Developed by Arshan Dabirsiaghi (w/ Jeff Williams)
A sub-project under the ESAPI umbrella
The *Star Trek: TNG* of WAFs

- Free
- Usable
- Robust
- Pragmatic
- Performant
- Open Source
## “Yeah, well I hate WAFs”

- **Perfect! Me too.**

<table>
<thead>
<tr>
<th>WAF criticism</th>
<th>Is criticism stupid?</th>
<th>Does ESAPI WAF have problem?</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAFs add attack surface</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WAFs can create culture problems</td>
<td>Maybe</td>
<td>(not sure, probably)</td>
</tr>
<tr>
<td>WAFs can’t fix business logic vulnerabilities</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WAFs are way too expensive</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WAFs complicate networks</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
WAFs were for Federalists (part 1)

App A
Development Team A

App B
Development Team B

App C
Development Team C

WAF
Security Team
WAFs were for Federalists (part 2)
Why fix in ESAPI WAF vs. fix in code?

Time of Vulnerability Discovery

Time when Vulnerability Patched
Why fix in ESAPI WAF vs. fix in code?
Advantages of Application-Layer WAFs

**Performance** – only your rules are checked, plus state is already managed by the app server

**Capability** – being closer to the app lets us do more and I can’t wait to tell you about it

**Process** – rules are closer to application owner, shortening discovery-to-patch time, also fix-to-patch-removal time
Principle: Make common tasks easy, uncommon tasks possible

```xml
<virtual-patches>
  <virtual-patch
    id="bugtracker-id-1234"
    path="/vulnerable.do"
    variable="request.parameters.bar"
    pattern="[0-9a-zA-Z]*"
    message="zmg attax" />
</virtual-patches>

<bean-shell-rules>
  <bean-shell-script
    id="user-lockout-rule"
    file="/enforce_user_lockout.bsh"
    stage="before-request-body"/>
</bean-shell-rules>

import org.acme.user.*;
User user = session.getAttribute("u");
If ( user.isLocked() )
  action = new RedirectAction();
```
Fixing Injection Flaws

**XSS**
- Fix with input validation virtual patch (black/white list)
- Fix with sanitization (BeanShell script)
- Fix with output encoding (egress rule)

**SQL injection**
- Fix with input validation virtual patch (black/white list)
- Fix with sanitization (BeanShell script)

**Command injection**
- Fix with input validation virtual patch (black/white list)
- Fix with sanitization (BeanShell script)
**Business Logic Flaws**

<table>
<thead>
<tr>
<th>Path</th>
<th>Vulnerability</th>
<th>Description</th>
</tr>
</thead>
</table>
| /ws/lmptWebService.rest       | Missing Authentication                 | • YES WE CAN  
• Presence/value of session variable  
• Presence of appliance-supplied header  
• BeanShell script |
| /admin/shutdown                | Missing Functional Access Control      | • YES WE CAN  
• Check roles in session  
• Check roles provided by appliance-supplied header  
• BeanShell script |
| /viewAccount?id=1826           | Missing Data Layer Access Control      | • YES WE CAN  
• BeanShell script |
Adding “Outbound” Security

Headers
- Add anti-clickjacking header
- Set uniform content-type

Cookies
- Add HttpOnly flag
- Add secure flag

Body
- Detect outbound information
- Replace outbound information

HTTP/1.1 200 OK
Date: Mon, 11 Jun 2007 16:49:25 GMT
Server: Apache/1.3.33 (Unix)
Last-Modified: Mon, 29 Aug 2005 12:01:40 GMT
ETag: "5800535-3e72-4312924"
Accept-Ranges: bytes
Content-Length: 15986
Content-Type: image/jpeg
Yes, we know all about early failing

- Do I care about URL?
- What about content-type?

Ok, go on...

- Is IP range private?
- Etc.

Ok, go on...

- Perform rule

Worst case scenario
Meet JForum 2.1.8

Awesome, free, fully featured forum software.

4 hours of code review/pen testing:
10 findings
XSS/Unchecked redirect
Add HttpOnly
Add anti-clickjacking header
Privilege escalation
XSS/Unchecked redirect

Add HttpOnly

Add anti-clickjacking header

Privilege escalation
XSS/Unchecked redirect

Add anti-clickjacking header

Privilege escalation
XSS/Unchecked redirect

Add HttpOnly

Add anti-clickjacking header

Privilege escalation
XSS/Unchecked redirect

Add anti-clickjacking header

Privilege escalation
Package org.owasp.esapi.waf

This package contains the ESAPI Web Application Firewall (WAF).

See:
Description

Class Summary

ESAPIWebApplicationFirewallFilter This is the main class for the ESAPI Web Application Firewall (WAF).

Exception Summary

ConfigurationException The Exception to be thrown when there is an error parsing a policy file.

Package org.owasp.esapi.waf Description

This package contains the ESAPI Web Application Firewall (WAF). It is an optional feature of ESAPI that can be used with or without ESAPI's other security controls in place. Its purpose is to provide fast virtual patching capabilities against known vulnerabilities or the enforcement of existing security policies where possible.
2. The policy file

The ESAPI Web Application Firewall (WAF) is driven by an XML policy file that tells it what rules to enforce in the application. These rules can do a number of things, from simple virtual patching to complex authorization enforcement with BeanShell scripts.

This document describes the structure of the policy file, the individual rules and how they work. There are also a number of examples in order to guide you during implementation. The following picture shows you a visual representation of the policy file XSD, a formal specification for the layout of a policy file:

![Diagram of policy file XSD](http://owasp-esapi-java.googlecode.com/svn/trunk/documentation/OWASP%20ESAPI%20WAF%20Configuration%20Guide.pdf)
OWASP ESAPI
WAF
AVAILABLE NOW
$0

Arshan Dabirsiaghi
Director of Research, Aspect Security
@nahsra, i8<messiah>.com
http://www.aspectsecurity.com/