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?


• {{7*7}}

http://artsploit.blogspot.co.uk/2016/08/pprce2.html
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?  
  ".$\%D0%B5ep(10)."  (Cyrillic е)
  False Negative

• If " is filtered?  
  "{$sleep(10)}"  
  False Negative

• SQLi in double quotes
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
Cookie: session=pZGFjciI6IjAkLCJlx2V4cCI6MTA4
Connection: close
Origin: null
X-Forwarded-For: 127.0.0.1
X-Forwarded-Host: evil.com

http://secalert.net/2013/12/13/ebay-remote-code-execution/
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 \times 7 \} \implies 49 \$

\[ 7 \times 7 \implies 49 \]

\[ \backslash x 41 \implies A \]

\[ \backslash \backslash \implies \backslash \]
BACKSLASH CONSUMPTION

Get baseline:
\z z \rightarrow \z z

Look for anomalies:
\" \rightarrow \"
\$ \rightarrow \$
\{ \rightarrow { 
\x41 \rightarrow \x41
Suspicious Input Transformation

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

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

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:
Interesting transformations:

```plaintext
\{ => {
  { => {
    } => }
  } =>
  ( => (  
    ) => )
  ) =>

\{ => [  
  [ => [  
    ] => ]
  ] => ]

\' => 
  \` => 
  \# => #
  # => #
  \& => &
  \$ => $
  \| => |
  ^ => ^
```

Boring transformations:

```plaintext
\\101 => \101
\\x2d => x2d
\\u0041 => u0041
\\s => 1
\\x0 => x0
\" => "
\( => (  
  ) => )
  ) => )
  ] => ]
  $ => $
  => `  
  / => /
\@ => @
\# => #
\% => %
\& => &
  ] => |
  => ^
```

Suspicious Input Transformation

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

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

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:
Interesting transformations:

```plaintext
\0 =>
```

Boring transformations:

```plaintext
\x0101 => 101
\x041 => x41
\x0041 => u0041
\s => 1
\x0 => x0
\" => "
\( => (  
  ) => )
  ) => )
  ] => ]
  $ => $
  => `  
  / => /
\@ => @
\# => #
\% => %
\& => &
  ] => |
  => ^
```
BACKSLASH CONSUMPTION FLAWS&FIXES

JSON output encoding
if (Content-Type == text/json) decode_json()

Accidental unicode
foo\\u0 => foo\u00255c\u00255c
Tighten post-backslash charset

Relies on processed-input reflection
Fundamental design flaw
DIFFING

break
don't break
TWO TYPES OF MUTATIONS

• Distinct response on certain syntax
  
  /post_comment?text=baseComment 200 OK
  /post_comment?text=randomtext 200 OK
  /post_comment?text=random\'text 500 Oops
  /post_comment?text=random\\text 200 OK

• Syntax error indistinguishable from incorrect value
  
  /profile?user=bob 200 OK
  /profile?user=randomtext 500 Oops
  /profile?user=random\'text 500 Oops
  /profile?user=random\\text 500 Oops
  /profile?user=bo'||b 200 OK
  /profile?user=bo'|z'b 500 Oops
EXACT RESPONSE MATCHING: A BAD IDEA

HTTP Headers change order  
Sort headers

Timetamps 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-Comment-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

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:

```java
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

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 z(z)||z"
  - error: 2 vs 1

- **Concatenation:** "+ (z+``z`` z(z)+``z"
  - error: 2 vs 1
  - Content: 16 vs 3

- **Concatenation:** "& (z&``z`` z(z)&``z"
  - error: 2 vs 1
  - Reflection count: 3 vs 0

- **JavaScript injection** (``+isFinite(1)+`` vs ```+isFinite(1)+``````
  - error: 2 vs 1
  - Content: 11 vs 3
THE ARSENAL

• String injection
• Number: 37/0 \textit{vs} 37/1 \ldots 37/power(unix\_timestamp(),0)
• Interpolation: $\{\{ \textit{vs} \} \}$
• OrderBy: 1,\textit{abs}(1,2) \textit{vs} 1,\textit{abs}(1)

• Comment: /***/z*/ \textit{vs} /**zz*/
• Function: \texttt{sprintf} \textit{vs} \texttt{sprintf}
HUNTING FINDINGS
EVALUATING TESTBEDS

• 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
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
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
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**

```
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!
```
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 (sprintg vs sprintf)
  • <div: 13 vs 14

GET /hosting/search?q=sprintg 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'\" 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