QUALYS®

Protocol-Level Evasion of Web Application Firewalls

Ivan Ristic
Director of Engineering

True Evasion Story

- Once, a long time ago, I evaded a web application firewall by adding a single character to a valid request. *Can you spot it below?*

```
GET /myapp/admin.php?userid=1001 HTTP/1.1
Host: www.example.com.
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.0)
 Gecko/20100101 Firefox/13.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
```

True Evasion Story

- Once, a long time ago, I evaded a web application firewall by adding a single character to a valid request. *Can you spot it below?*

```
GET /myapp/admin.php?user                    1.1
Host: www.example.com.
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.0)
 Gecko/20100101 Firefox/13.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
```

Why Do I Care?

- Spent years developing WAFs and related software:
  - Built **ModSecurity** (2002-2009)
  - Built **libhtp** (2009-2010)
  - Now working on **IronBee** (not coding, though)
- WAF concepts are powerful, but the field needs *more research* and the market needs *more transparency*

modsecurity

libhtp

IRONBEE™
Open Source WAF

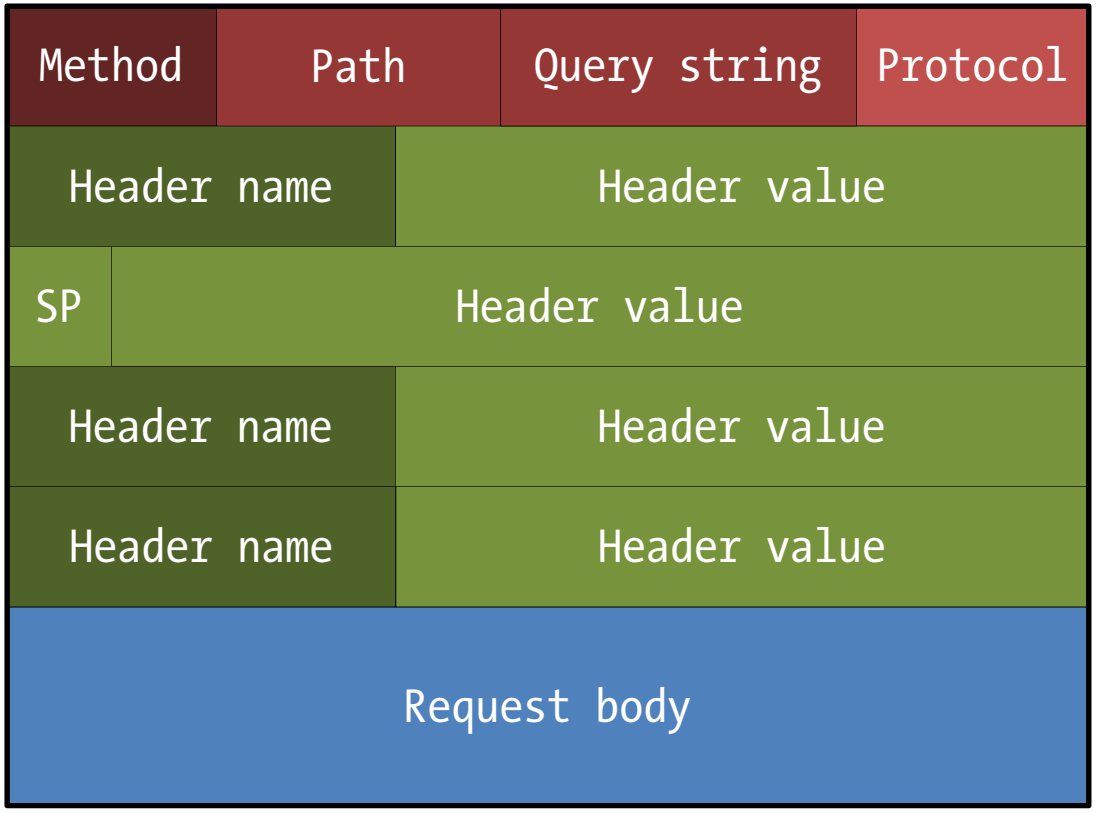
1 INTRODUCTION TO PROTOCOL-LEVEL EVASION

Impedance Mismatch

- *Impedance mismatch*, in the context of security monitoring, refers to the problem of different interpretations of the same data stream
  - The security tool sees one thing
  - The backend server sees another
- Possible causes:
  - Ambiguous standards
  - Partial and "Works for me" backend implementations
  - "Helpful" developer mentality
  - Insufficient attention by security product developers

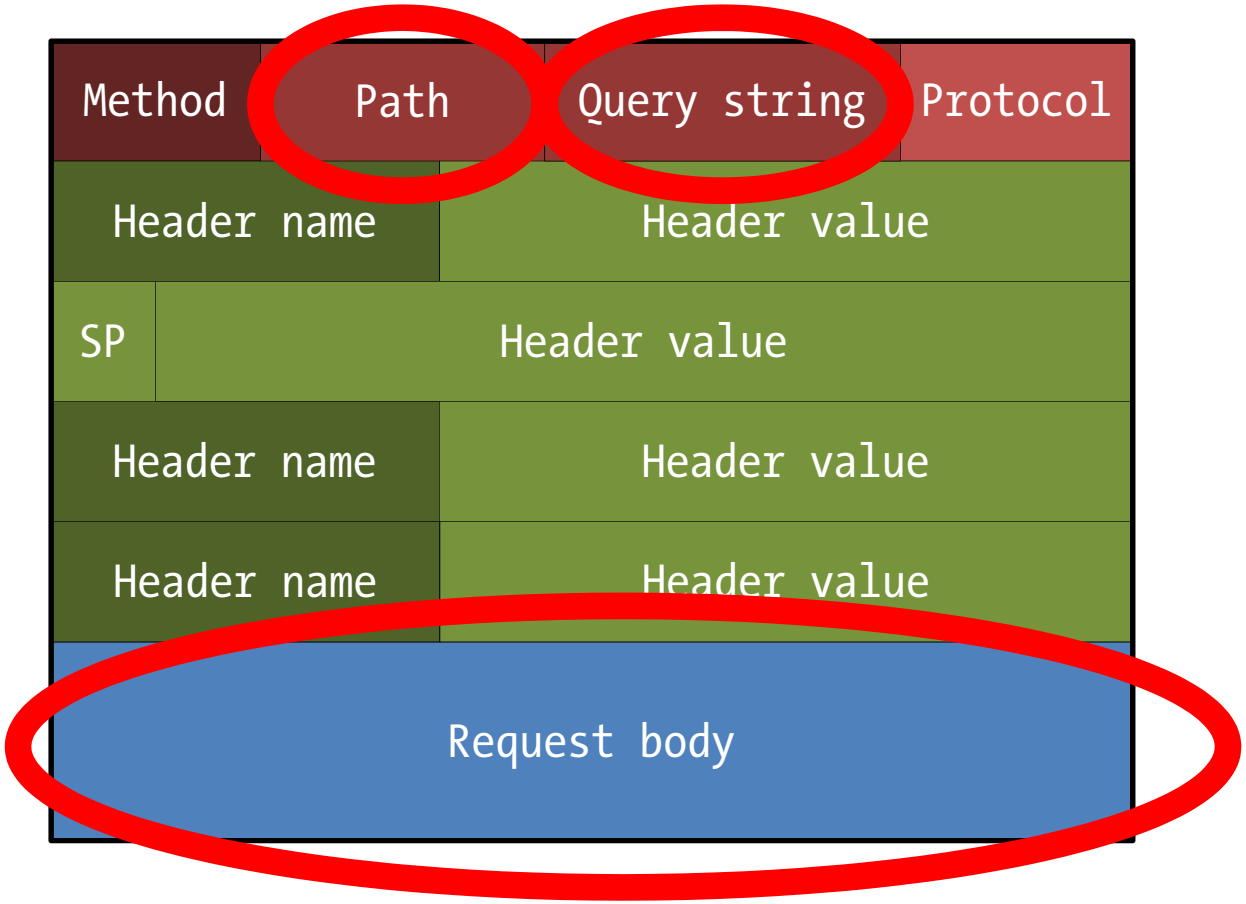
Protocol-Level Evasion Overview

- HTTP
  - Message parsing
  - Request line
  - Request headers
  - Cookies
- Hostname
- Path
- Parameters
- Request body
  - Urlencoded
  - Multipart



Protocol-Level Evasion Overview

- HTTP
  - Message parsing
  - Request line
  - Request headers
  - Cookies
- Hostname
- Path
- Parameters
- Request body
  - Urlencoded
  - Multipart



Virtual Patching

- Virtual patching is probably the most widely used WAF feature
  1. You know you have a problem
  2. You can't resolve it, or can't resolve it in a timely manner
  3. You deploy a WAF as a short-term mitigation measure
- Challenge:
  - To support the narrow focus of virtual patches, WAFs have to make a lot of processing decisions
  - The more decision points there are, the easier it is to successfully evade detection

2 PATH EVASION

Attacking Patch Activation

- An application entry point might look like this:

```
/myapp/admin.php?userid=1001
```

- And the virtual patch, using Apache and ModSecurity, like this:

```
<Location /myapp/admin.php>
  # Allow only numbers in userid
  SecRule ARGS:userid "!^\d+$"
</Location>
```

PATH_INFO and Path Parameters

- Surprisingly, some WAFs* still don't know about PATH_INFO:

```
/myapp/admin.php/xyz?userid=X
```

- If PATH_INFO is not supported by the backend server, you might want to try path parameters (e.g., works on Tomcat):

```
/myapp/admin.php;random=value?userid=X
```

(*) Neither approach works against Apache, because it uses Location parameter as prefix.

Self-Contained ModSecurity Rules

- Rules written like this are very easy to find:

```
SecRule REQUEST_FILENAME "@streq /myapp/admin.php“ \
    "chain,phase:2,deny"
SecRule ARGS:userid "!^\d+$"
```

- Problems:
  - The use of `@streq` misses `PATH_INFO` and path parameters attacks
  - Apache may not handle all obfuscation attacks, for example:

```
/myapp//admin.php
/myapp/./admin.php
/myapp/xyz/../admin.php
```

Self-Contained ModSecurity Rules

- Here's a better version of the same patch:

```
SecRule REQUEST_FILENAME \
"@beginsWith /myapp/admin.php" \
   "chain,phase:2,t:normalizePath,deny"
SecRule ARGS:userid "!^\d+$“
```

- Improvements:
  - Use `@beginsWith` (`@contains` is good, too)
  - Use transformation function `normalizePath` to counter path evasion attacks

Backend Feature Variations

- In a proxy deployment, you have to watch for impedance mismatch with various backend features:

```
/myapp\admin.php
/myapp/AdMiN.php
```

- Using Apache and ModSecurity:

```
<Location ~ (?i)^[\x5c/]+myapp[\x5c/]+admin\.php>
   SecRule ARGS:userid "!^\d+$"
</Location>
```

Backend Feature Variations

- In a proxy deployment, you have to watch for impedance mismatch with various backend features:

```
/myapp\admin.php
/myapp/AdMiN.php
```

- ModSecurity only:

```
SecRule REQUEST_FILENAME \
"@beginsWith /myapp/admin.php" \
   "chain,phase:2,t:lowercase,t:normalizePathWin,deny"
SecRule ARGS:userid "!^[0-9]+$"
```

Path Parameters Again

- Path parameters are actually *path segment parameters*, and can be used with any segment:

```
/myapp;param=value/admin.php?userid=X
```

- New patch version:

```
<Location ~ (?i)^[\x5c/]+myapp(;[^\x5c/]*)?
[\x5c/]+admin\.php(;[^\x5c/]*)?>
   SecRule ARGS:userid "!^\d+$"
</Location>
```

- ModSecurity needs a new transformation function; could use the same pattern as above or reject all path segment parameters

Short Filenames on Windows

- Windows uses short filenames to support legacy applications. For example:

  `admin.aspx`

  becomes

  `ADMIN~1.ASP`

- Ideal for virtual patch evasion under right circumstances:
  - Does not work with IIS
  - But does work with Apache running on Windows

Path Evasion against IIS 5.1

- IIS 5.1 (and, presumably, earlier) are *very* flexible when it comes to path processing:
  1. Overlong 2- or 3-byte UTF-8 representing either / or \
  2. In fact, any overlong UTF-8 character facilitates evasion
  3. Best-fit mapping of UTF-8 characters; for example U+0107 becomes c
  4. Best-fit mapping of %u-encoded characters
  5. Full-width mapping with UTF-8 encoded characters; for example U+FF0F becomes /
  6. Full-width mapping of %u encoding
  7. Terminate path using an encoded NUL byte (%00)
- IIS 5.1 and IIS 6 accept %u-encoded slashes

Path Handling of Major Platforms

Test	IIS 5.1	IIS 6.0	IIS 7.0	IIS 7.5	Apache 2.x	Tomcat 6.x
Path 00: Baseline test	Yes	Yes	Yes	Yes	Yes	Yes
Path 01: Supports %HH encoding	Yes	Yes	Yes	Yes	Yes	Yes
Path 02: Supports %uHHHH encoding	Yes	Yes	Status 400	Status 400	Status 400	Status 400
Path 03: Supports UTF-8 in filenames (encoded)	No	Yes	Yes	Yes	Yes (pass-through)	Configurable
Path 04: Supports UTF-8 in filenames (bare)	No	No	No	No	Yes (pass-through)	Configurable
Path 05: Performs best-fit mapping for %u	Yes	No (404; logs best	Status 400	Status 400	Status 400	Status 400
Path 06: Performs best-fit mapping for bare UTF-8	Yes	No	No	No	No	No
Path 07: Performs best-fit mapping for encoded UTF-8	Yes	No	No	No	No	No
Path 08: Invalid %HH encoding handling	Preserves %	Status 400	Status 400	Status 400	Status 400	Status 400
Path 09: Invalid %uHH encoding handling	Preserves %	Status 400	Status 400	Status 400	Status 400	Status 400
Path 10: Valid vs invalid %HH preference (e.g., d.txt vs %64.txt)	Valid	Valid	Valid	Valid	Valid	Valid
Path 11: Valid vs invalid %HHHH preference	Valid	Valid	Status 400	Status 400	Status 400	Status 400
Path 12: NUL byte (encoded)	Terminates path	Status 400	Status 400	Status 400	Status 404	Status 400
Path 13: NUL byte (bare)	Status 400	Status 400	Status 400	Status 400	Terminates path	Status 400
Path 14: Backslash as path segment separator	Yes	Yes	Yes	Yes	No	Status 400
Path 15: Forward slash as path segment separator (%u-encoded)	Yes	Yes	Status 400	Status 400	Status 400	Status 400
Path 16: Forward slash as path segment separator (URL-encoded)	Yes	Yes	Yes	Yes	Status 404 [No if e	Status 400
Path 17: Backslash as path segment separator (URL-encoded)	Yes	Yes	Yes	Yes	No	Status 400
Path 18: Backslash as path segment separator (%u-encoded)	Yes	Yes	Status 400	Status 400	Status 400	Status 400
Path 19: Control characters - encoded	No effect	Status 400	Status 400	Status 400	No effect	No effect
Path 20: Control characters - bare	No effect	Status 400	Status 400	Status 400	No effect	No effect
Path 21: Overlong UTF-8 sequences (non-separators) 2-byte sequence - encoded	Yes	No	No	No	No	No
Path 22: Overlong UTF-8 sequences (non-separators) 3-byte sequence - encoded	Yes	Status 400	Status 400	Status 400	No	No
Path 23: Overlong UTF-8 sequences (non-separators) 4-byte sequence - encoded	No	Status 400	Status 400	Status 400	No	No
Path 24: Overlong UTF-8 sequences (non-separators) 2-byte sequence - bare	Yes	No	No	No	No	No
Path 25: Overlong UTF-8 sequences (non-separators) 3-byte sequence - bare	Yes	Status 400	Status 400	Status 400	No	No
Path 26: Overlong UTF-8 sequences (non-separators) 4-byte sequence - bare	No	Status 400	Status 400	Status 400	No	No
Path 27: Overlong UTF-8 sequences (separators) 2-byte sequence - encoded	Yes	No	No	No	No	No
Path 28: Overlong UTF-8 sequences (separators) 3-byte sequence - encoded	Yes	No	No	No	No	No
Path 29: Overlong UTF-8 sequences (separators) 4-byte sequence - encoded	No	No	No	No	No	No
Path 30: Overlong UTF-8 sequences (separators) 2-byte sequence - bare	Yes	No	No	No	No	No
Path 31: Overlong UTF-8 sequences (separators) 3-byte sequence - bare	Yes	No	No	No	No	No
Path 32: Overlong UTF-8 sequences (separators) 4-byte sequence - bare	No	No	No	No	No	No
Path 33: Fullwidth form mapping from %u encoding	Yes	No (404; logs best	Status 400	Status 400	Status 400	Status 400
Path 34: Invalid UTF-8 encoding (encoded)	No effect	Status 400	Status 400	Status 400	No effect	No effect
Path 35: Fullwidth form mapping from UTF-8 encoded	Yes	No	No	No	No	No
Path 36: Double URL decoding	No	No	No	No	No	No
Path 37: Unicode normalization	No	No	No	No	No	No
Path 38: Fullwidth form mapping from UTF-8 bare	Yes	No	No	No	No	No
Path 39: Supports PATH_INFO	Yes, configurable	Yes, configurable	Yes, configurable	Yes, configurable	Yes, configurable	Yes, configurable
Path 40: Supports path segment parameters	No	No	No	No	No	Yes
Path 41: Supports short filenames on Windows	No	No	No	No	Yes	No
Path 42: Supports Alternate Data Streams (ADS)	No	No	No	No	No	No

Path Handling of Major Platforms

	Test	IIS 5.1	IIS 6.0	IIS 7.0	IIS 7.5	Apache 2.x	Tomcat 6.x
4	Path 02: Supports %uHHHH encoding	Yes	Yes	Status 400	Status 400	Status 400	Status 400
5	Path 03: Supports UTF-8 in filenames (encoded)	No	Yes	Yes	Yes	Yes (pass-through)	Configurable
6	Path 04: Supports UTF-8 in filenames (bare)	No	No	No	No	Yes (pass-through)	Configurable
7	Path 05: Performs best-fit mapping for %u	Yes	No (404; logs best	Status 400	Status 400	Status 400	Status 400
8	Path 06: Performs best-fit mapping for bare UTF-8						
9	Path 07: Performs best-fit mapping for encoded UTF-8						
10	Path 08: Invalid %HH encoding handling						
11	Path 09: Invalid %uHH encoding handling						
12	Path 10: Valid vs invalid %HH preference (e.g., d.txt vs %64.txt)						
13	Path 11: Valid vs invalid %HHHH preference						
14	Path 12: NUL byte (encoded)						
15	Path 13: NUL byte (bare)						
16	Path 14: Backslash as path segment separator						
17	Path 15: Forward slash as path segment separator (%u-encoded)						
18	Path 16: Forward slash as path segment separator (URL-encoded)						
19	Path 17: Backslash as path segment separator (URL-encoded)						
	segment separator (%u-encoded)						
	encoded						
	bare						
	uences (non-separators) 2-byte sequen						
	uences (non-separators) 3-byte sequen						
	uences (non-separators) 4-byte sequen						
	uences (non-separators) 2-byte sequen						
	uences (non-separators) 3-byte sequen						
	uences (non-separators) 4-byte sequen						
	uences (separators) 2-byte sequence -						
	uences (separators) 3-byte sequence -						
31	Path 29: Overlong UTF-8 sequences (separators) 4-byte sequence -						
32	Path 30: Overlong UTF-8 sequences (separators) 2-byte sequence -						
33	Path 31: Overlong UTF-8 sequences (separators) 3-byte sequence -						
34	Path 32: Overlong UTF-8 sequences (separators) 4-byte sequence -						
35	Path 33: Fullwidth form mapping from %u encoding						
36	Path 34: Invalid UTF-8 encoding (encoded)						
37	Path 35: Fullwidth form mapping from UTF-8 encoded						
38	Path 36: Double URL decoding						
39	Path 37: Unicode normalization						
40	Path 38: Fullwidth form mapping from UTF-8 bare						
41	Path 39: Supports PATH_INFO	Yes, configurable	Yes, configurable	Yes, configurable	Yes, configurable	Yes, configurable	Yes, configurable
42	Path 40: Supports path segment parameters	No	No	No	No	No	Yes
43	Path 41: Supports short filenames on Windows	No	No	No	No	Yes	No
44	Path 42: Supports Alternate Data Streams (ADS)	No	No	No	No	No	No

42

TESTS

3 PARAMETER EVASION

Parameter Cardinality and Case

- In the simplest case, supplying multiple parameters or varying the case of parameter names may work:

  `/myapp/admin.php?userid=1&userid=2`

  `/myapp/admin.php?uSeRiD=1&userid=2`

- However, these techniques are more likely to work against custom-coded defenses; WAFs will have caught up by now.

PHP's Cookies as Parameters

- PHP can be configured to treat cookies as parameters, and place them in the $_REQUEST array:

```
GET /myapp/admin.php
Cookie: userid=X
```

- This is still the default behaviour in the code, with an override in the default php.ini (which can easily be misconfigured).

HTTP Parameter Pollution

- Depending on the backend and the code used, the WAF may not know exactly that the application sees:

  `/myapp/admin.php?userid=1&userid=2`

Technology	Behaviour	Result
ASP	Concatenate	userid=1,2
PHP	Last occurrence	userid=2
Java	First occurrence	userid=1

A better overview is available in the *HTTP Parameter Pollution* slides.

Tricks with PHP Parameter Names

- PHP will change parameter names when they contain some characters it does not like:
  - Whitespace at the beginning is removed
  - Whitespace, dot, and open bracket characters in the middle converted to underscores

```
/myapp/admin.php?+userid=X
```

Invalid URL Encoding

- Different platforms react differently to invalid encoding.
- ASP removes a % character that is not followed by 2 hexadecimal digits:

  /myapp/admin.php?user%id=X

- In the old days, many C-based applications had incorrect decoding routines, which lacked error detection.

  /myapp/admin.php?user%}9d=X

  /myapp/admin.php?user%69d=X

Content Type Evasion

- When parameters are transported in request body, you can attack the encoding detection mechanism
  - Attack applications that hard-code processing:
    - Omit the `Content-Type` request header
    - Place an arbitrary value in it
    - Use `multipart/form-data`, and craft the request body to be a valid multipart payload (the app will still parse as Urlencoded)
  - Attack apps with lax content type detection:
    - For example, Apache Commons FileUpload accepts any MIME type that begins with `multipart/` as `multipart/form-data`
  - Use less common formats, such as JSON
  - Use a different transport, for example WebSockets

ModSecurity Bypass

- By default, ModSecurity ignores unknown MIME types
  - With Apache Commons FileUpload, send a request body with `multipart/whatever` MIME type
  - Request bodies using encodings other than Urlencoded and Multipart are completely ignored
- Possible improvements to ModSecurity:
  - Fail closed upon detecting unknown MIME type
  - Inspect all request bodies as a stream of bytes

4 MULTIPART EVASION

Multipart Format Overview

```
POST / HTTP/1.0
Content-Type: multipart/form-data; boundary=0000
Host: www.example.com
Content-Length: 10269

--0000
Content-Disposition: form-data; name="name"

John Smith
--0000
Content-Disposition: form-data; name="email"

john.smith@example.com
--0000
Content-Disposition: form-data; name="image"; filename="image.jpg"
Content-Type: image/jpeg

FILE CONTENTS REMOVED
--0000--
```

1

2

3

4

5

Apache Commons FileUpload

- Define constant for later use:

```
public static final String
    MULTIPART = "multipart/";
```

- Determine if Multipart request body is present:

```
if (contentType.toLowerCase().
    startsWith(MULTIPART)) {
    return true;
}
```

ModSecurity CRS Bypass

- ModSecurity Core Rules will attempt to restrict MIME types, but not always successfully:
  - With Apache Commons FileUpload, send a request body with `multipart/` MIME type.
  - Reported as fixed in CRS 2.2.5.
- The flaw was in this rule, where the check was not strict enough:

```
SecRule REQUEST_CONTENT_TYPE "!@within \
    application/x-www-form-urlencoded \
    multipart/form-data"
```

Content-Type Evasion

- Trick the WAF into not seeing a Multipart request body
- Examples:

```
Content-Type: multipart/form-data ; boundary=0000
Content-Type: mUltiPart/ForM-dATa; boundary=0000
Content-Type: multipart/form-dataX; boundary=0000
Content-Type: multipart/form-data, boundary=0000
Content-Type: multipart/form-data boundary=0000
Content-Type: multipart/whatever; boundary=0000
Content-Type: multipart/; boundary=0000
```

ModSecurity with Apache Commons FileUpload bypass

PHP Source Code

```
boundary = strstr(content_type, "boundary");
if (!boundary) {
    /* Lowercase header and try again */
}

if (!boundary ||
    !(boundary = strchr(boundary, '='))) {
    /* Return with error */
}
```

1

2

3

Boundary Evasion

- Trick the WAF into seeing a different boundary
- Examples:

```
Content-Type: multipart/form-data;
 boundary =0000; boundary=1111
Content-Type: multipart/form-data;
 boundaryX=0000; boundary=1111
Content-Type: multipart/form-data;
 boundary=0000; boundary=1111
Content-Type: multipart/form-data;
 boundary=0000; BOUNDARY=1111
Content-Type: multipart/form-data;
 boundary=0000'1111
```

Reported by Stefan Esser in 2009 to have worked against F5

Part Evasion

- Boundary evasion leads to part evasion, but even when you get the boundary right you can still miss things
- In 2009, Stefan Esser reported that PHP continues to process the parts that appear after the “last” part

```
--0000
Content-Disposition: form-data; name="name"

John Smith
--0000--
Content-Disposition: form-data; name=“name"

ATTACK
--0000
```

Parameter Name Evasion

- Focuses on differences in parameter name parsing.
- Example attacks:

```
Content-Disposition: form-data; name="n1"; name="n2"
Content-Disposition: form-data; name="n1"; name ="n2"
```

- How PHP parses parameter names:

```
Content-Disposition: form-data; name="n1"; name="n2"
Content-Disposition: form-data; name="n1"; name ="n2"
```

Parameter Type Evasion

- WAFs may treat files differently. For example:
  - ModSecurity has different inspection controls for files
  - No file inspection in the CRS
- ModSecurity bypass reported by Stefan Esser in 2009
  - Thought to have been fixed (I was not involved)
  - Stefan's original payload below

```
Content-Disposition: form-data;
 name=';filename="";name=payload;"
```

Parameter Type Evasion

- This is what ModSecurity saw:

```
Content-Disposition: form-data;
 name=';filename="';name=payload;"
```

name filename

- This is what PHP sees:

```
Content-Disposition: form-data;
 name=';filename="';name=payload;"
```

name (ignored) name

Parameter Type Evasion

- Flaw thought to have been fixed
  - I rediscovered the problem during my evasion research
- The original problem had been misunderstood and addressed incorrectly:
  - ModSecurity added support for single quotes in **parameter values**
  - PHP supports single-quote escaping **anywhere within the C-D header**

- New ModSecurity bypass* with only **1** extra character:

```
Content-Disposition: form-data;
 name=x';filename="';name=payload;"
```

(*) Reported to have been addressed in ModSecurity 2.6.6

Multipart Evasion Summary

- Complex and vaguely specified format
- Implementations are often:
  - Quick & dirty (whatever works)
  - Focused on real-life use cases (not the specification)
- Rife opportunities for evasion
- There are 37 tests available in the repository
  - Tested against ModSecurity and PHP
  - Testing of the major platforms will follow soon

37
TESTS

5 WHAT NEXT?

Future Work

- At this time:
  - Path handling has good coverage (tests + results)
  - Parameter handling and multipart test cases in good shape
    - Need to test major platforms
- Future activity
  - Complete other areas of protocol-level evasion
    - HTTP parsing
    - Character set issues
    - Hostname evasion
  - Document all techniques in the *Evasion Techniques Catalogue*

Where to Go From Here

- More information in the accompanying whitepaper
- Get the tools and docs from GitHub: https://github.com/ironbee/waf-research
  - Path handling research
  - Baseline, path, and multipart test cases
- Test your security products
- Contribute your results

>100 TESTS

IRONBEE™
Open Source WAF

github
SOCIAL CODING

QUALYS®

black hat® USA 2012

Thank You

Ivan Ristic

iristic@qualys.com

Twitter: @ivanristic

How to Write a Good Virtual Patch

- Take these steps to write a good virtual patch:
  1. Study the problem, ideally by reading source code
     - If the source code is not available, do what you can by analyzing the advisory, the exploit, and by attacking the application
  2. Use a path that can withstand evasion attempts
  3. Enumerate all parameters
  4. For each parameter
     1. Determine how many times it can appear in request
     2. Determine what it is allowed to contain
  5. Reject requests with unknown parameters
- Outside the patch, enforce strict configuration that does not allow requests with anomalies

Baseline Tests

- In the repository, there is a set of baseline tests designed to determine if all parts of a HTTP requests are inspected by a WAF
- Instructions:
  1. Find one payload that is blocked by the WAF
  2. Submit payload in every different logical location
  3. Determine locations that are not monitored
  4. Seek ways to exploit the application in that way

22
TESTS

Why Should You Care?

- Researchers:
  - Fascinating new data, and effort to systematically and collaboratively analyse how WAFs perform in this area
- Testers (breakers):
  - Lots of practical assessment techniques
- Defenders:
  - Lots of practical information about Apache and ModSecurity
  - A better picture of the true state of your defences (and an opportunity to tell your vendor how much you care)
- Vendors:
  - Good reason to allocate more funds to the core functionality of your WAF, leading to a better product

Donald Knuth on Email

"Email is a wonderful thing for people whose role in life is to be on top of things. ***But not for me; my role is to be on the bottom of things.****"*

Previous Work

- A look at whisker's anti-IDS tactics
  Rain Forest Puppy (1999)
- Bypassing Content Filtering Software
  3APA3A (2002)
- HTTP IDS Evasions Revisited
  Daniel J. Roelker (2003)
- Snort's README.http_inspect
  Sourcefire et al (2005)
- Shocking News in PHP Exploitation
  Stefan Esser (2009)
- HTTP Parameter Pollution
  Luca Carettoni and Stefano di Paola (2009)

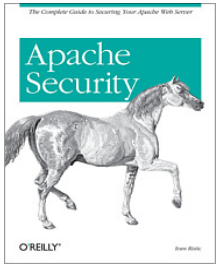
About Ivan Ristic

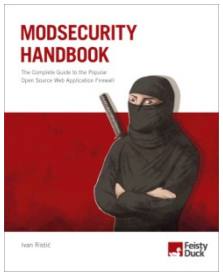
Ivan is a compulsive developer, application security researcher, writer, publisher, and entrepreneur.

- ***Apache Security***, O'Reilly (2005)
- **ModSecurity**, open source web application firewall
- **SSL Labs**, SSL/TLS, and PKI research
- ***ModSecurity Handbook***, Feisty Duck (2010)
- **IronBee**, a next-generation open source web application firewall

modsecurity

QUALYS® SSL LABS





IRONBEE™
Open Source WAF