

BACKSLASH POWERED SCANNING

Hunting Unknown Vulnerability Classes

James Kettle

Marketizer

Invalid username or password

Username:

Password:

Login

marketizer1

Who am I?

 @albinowax

Head of Research at PortSwigger Web Security



Design scanner checks

- Cross-Site Request Forgery, Client-Side Template Injection
- Server-Side Template Injection
- Burp Collaborator (asynchronous vulnerabilities)

OUTLINE

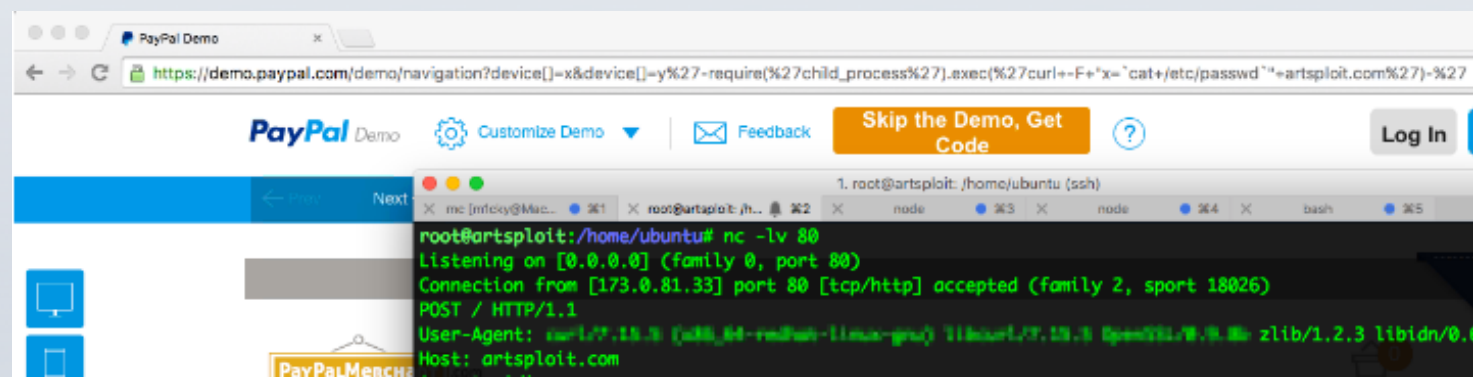
- The three failures of scanners
- Solving the Million Payload Problem
 - The clickbait approach
 - The ambitious approach
- Hunting findings
 - Scanning at scale
 - Findings, illustrations & demos
- Q&A

BLIND SPOT 1/3: RARE TECHNOLOGY

- Security through obscurity works (versus scanners)
- How many types of Server-Side Template Injection does your scanner support?

Amber, Apache Velocity, action4JAVA, ASP.NET (Microsoft), ASP.NET (Mono), AutoGen, Beard, Blade, Blitz, Casper, CheetahTemplate, Chip Template Engine, Chunk Templates, CL-EMB, CodeCharge Studio, ColdFusion, Cottle, csharptemplates, CTPP, dbPager, Dermis, Django, DTL::Fast (port of Django templates), Djolt-objc, Dwoo, Dylan Server Pages, ECT, eRuby, FigDice, FreeMarker, Genshi (templating language), Go templates, Google-ctemplate, Grantlee Template System, GvTags, H2o, HAH, Haml, Hamlets, Handlebars, Hyperkit PHP/XML Template Engine, Histone template Engine, HTML-TEMPLATE, HTTL, Jade, JavaServer Pages, jin-template, Jinja, Jinja2, JScore, Kalahari, Kid (templating language), Liquid, Lofn, Lucee, Mako, Mars-Templater, MiniTemplator, mTemplate, Mustache, nTPL, Open Power Template, Obyx, Pebble, Outline, pHAML, PHP, PURE Unobtrusive Rendering Engine, pyratemp, QueryTemplates, RainTPL, Razor, Rythm, Scalate, Scurvy, Simphple, Smarty, StampTE, StringTemplate, SUIT Framework, Template Attribute Language, Twital, Template Blocks, Template Toolkit, Thymeleaf, TinyButStrong, Tonic, Toupl, Twig, Twirl, uBook Template, vlibTemplate, WebMacro, ZeniTPL, BabaJS, Rage, PlannerFw, Fenom

- `{{7*7}}`



BLIND SPOT 2/3: Variants & filters

- How do we detect blind eval() injection

```
" .sleep(10) ."
```

- If parenthesis is filtered?

```
" . `sleep 10` ."
```

False Negative

- If there's a WAF?

```
" .s1%D0%B5ep(10) ."
```

(Cyrillic e)

False Negative

- If " is filtered?

```
{${sleep(10)}}
```

False Negative

- SQLi in double quotes

BLIND SPOT 3/3: Buried vulnerabilities

```
GET /search/?q=david&q[1]=sec${phpinfo()} } } HTTP/1.1
Host: sea.ebay.com.sg
User-Agent: Mozilla/5.0 etc Firefox/49.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://sea.ebay.com.sg/
Cookie: session=pZGFjciI6IjAkLCJlx2V4cCI6MTA4
Connection: close
Origin: null
X-Forwarded-For: 127.0.0.1
X-Forwarded-Host: evil.com
```

A SCANNER PROOF APPLICATION

- Code with an ancient, obscure web language
- Store data with a NoSQL variant, crazy syntax preferable
 - If you must use SQL, use double-quotes
- Layer a few WAFs on top

```
SELECT id FROM users WHERE user="$username"
```

```
" onmouseover=alert(1)
```

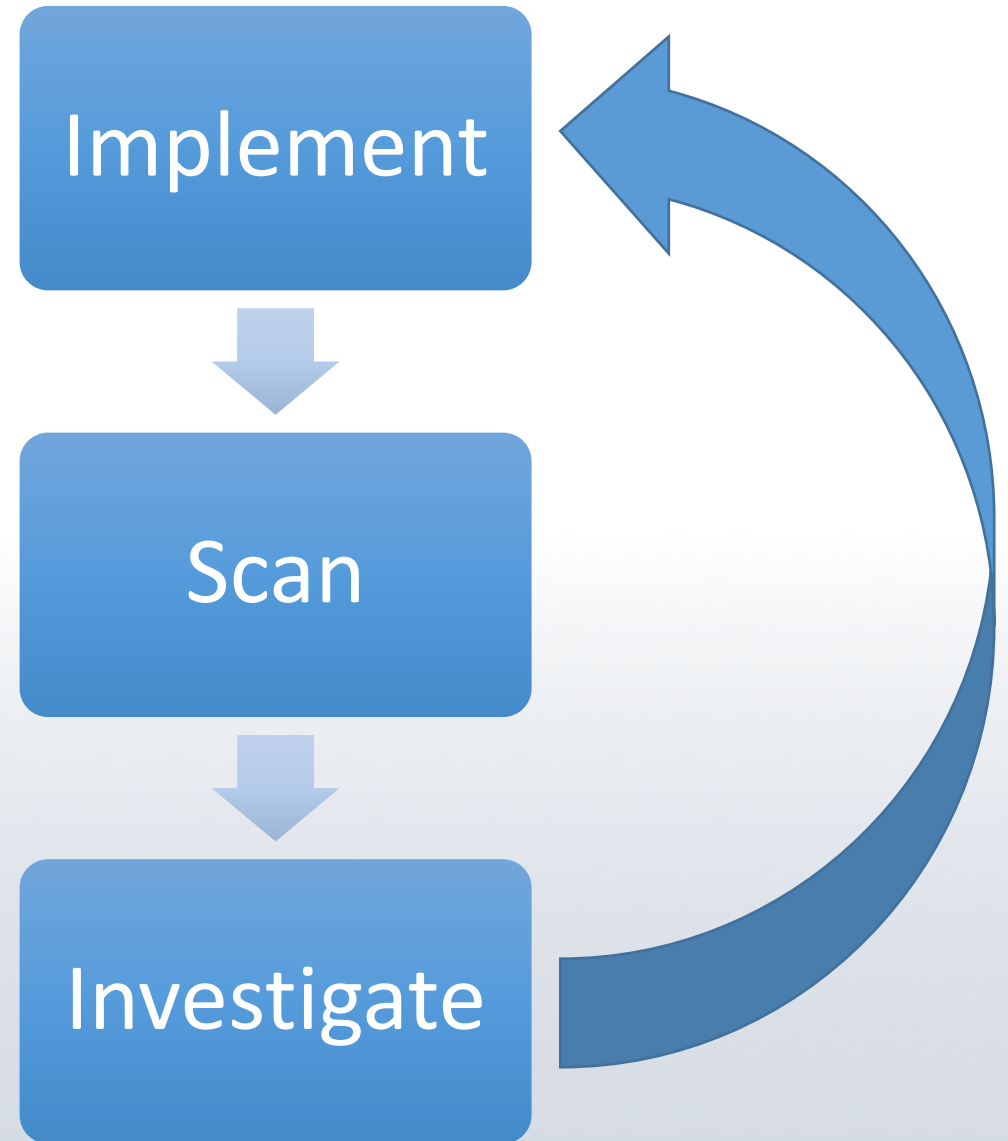

The Million Payload Problem

IDENTIFYING SUSPECTS

Don't scan for vulnerabilities

Scan for suspicious behaviour

Iteratively gather evidence



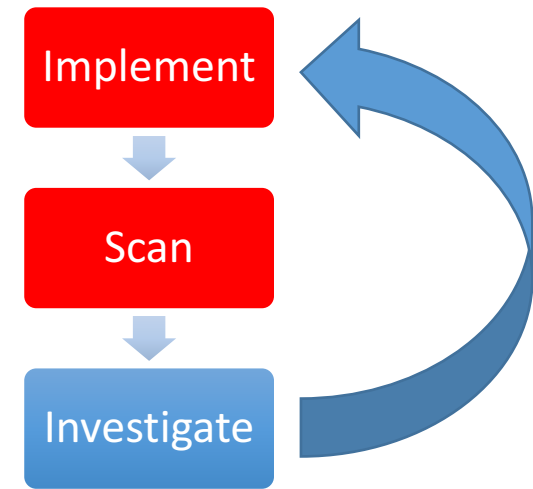
BACKSLASH CONSUMPTION

`{7*7}` => 49

`7*7` => 49

`\x41` => A

`\\` => \



BACKSLASH CONSUMPTION

Get baseline:

`\zz => \zz`

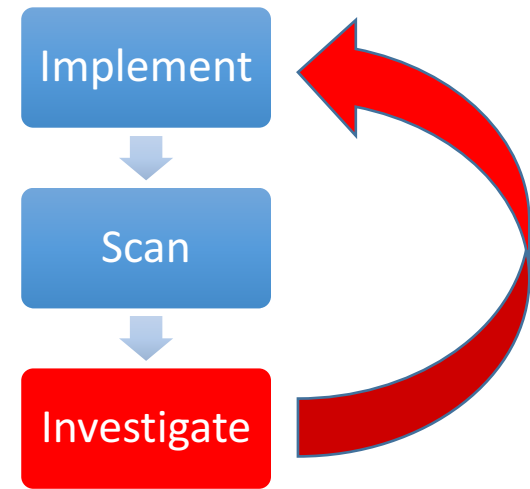
Look for anomalies:

`\" => \"`

`\$ => \$`

`\{ => {`

`\x41 => \x41`



? Suspicious Input Transformation

? Suspicious Input Transformation

Issue: Suspicious Input Transformation
Severity: High
Confidence: Tentative
Host: http://codepen.io
Path: /preprocessors

Issue: Suspicious Input Transformation
Severity: High
Confidence: Tentative
Host: https://www.secnews.gr
Path: /

Note: This issue was generated by the Burp extension: protoScan2.

Note: This issue was generated by the Burp extension: Backslash Powered Scanner.

Issue detail

Issue detail

The application transforms input in a way that suggests it might be vulnerable to some kind of server-side code injection
Affected parameter:1
Interesting transformations:

The application transforms input in a way that suggests it might be vulnerable to some kind of server-side code injection
Affected parameter:s
Interesting transformations:

- \{ => {
- { => {
- \} => }
- } => }
- \ (=> (
- (=> (
- \) =>)
-) =>)
- \ [=> [
- [=> [
- \] =>]
-] =>]
- \ ` => `
- ` => `
- \ # => #
- # => #
- \ & => &
- & => &
- \ | => |
- | => |
- \ ^ => ^
- ^ => ^

- \0 =>

Server-Side Markdown Injection

Boring transformations:

Boring transformations:

- \101 => \101
- \x41 => \x41
- \u0041 => \u0041
- \0 => \0
- \1 => \1
- \' => \'
- \" => \"
- \\$ => \\$
- \ => \

- \101 => 101
- \x41 => x41
- \u0041 => u0041
- \1 => 1
- \x0 => x0
- ' => '
- " => "
- { => {
- } => }
- (=> (
-) =>)
- [=> [
-] =>]
- \$ => \$
- ` => `
- / => /
- @ => @
- # => #
- ; => ;
- % => %
- & => &
- | => |
- ; => ;
- ^ => ^
- ? => ?

stripslashes

BACKSLASH CONSUMPTION FLAWS&FIXES

JSON output encoding

```
if (Content-Type == text/json) decode_json()
```



Accidental unicode

```
foo\\u0 => foo\u00255c\u00255cu0
```

Tighten post-backslash charset



Relies on processed-input reflection

Fundamental design flaw



DIFFING

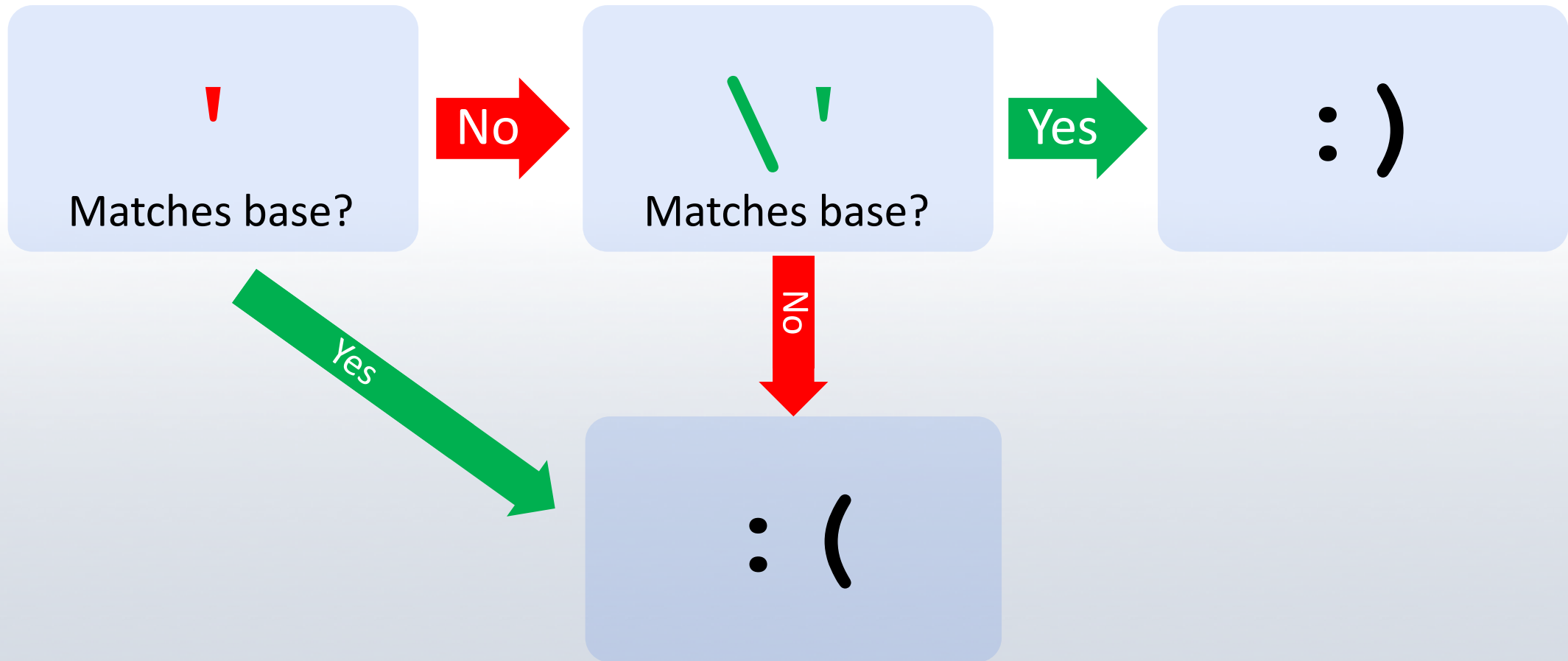


break



don't break

DIFFING



TWO TYPES OF MUTATIONS

- Distinct response on certain syntax

<code>/post_comment?text=baseComment</code>	200	OK
<code>/post_comment?text=randomtext</code>	200	OK
<code>/post_comment?text=random' text</code>	500	Oops
<code>/post_comment?text=random\ ' text</code>	200	OK

- Syntax error indistinguishable from incorrect value

<code>/profile?user=bob</code>	200	OK
<code>/profile?user=randomtext</code>	500	Oops
<code>/profile?user=random' text</code>	500	Oops
<code>/profile?user=random\ ' text</code>	500	Oops
<code>/profile?user=bo' 'b</code>	200	OK
<code>/profile?user=bo' z' b</code>	500	Oops

EXACT RESPONSE MATCHING: A BAD IDEA

HTTP Headers change order

Sort headers

Timestamps change

Regex them out

Applications reflect input

Regex out input

The input is $x=0$, can't regex that

Pad input with leading zeros

Responses contain outright random content

Repeat requests, merge using Longest-Common-Subsequences

Responses sometimes alternate

Mix up probe order

Deterministic transformations of input

Use probe batches: $x/1$ vs

Caches make random content permanent

Add cachebuster

Two distinct responses

multiple fingerprints

<https://github.com/wp-plugins/leaflet-maps-marker/blob/master/leaflet-georss.php>

CLEARING THINGS UP

- Assert on what's consistent
 - Status code, content type, tag structure, line count, word count
 - Keywords
 - Leading/trailing characters
 - Reflection count

- We made a Burp Extender API for this:

```
responseDetails.updateWith(response1);  
responseDetails.updateWith(response2);  
List<String> consistentDetails =  
    responseDetails.getInvariantAttributes();
```

SURVEY

- Does the application react to fuzzing?

Yes: `\z`z'z"\` **vs** `\`z\'z\"z\\`

- Which part of the fuzz string caused the reaction?

Quote: `z"\z` **vs** `z\"z`

- Which characters work for concatenation?

Plus: `z"z"z` **vs** `z"+"z`

- Can I call a generic function?

Yes: `"+abz(1)+"` **vs** `"+abs(1)+"`

- Can I call a language-specific function?

JavaScript: `"+isBlah(1)+"` **vs** `"+isFinite(1)+"`



Fuzzable: JavaScript injection

Issue: **Fuzzable: JavaScript injection**
Severity: **High**
Confidence: **Firm**
Host: **http://codepen.io**
Path: **/preprocessors**

Note: This issue was generated by the Burp extension: protoScan2.

Issue detail

The application reacts to inputs in a way that suggests it might be vulnerable to some kind of server-side code injection. The probes are listed below in chronological order.

Successful probes

- **Basic fuzz** (`\z`z'z"\` vs \`z\'z"\`)`
 - error: **2** vs **1**
 - Content: **17** vs **3**
- **String – doublequoted** (`\zz" vs \"`)
 - error: **2** vs **1**
 - Content: **16** vs **3**
- **Concatenation: "||"** (`z||"z(z"z vs z(z"||"z)`)
 - error: **2** vs **1**
- **Concatenation: "+"** (`z+"z(z"z vs z(z"+"z)`)
 - error: **2** vs **1**
 - Content: **16** vs **3**
- **Concatenation: "&"** (`z&"z(z"z vs z(z"&"z)`)
 - error: **2** vs **1**
 - Reflection count: **3** vs **0**
- **JavaScript injection** (`"+isFinitee(1)+" vs "+isFinite(1)+"`)
 - error: **2** vs **1**
 - Content: **11** vs **3**

THE ARSENAL

- String injection
- Number: `37/0` vs `37/1` ... `37/power(unix_timestamp(),0)`
- Interpolation: `${{` vs `}}`
- OrderBy: `1,abs(1,2)` vs `1,abs(1)`

- Comment: `/**/z*/` vs `/*zz*/`
- Function: `sprintg` vs `sprintf`

HUNTING FINDINGS

EVALUATING TESTBEDS



Realism

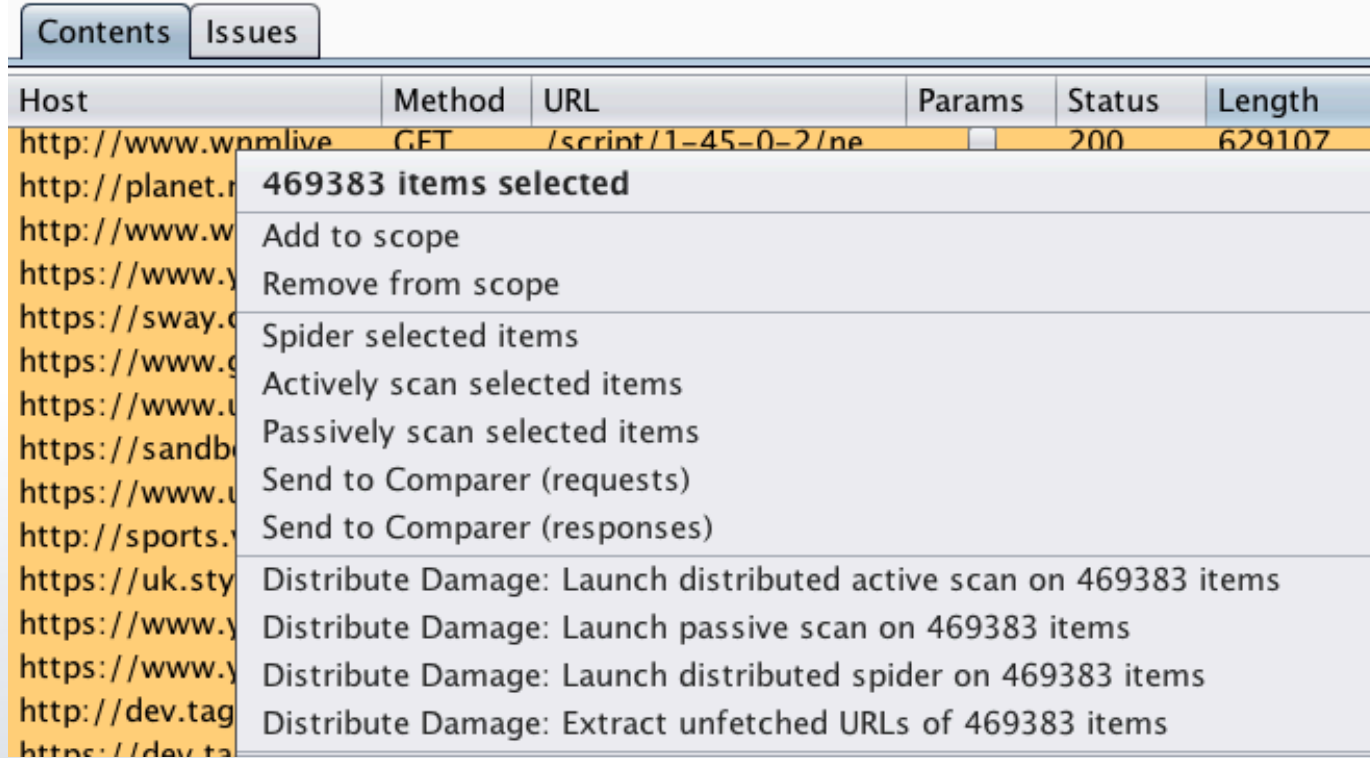
- Hand coded labs
 - Absolute control
- OWASP Broken Web Apps
 - Source code access
 - Mildly unrealistic, only so large
- Pentests
 - Limited supply
- Every bug-bounty site
 - Free cash
 - **MIDNIGHT BLACK BOX**



Code insight

TESTING AT SCALE

- Requirements
 - Per-domain throttling
 - High net speed
 - Attack-surface optimisation
- distributeDamage
 - Interleave target hosts
 - Extract URLs to file for spidering
 - Scan each parameter once per site per response type



The screenshot shows a web application interface with two tabs: "Contents" and "Issues". Below the tabs is a table with columns: Host, Method, URL, Params, Status, and Length. The table contains several rows of data, with the first row highlighted in orange. A context menu is open over the first row, listing various actions such as "469383 items selected", "Add to scope", "Remove from scope", "Spider selected items", "Actively scan selected items", "Passively scan selected items", "Send to Comparer (requests)", "Send to Comparer (responses)", "Distribute Damage: Launch distributed active scan on 469383 items", "Distribute Damage: Launch passive scan on 469383 items", "Distribute Damage: Launch distributed spider on 469383 items", and "Distribute Damage: Extract un fetched URLs of 469383 items".

Host	Method	URL	Params	Status	Length
http://www.wnmlive	GET	/script/1-45-0-2/ne		200	629107
http://planet.r					
http://www.w					
https://www.y					
https://sway.c					
https://www.g					
https://www.u					
https://sandb					
https://www.u					
http://sports.y					
https://uk.sty					
https://www.y					
https://www.y					
http://dev.tag					
https://dev.ta					

SAMPLE - EASY

Basic fuzz (\z`z'z"\ vs \`z\'z\"\\)

Content: 5357 vs 5263

String - apostrophe (\zz'z vs z\\\ 'z)

Content: 5357 vs 5263

Concatenation: '|| (z||'z(z'z vs z(z' ||'z)

Content: 5357 vs 5263

Basic function injection ('||abf(1)||' vs '||abs(1)||')

Content: 5281 vs 5263

MySQL injection ('||power(unix_timestamp(),0)||' vs
'||power(unix_timestamp(),0)||')

Content: 5281 vs 5263

SAMPLE – TRICKIER

String - doublequoted (`\zz"` vs `\"`)

- error: 1 vs 0
- Content: 9 vs 1
- Tags: 3 vs 0

Concatenation: `". (z."z(z"z` vs `z(z"."z)`

error: 1 vs 0

Content: 9 vs 1

Tags: 3 vs 0

Interpolation - dollar (`z${z` vs `}${z)`

- error: 1 vs 0
- Content: 9 vs 1
- Tags: 3 vs 0

SAMPLE - INTEL

Successful probes

- Interpolation fuzz (z%{{zz\${{z vs }}%z}}\$z)
 - Content start: **text** vs **[blank]**
 - error: **0** vs **1**
 - Status code: **200** vs **500**
 - Content: **2** vs **1**
- Interpolation - dollar (z\${{z vs }}\$z)
 - Content start: **text** vs **[blank]**
 - error: **0** vs **1**
 - Status code: **200** vs **500**
 - Content: **2** vs **1**
- Interpolation - percent (z%{{z vs }}%z)
 - Content start: **text** vs **[blank]**
 - error: **0** vs **1**
 - Status code: **200** vs **500**
 - Content: **2** vs **1**

Raw	Headers	Hex
<pre>http/1.1 200 ok date: fri, 30 sep 2016 13:49:34 gmt server: apache/2.4.16 (unix) openssl/1.0.1e-fips content-length: 14 connection: close content-type: text/html; charset=utf-8 access denied!</pre>		

SAMPLE – REGEX INJECTION

Backslash (\ vs \\)

```
java.lang.illegalargumentexception: character to be escaped is missing
  java.util.regex.matcher.appendreplacement(matcher.java:809)
org.tuckey.web.filters.urlrewrite.utils.regexmatcher.replaceall(regexmatcher.java:72)
```

Interesting transformations:

- \0 => Truncated
- \1 => Truncated
- \\$ => \$
- \$ => \$

```
GET /folder?q=foo\0bar HTTP/1.1
```

```
HTTP/1.1 301 Moved Permanently
```

```
Location: https://redacted.com/folder/?q=foohttp://redacted.com/folder/bar
```

SAMPLE – MYSTERY

- `\z`z'z"\` vs `\`z\'z\"\\`
- [No followups]
- `foo"z:` Set-Cookie: bci=1234; domain="foo\"z";
- `foo\:` Set-Cookie: bci=1234; domain="foo\";
- `foo"z\:` 500 Internal Server Error

SAMPLE - FALSE POSITIVE

- **Function hijacking (sprintf vs printf)**
 - `<div: 13 vs 14`

```
GET /hosting/search?q=sprintf HTTP/1.1  
Host: code.google.com
```

```
GET /hosting/search?q=sprintf HTTP/1.1  
Host: code.google.com
```

SAMPLE - INTEL

- `0/**z'*/ vs 0/*/*/z'*/`
- `0<!--foo--> vs 0<!--foo->`
- `0<iframe> vs 0<zframe>`
- A WAF is re-writing requests to remove comments
- Effectively disables browser XSS filters \o/

SAMPLE – JSON/SOLR

- **Basic fuzz** (`\z`z'z"\` vs `\`z\'z\"\\`)
 - Content: **1578** vs **1575**
- **Backslash** (`\` vs `\\`)
 - Content: **1576** vs **1575**
- **String - doublequoted** (`\zz"` vs `\"`)
 - Content: **1578** vs **1575**
- `\u006d\u0069\u0072\u0072\u006f\u0072` => mirror
- Apache Solr JSON API

DEMOS

LESSONS LEARNED

- Payload iteration is invaluable
 - Minimize iteration size
- Beware search functions, WAFs, and regex injection
- Scanners can gather intelligence
- Approach with an open mind

- Per-host throttling isn't perfect

COMING SOON: ITERABLE INPUT DETECTION

- `/edit_profile?id=734`
- How do we determine id is iterable?
 - id=734, id=735 and id=736 are distinct
 - Could be encryption, seed...
- We're interested in where there's a finite number of entries
 - id=10735 and 10736 are the same
- Are we supposed to see id=735?

FURTHER RESEARCH

- Zero-info username enumeration
- Guessing params (extract/mass-assignment)
 - SSTI
- Detecting backend parameter pollution
- Fishing for objects
- Control flow mapping (page=blah)
- Detect spellchecking (implies eval())
 - Send thier, grep for their

RESOURCES

Backslash Powered Scanner code:

<https://github.com/portswigger/backslash-powered-scanner>

DistributeDamage code:

<https://github.com/portswigger/distribute-damage>

Whitepaper:

<http://blog.portswigger.net/2016/10/backslash-powered-scanning.html>

TAKE-AWAYS

Use generic payloads then iterate

Lean on the operator's strengths

Scanners can find research grade vulnerabilities

 @albinowax

james.kettle@portswigger.net

