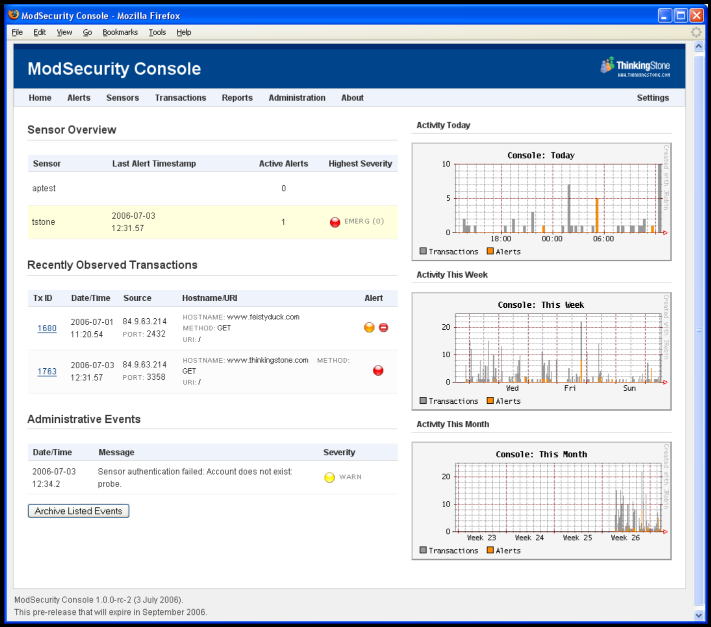
- Performance.
- Quality.
- Stability.
- Plug and Play.

Automated updates supported since ModSecurity 2.5.

ModSecurity Community Console

Self-contained application designed for alert aggregation, monitoring and reporting.

- Portable (Java).
- Embedded database.
- Free for up to 3 sensors.
- Not open source.



Sensor Overview

Sensor	Last Alert Timestamp	Active Alerts	Highest Severity
aptest		0	
tstone	2006-07-03 12:31.57	1	● EMERG (0)

Recently Observed Transactions

Tx ID	Date/Time	Source	Hostname/URI	Alert
1680	2006-07-01 11:20.54	84.9.63.214 PORT: 2432	HOSTNAME: www.feistyduck.com METHOD: GET URI: /	● ⊖
1763	2006-07-03 12:31.57	84.9.63.214 PORT: 3358	HOSTNAME: www.thinkingstone.com METHOD: GET URI: /	●

Administrative Events

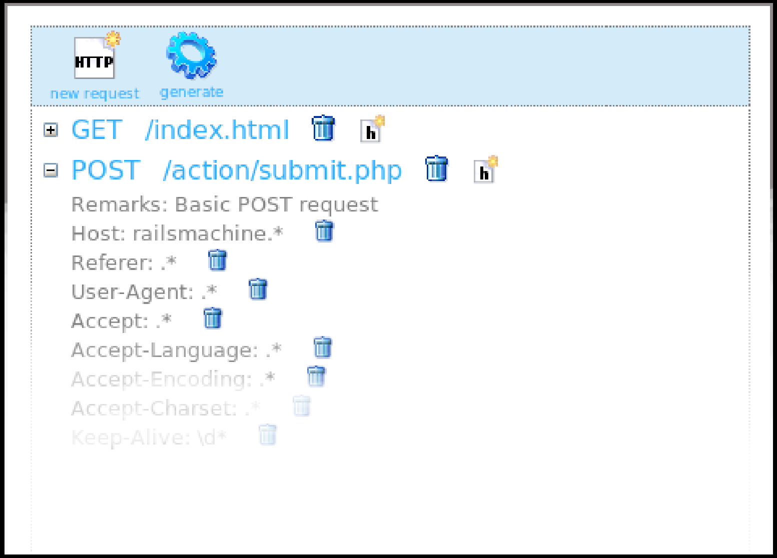
Date/Time	Message	Severity
2006-07-03 12:34.2	Sensor authentication failed: Account does not exist: probe.	● WARN

ModSecurity Console 1.0.0-rc-2 (3 July 2006).
This pre-release that will expire in September 2006.

REMO

A project to build a graphical rule editor for ModSecurity with a positive / whitelist approach.

- REMO stands for Rule Editor for ModSecurity.
- Community project run by Christian Folini.



⊞ GET /index.html

⊟ POST /action/submit.php

Remarks: Basic POST request
Host: railsmachine.*
Referer: .*
User-Agent: .*
Accept: .*
Accept-Language: .*
Accept-Encoding: .*
Accept-Charset: .*
Keep-Alive: \d*

Distributed Open Proxy Honeypots

A network of open proxy sensors, each deployed with ModSecurity configured to log to the central server.

Goals:

- Observe what the bad guys are doing.
- Fine tune detection rules.
- WASC project (**webappsec.org**), run by Ryan Barnett.

Questions?

Thank you!

Ivan Ristic

ivanr@webkreator.com