

The cookie monster in our browsers



@filedescriptor
HITCON 2019

@filedescriptor

- From Hong Kong 🇭🇰
- Pentester for Cure53
- Love WebApp Sec & Browser Sec
- Bug Bounty Hunter (#1 on Twitter's program)



Motivation

Out of Scope

Domain assets.spotify.com

Motivation

Out-of-Scope Sub-Domains:

- http://*.pornhub.com/
- http://*.pornhub.com/live/
- http://*.pornhub.com/jobs/
- http://*.pornhubpremium.com/
- http://*.pronstore.com/

Out of Scope

Domain assets.spotify.com

Motivation

 bugtriage-rob closed the report and changed the status to ● Informative. Aug 4th (3 years ago)
Thanks for your report.

While we appreciate your efforts to help keep Uber secure, I'm afraid this doesn't qualify for this program as the domain *.et.uber.com is out of scope for this program. You can find the list of in-scope properties on our program page: hackerone.com/uber

Thanks and good luck in your future bug hunting.

 raghav_bisht posted a comment.
Respected...
Its a request Once you patched the vulnerability "Do disclose the report"

 lyoung-uber reopened this report.

 lyoung-uber closed the report and changed the status to ● Not Applicable.
Closing as Not Applicable since this is out-of-scope.

 raghav_bisht posted a comment.
@lyoung-uber you fucking asshole mother fucker I know this is "Out of scope" and your team member @bugtriage-rob marked it has Informative and closed the report, still I didn't argue about it and accepted it.....fucker.
I respectfully asked you to disclosure my report and you moron mother fucker deducted my Reputation Point

Bloody Mother Fucker..... TAXI DRIVER.....

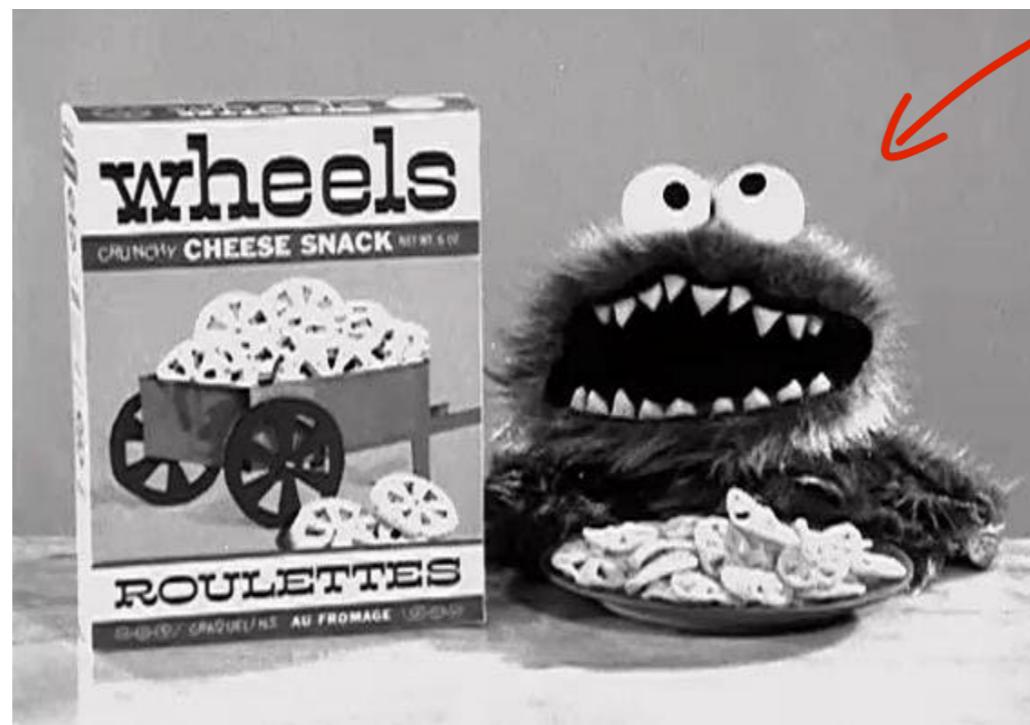
Out-of-Scope Sub-Domains:

- http://*.pornhub.com/
- http://*.pornhub.com/live/
- http://*.pornhub.com/jobs/
- http://*.pornhubpremium.com/
- http://*.pronstore.com/

Out of Scope

Domain	assets.spotify.com
--------	--------------------

1966



History

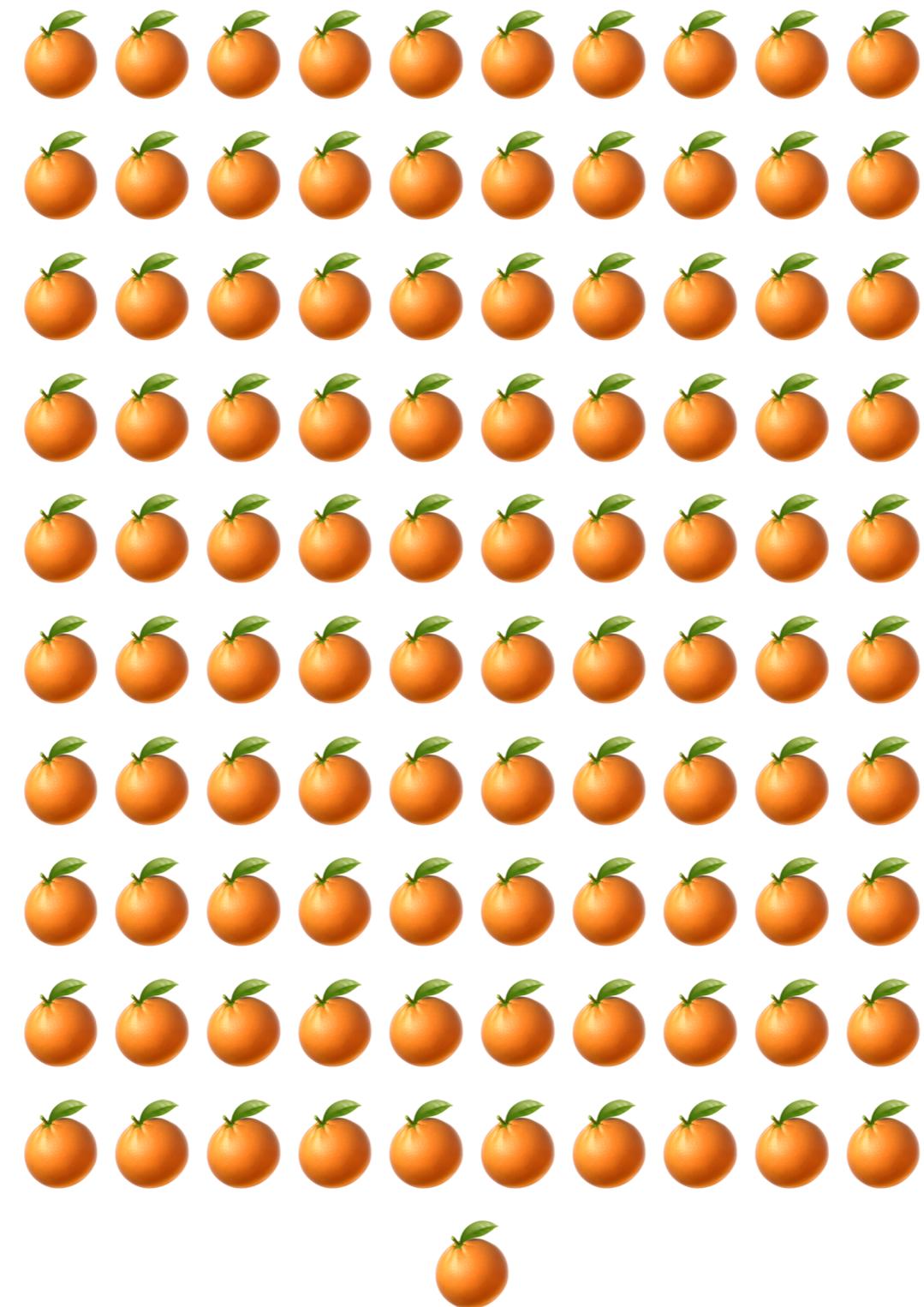
The Dark Age

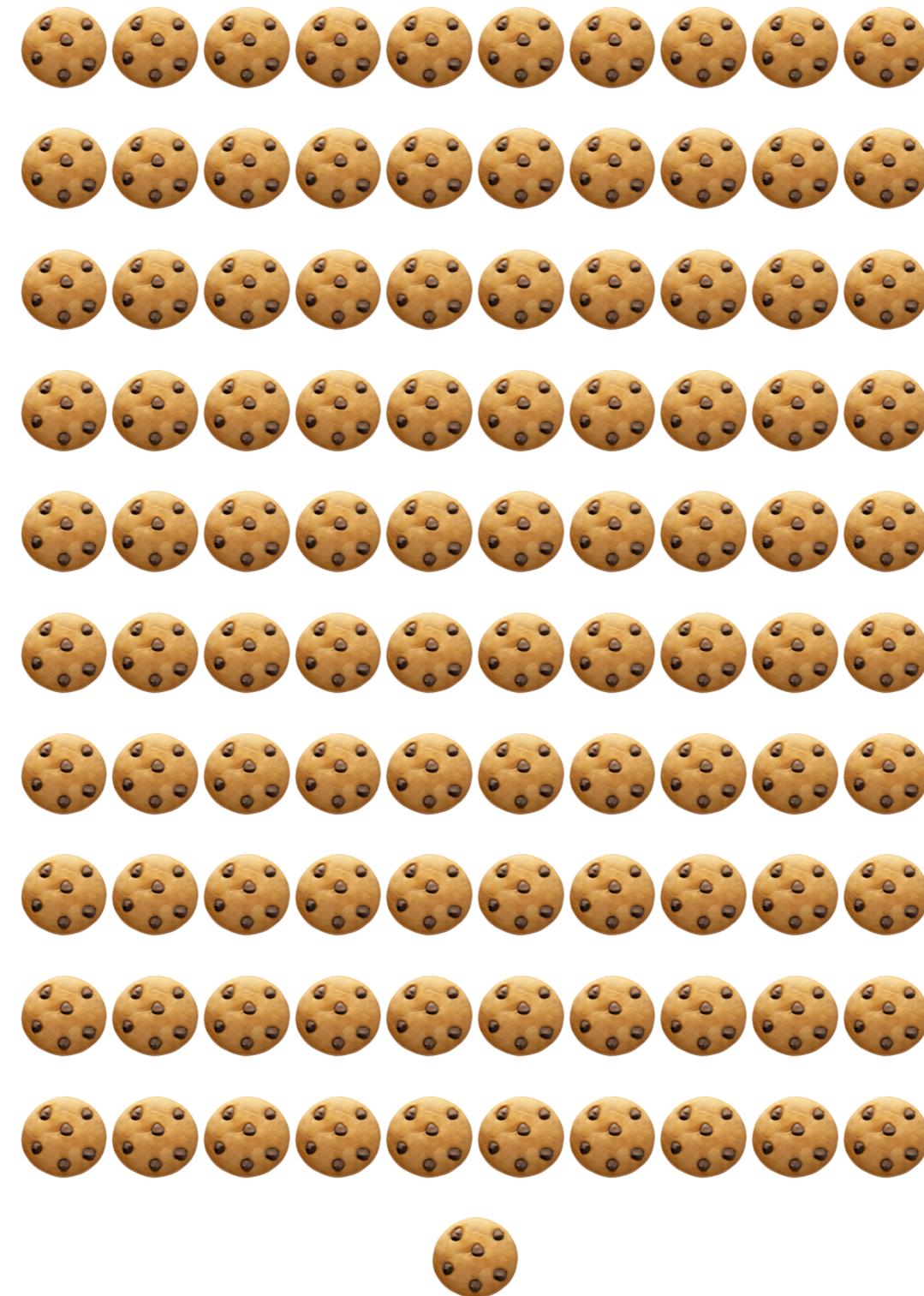
1994	1997	2000
Netscape's cookie_spec	RFC 2109	RFC 2965
Basic Syntax	More Attributes	Obsoletes RFC 2109
Mechanism	Privacy Control	Set-Cookie2 & Cookie2

No browser followed these specs!

The Modern Age

2011	2015	2016	2016
RFC 6265	Cookie Prefixes (RFC6265bis)	Same-site Cookies (RFC6265bis)	Strict Secure Cookies (RFC6265bis)
Obsoletes RFC 2965 Summarizes reality HttpOnly flag	Improves Integrity across subdomains over secure channel	Kills CSRF & Co.	Prevents secure cookies overwrite from non-secure origin





HTTP Response

HTTP/1.1 200 OK

[...]

Set-Cookie: sid=123; path=/admin

JavaScript API (write)

document.cookie = 'lang=en'

HTTP Response

```
HTTP/1.1 200 OK
```

```
[ ... ]
```

```
Set-Cookie: sid=123; path=/admin
```

JavaScript API (write)

```
document.cookie = 'lang=en'
```

Subsequent HTTP Request

```
POST /admin HTTP/1.1
```

```
[ ... ]
```

```
Cookie: sid=123; lang=en
```

JavaScript API (read)

```
document.cookie  
// sid=123; lang=en
```

*Attributes do not appear in requests

Name	Value	Attribute	Flag
Set-Cookie:	sid=123; path=/admin; Secure		

Attribute						Flag
Expires	Max-Age	Domain	Path	SameSite	Secure	HttpOnly

Attribute				Flag	
Expires	Max-Age	Domain	Path	SameSite	Secure
					HttpOnly

We will focus on these attributes in this talk

Domain

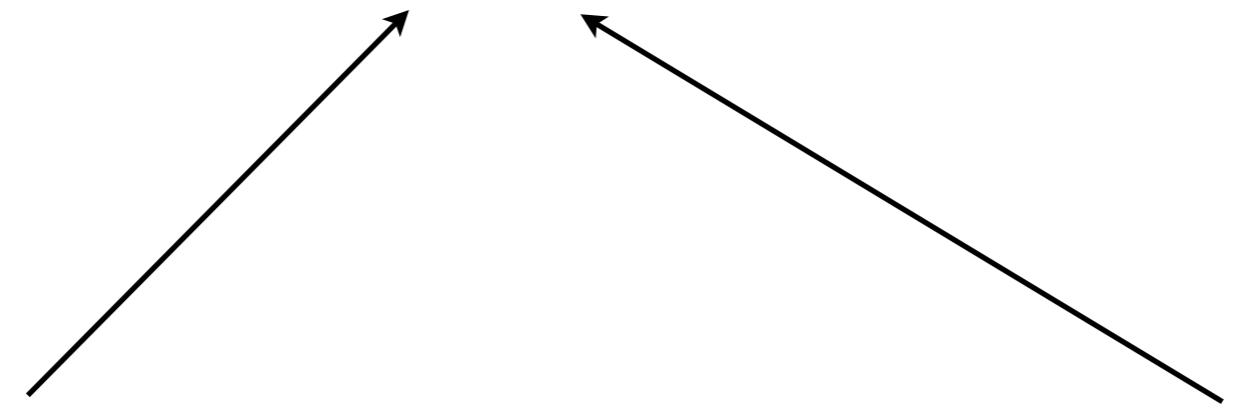
Domain to subdomains

example.com

Set-Cookie: foo=bar; domain=.example.com

sub.example.com

sub.of.sub.example.com



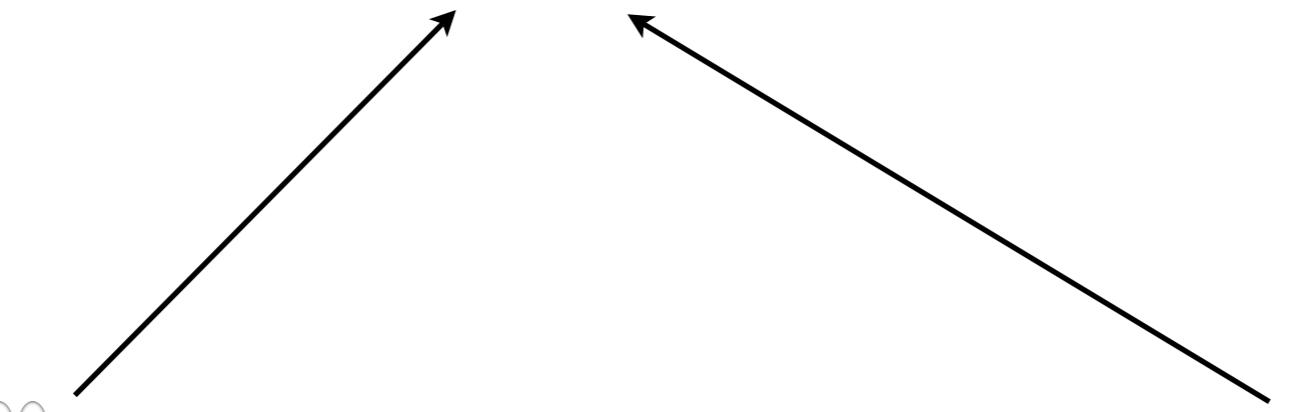
Subdomains to subdomains

sub.example.com

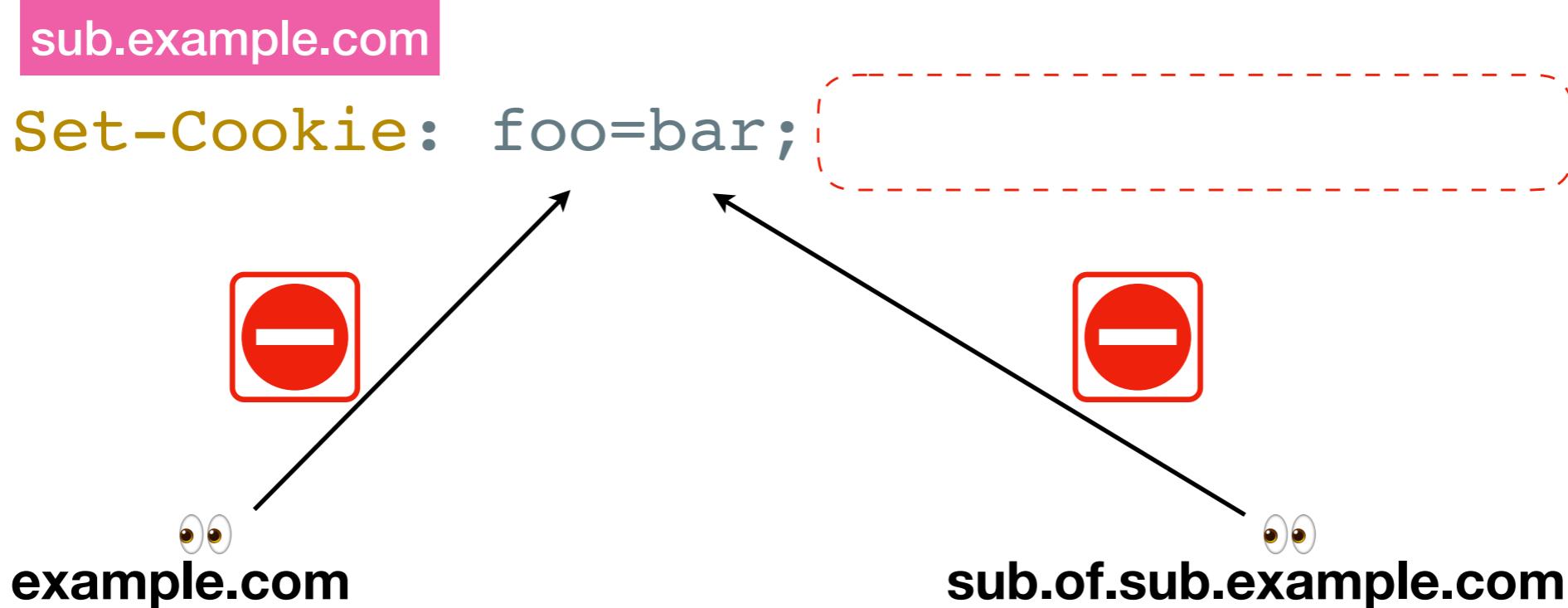
Set-Cookie: foo=bar; domain=.example.com

example.com

sub.of.sub.example.com



Current domain



THIS
IS
AVATAR **FD**
[@filedescriptor](#)

Follow



What is this cookie scoped to?

Set-Cookie: foo=bar; domain=[example.com](#)

- Only example.com
- Only *.example.com
- Both

THIS
IS
AVATAR FD
@filedescriptor

Follow



What is this cookie scoped to?

Set-Cookie: foo=bar; domain=[example.com](#)

41% Only example.com

11% Only *.example.com

48% Both

245 votes • Final results



[REDACTED] 4:13 PM

oh interesting

i learned something today

i thought that domain=example.com was "set cookie on example.com only"

and domain=.example.com was "example.com and all subdomains"

Dot or no Dot?

- They have **no** difference (old RFC vs new RFC style)
- Both widen the scope of a cookie to all (sub)domains
- The correct way to limit the scope is to **not** have the domain attribute
- Some websites add the domain attribute for all cookies
 - If one of the subdomains is compromised, such cookies will be leaked to unauthorized parties

"Some existing user agents treat an absent Domain attribute as if the Domain attribute were present and contained the current host name."

– *RFC 6265 (4.1.2.3.)*

Internet Explorer Cookie Internals X +

https://blogs.msdn.microsoft.com/ieinternals/2009/08/20/internet-explorer-cookie-internals-faq/ Incognito

This site uses cookies for analytics, personalized content and ads. By continuing to browse this site, you agree to this use. Learn more

Microsoft | Developer Search Sign in

IEInternals A look at Internet Explorer from the inside out. @EricLaw left Microsoft in 2012, and rejoined Microsoft Edge in 2018.

Follow Us 

Internet Explorer Cookie Internals (FAQ)

Q3: If I don't specify a DOMAIN attribute when a cookie, IE sends it to all nested subdomains anyway?

A: Yes, a cookie set on example.com will be sent to sub2.sub1.example.com.

Internet Explorer differs from other browsers in this regard. Here's a [test case](#).

Update: This behavior bug was removed from Edge by Windows 10 RS3, but remained in IE11 on Windows 10. By Windows 10 RS4 (April 2018), both Edge and Internet Explorer match other browsers.

Q1: IE's cookie code doesn't support max-age, etc.

A: Correct. Internet Explorer (including IE11) does not support the max-age directive. WinINET (the network stack below IE) has cookie implementation based on the pre-RFC Netscape draft spec for cookies. This means that directives like **max-age**, versioned cookies, etc, are not supported in any version of Internet Explorer.

Still isn't fixed in IE11 on Windows 7 / 8.1!

Q2: If I don't specify a leading dot when setting the DOMAIN attribute, IE doesn't care?

A: Correct. All current version browsers (Chrome, FF, Opera, etc) seem to treat a leading dot as implicit. Here's a [test case](#).

Categories: performance dev, browserInIE10 https, features limitations ie9, ie6dev ActiveX add-ons, caching Q&A

Archives

January 2015 (3)
November 2014 (1)
October 2014 (1)
September 2014 (2)



More images

Cookie bomb

A blockbuster **bomb** or **cookie** was any of several of the largest conventional **bombs** used in World War II by the Royal Air Force (RAF).

[Blockbuster bomb - Wikipedia](#)

https://en.wikipedia.org/wiki/Blockbuster_bomb

Cookie Bomb

- Most servers have a length limit on request headers
- When this limit is exceeded, HTTP 413 or 431 is returned
- Limited cookies injection can still result in client-side DoS
- Domain & Expire attributes help persist the attack across (sub)domains.

#57356 DOM based cookie b... X +

HackerOne, Inc. [US] | https://hackerone.com/reports/57356 SIGN IN | SIGN UP

Incognito (2) 🍀 ↴

hackerone

FOR BUSINESS FOR HACKERS HACKTIVITY COMPANY TRY HACKERONE

 **filedescriptor (filedescriptor)** 5655 Reputation 89th Rank 6.42 Signal 95th Percentile 37.50 Impact 99th Percentile

45 #57356 DOM based cookie bomb Share:      

State Resolved (Closed) **Severity**  No Rating (---)

Disclosed April 11, 2017 11:24am +0800 **Participants** 

Reported To Twitter **Visibility** Disclosed (Full)

Weakness Denial of Service

Bounty \$280

[Collapse](#)

TIMELINE

 **filedescriptor** submitted a report to Twitter. Apr 19th (4 years ago)
Hi,
I would like to report an issue that allows attackers to plant a "cookie bomb" on a victim's browser, so that the victim will be unable to access any Twitter services.

PoC

1. Go to <http://innerht.ml/pocs/twitter-dom-based-cookie-bomb/>
2. Click on the "DoS" link
3. Wait for a moment
4. Now Twitter will be unaccessible

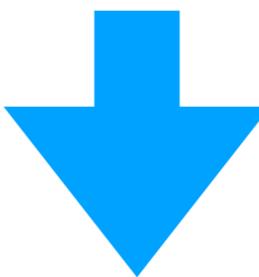


```
function d(a) {  
...  
    var b = document.referrer || "none",  
        d = "ev_redir_" + encodeURIComponent(a) + "=" + b + "; path=/";  
    document.cookie = d;  
...  
}  
...  
window.location.hash != "" && d(window.location.hash.substr(1).toLowerCase())
```

https://example.com/aaa...aaa → **https://twitter.com/#a**

https://example.com/aaa...aaa → **https://twitter.com/#b**

https://example.com/aaa...aaa → **https://twitter.com/#c**



GET / HTTP/1.1

[...]

Cookie: ev_redir_a=aaa...aaa;
ev_redir_b=aaa...aaa;
ev_redir_c=aaa...aaa

} 8kB+

Screenshot of the Chrome DevTools Network tab showing a failed request to Twitter.

The Network tab displays two requests:

- twitter.com**: Status 431, Type document, Other initiator, 48 B size, 387 ms time.
- favicon.ico**: Status 200, Type x-icon, Other initiator, 1.2 KB size, 393 ms time.

The failed request (431) is highlighted with a red box, and its details are shown in the bottom summary bar and expanded in the Cookies section.

Summary Bar Details:

Icon	twitter.com	431	docu...	Other	48 B
------	-------------	-----	---------	-------	------

Cookies Section Details:

Name	Value	Domain	Path	Expir...	Size	HTTP	Secure	First...
▼ Request Cookies								
_ga	GA1.2.193351747.1434172703	N/A	N/A	N/A	8564			
_gat	1	N/A	N/A	N/A	32			
_twitter_sess	BAh7CiIKZmxhc2hJQzonQWN0aW9uQ29ud...	N/A	N/A	N/A	313			
auth_token	cc0c2259cab2fb54883b1200b13b63169b...	N/A	N/A	N/A	53			
guest_id	v1%3A143417269970515979	N/A	N/A	N/A	34			
lang	en	N/A	N/A	N/A	9			
pid	"v3:1434173638724744160371550"	N/A	N/A	N/A	36			
remember_checked_on	1	N/A	N/A	N/A	23			
reported_tweet_id	000000000000000000000000000000000000...	N/A	N/A	N/A	4017			
reported_user_id	00...	N/A	N/A	N/A	4018			
twid	"u=2993783110"	N/A	N/A	N/A	21			
Response Cookies								

The screenshot shows a blog post titled "Cookie Bomb or let's break the Internet." by Egor Homakov. The post discusses crafting a page that can pollute CDNs, blogging platforms, and other major networks with many cookies, causing browser requests to fail due to long cookie headers. It includes a screenshot of a browser showing a "This webpage is not available" error message. The sidebar lists recent posts from 2015 and 2014.

Saturday, January 18, 2014

Cookie Bomb or let's break the Internet.

TL;DR I can craft a page "polluting" CDNs, blogging platforms and other major networks with my cookies. Your browser will keep sending those cookies and servers will reject the requests, because Cookie header will be very long. The entire Internet will look down to you.

I have no idea if it's a known trick, but I believe it should be fixed. Severity: depends. I checked only with Chrome.

We all know a cookie can only contain 4k of data.
How many cookies can I creates? **Many!**
What cookies is browser going to send with every request? **All of them!**
How do servers usually react if the request is too long? **They don't respond**, like this:

If you're able to execute your own JS on SUB1.example.com it can cookie-bomb not only your SUB1 but the entire *.example.com network, including example.com.

Posts:

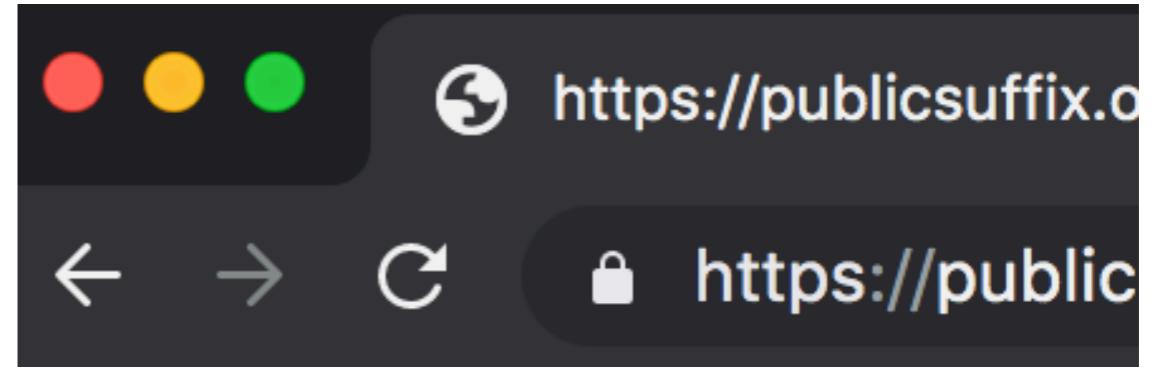
- 2015 (2)
- ▼ 2014 (19)
 - December (3)
 - November (1)
 - September (1)
 - July (1)
 - May (1)
 - March (1)
 - February (2)
 - ▼ January (9)
 - Turbo API: How to use CORS without Preflights
 - Two "WontFix" vulnerabilities in Facebook Connect
 - Header injection in Sinatra/Rack
 - Cookie Bomb or let's break the Internet.
 - Account hijacking on MtGox
 - Evolution of Open Relational Model

Shared domains're vulnerable by design

e.g. `github.io`

Public Suffix List

- Community curated
- Some domains cannot have cookies
- The same list that restricts domain=.com.tw



```
// GitHub, Inc.  
// Submitted by Patrick Toomey  
github.io  
githubusercontent.com
```

```
// GitLab, Inc.  
// Submitted by Alex Hanselka  
gitlab.io
```

```
// Glitch, Inc : https://glitch.com  
// Submitted by Mads Hartmann  
glitch.me
```

SD Exploiting the unexploitable with lesser known browser tricks X +

← → C 🔒 https://speakerdeck.com/filedescriptor/exploiting-the-unexploitable-with-lesser-known-browser-tricks?slide=27 ☆ Incognito (5) 🌐 🔍

Exploiting the unexploitable with lesser known browser tricks

Cookie 💣+ Appcache = ?

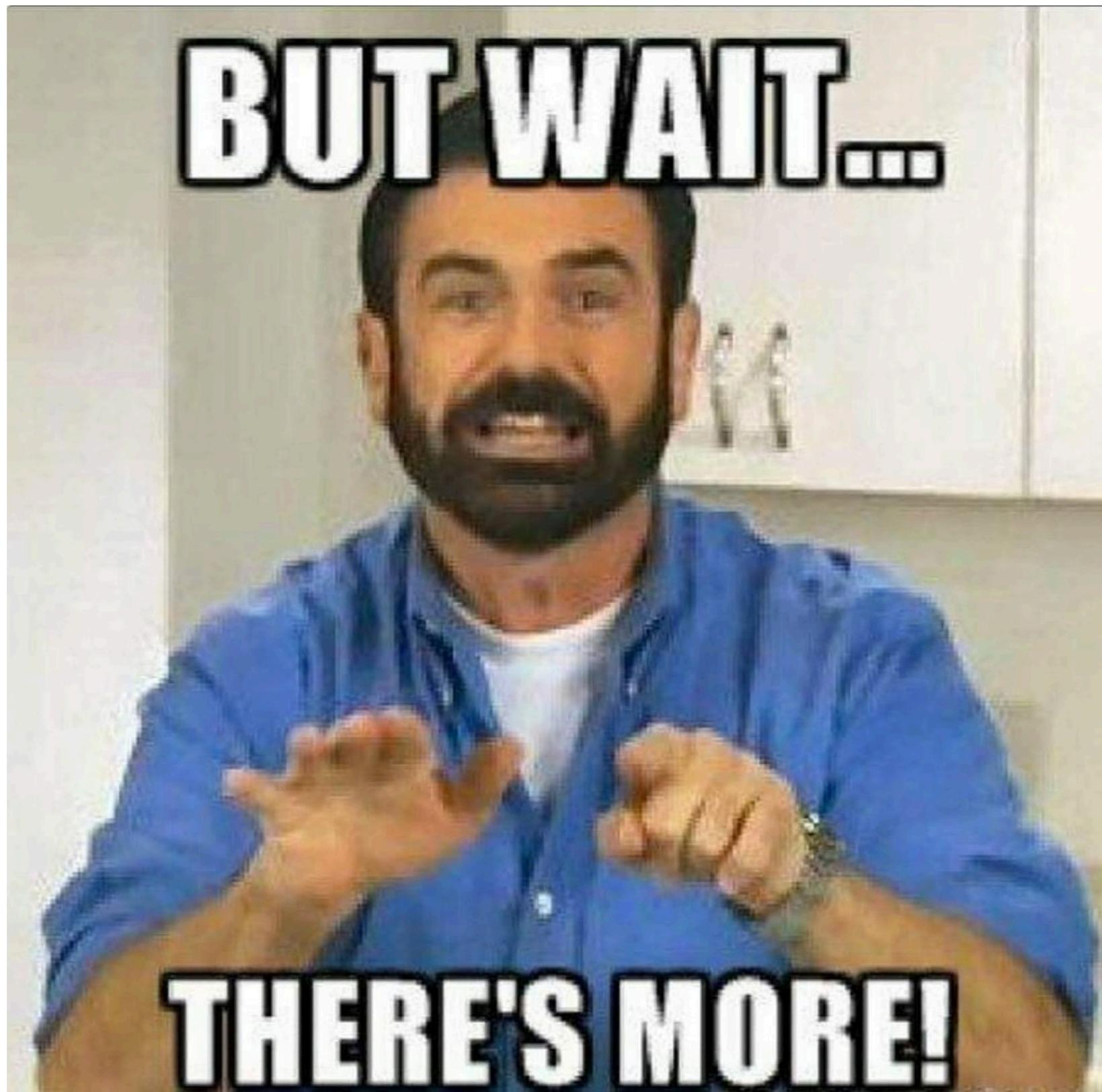
1. Set many cookies on root path
2. Requests to every file will result in HTTP 413
3. Appcache's fallback kicks in and replaces the response
4. ???
5. Profit!

filedescriptor May 11, 2017

Technology ☆ 16 ⚡ 16k ⬇

BUT WAIT...

THERE'S MORE!



XSS+OAuth

- Say you have a boring XSS
- And the site is using OAuth
- Sounds like you can use the XSS to takeover accounts?

Expectation

`https://google.com/oauth?client_id=example`



HTTP/1.1 302 Found
Location: `https://example.com/oauth/callback?code=123`
Set-Cookie: sid=123

↑
Steal



HTTP/1.1 302 Found
Location: `https://example.com/home`

Reality

https://google.com/oauth?client_id=example

HTTP/1.1 302 Found

Location: https://example.com/oauth/callback?code=123

Set-Cookie: sid=123

1. Authorization code is single-use

↑
Steal

2. Intermediate HTTP Redirect is transparent

```
> with(new XMLHttpRequest)open('get','https://httpbin.org/redirect-to?url=https://httpbin.org/redirect-to?url=/',false),send(),responseURL  
< "https://httpbin.org/"
```

HTTP/1.1 302 Found

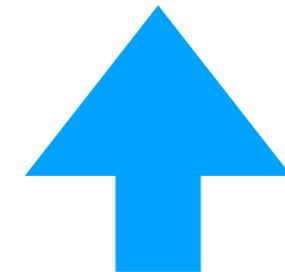
Location: https://example.com/home

XSS+🍪💣+OAuth

1. Perform Cookie Bomb Attack via XSS
2. Embed an iframe pointing to OAuth IdP
3. It redirects to target with the authorization code
4. Server rejects the request due to large header
5. Use XSS to get the authorization code from iframe URL



<https://example.com>



https://google.com/oauth?client_id=example



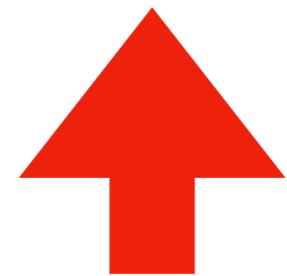
<https://example.com>



This page isn't working

If the problem continues, contact the site owner.

HTTP ERROR 431



iframe.contentWindow.location.href

<https://example.com/oauth/callback?code=123>



[REDACTED] rewarded [filedescriptor](#) with a \$250 bounty.

Jun 7th (2 months ago)

Thank you!



[REDACTED] rewarded [inhibitor181](#) with a \$250 bounty.

Jun 7th (2 months ago)

Thank you!



[inhibitor181](#) posted a comment.

Jun 7th (2 months ago)

This is an account takeover POC for [REDACTED] coming from a stored XSS. And after a successfull ATO the attacker can use the saved payment methods and create orders to arbitraty addresses of any price.



[REDACTED] changed the status to ● Triaged.

Jun 18th (2 months ago)



[tescoramen](#) HackerOne staff posted a comment.

Jun 20th (2 months ago)

Bounty amount currently being reviewed.



[REDACTED] rewarded [filedescriptor](#) with a \$2,250 bounty.

Jun 28th (2 months ago)

We are updating the bounty for this report. Thanks again @inhibitor181!



[REDACTED] rewarded [inhibitor181](#) with a \$2,250 bounty.

Jun 28th (2 months ago)

We are updating the bounty for this report. Thanks again @inhibitor181!

Path & HttpOnly

```
POST /admin HTTP/1.1
[...]
Cookie: csrf_token=foo; csrf_token=bar
```

This is a valid request

True or False?

HITCON ZeroDay

https://zeroday.hitcon.org/about/vp

漏洞 消息 排行榜 組織 奬勵計劃 註冊 or 登入

Elements Console Sources Network Performance Memory Application Security Audits

Manifest Service Workers Clear storage

XSRF-TOKEN: baz (Domain: .zeroda..., Path: /, Expires: N/A, Size: 13, HTTP: , Secure: , SameSite:)

XSRF-TOKEN: foo (Domain: .hitcon..., Path: /, Expires: N/A, Size: 13, HTTP: , Secure: , SameSite:)

XSRF-TOKEN: 123 (Domain: .zeroda..., Path: /about, Expires: N/A, Size: 13, HTTP: , Secure: , SameSite:)

Storage

Local Storage

XSRF-TOKEN: eyJpdil6ImtWVDNQSk5aaGNZNIFTWFwveCtMME... (Domain: zeroda..., Path: /, Expires: 2019-0..., Size: 288, HTTP: , Secure: checked, SameSite:)

XSRF-TOKEN: bar (Domain: zeroda..., Path: /about, Expires: N/A, Size: 13, HTTP: , Secure: , SameSite:)

Console What's New Network conditions Coverage

Default levels ▾

```
> document.cookie = 'XSRF-TOKEN=foo;domain=hitcon.org;path=/'
< "XSRF-TOKEN=foo;domain=hitcon.org;path=/"
> document.cookie = 'XSRF-TOKEN=bar;path=/about'
< "XSRF-TOKEN=bar;path=/about"
> document.cookie = 'XSRF-TOKEN=baz;domain=zeroday.hitcon.org;path=/'
< "XSRF-TOKEN=baz;domain=zeroday.hitcon.org;path=/"
> document.cookie = 'XSRF-TOKEN=123;domain=zeroday.hitcon.org;path=/about'
< "XSRF-TOKEN=123;domain=zeroday.hitcon.org;path=/about"
```

Cookie Tossing

- Cookie key consists of the tuple (name, domain, path)
- Each cookie-key-value has their own attribute list
- (Sub)domains can force a cookie with the same name to other (sub)domains
- Browser sends all cookies of the same name without attributes
- Server thus has no way to tell which one is from which domain/path

Yummy cookies across domain X +

https://github.blog/2013-04-09-yummy-cookies-across-domains/ Incognito

April 9, 2013 — Engineering, Featured

Yummy cookies across domains

 Vicent Martí

Last Friday we announced and performed a migration of all GitHub Pages to their own `github.io` domain. This was a long-planned migration, with the specific goal of mitigating phishing attacks and cross-domain cookie vulnerabilities arising from hosting custom user content in a subdomain of our main website.

There's been, however, some confusion regarding the implications and impact of these cross-domain cookie attacks. We hope this technical blog post will help clear things up.

Cookie tossing from a subdomain

When you log in on GitHub.com, we set a session cookie through the HTTP headers of the response. This cookie contains the session data that uniquely identifies you:

```
Set-Cookie: _session=THIS_IS_A_SESSION_TOKEN; path=/; expires=Sun, 01-Jan-2014 00:00:00 UTC
```

The session cookies that GitHub sends to web browsers are set on the default domain (`github.com`), which means they are not accessible from any subdomain at `*.github.com`. We also specify the `HttpOnly` attribute, which means they cannot be read through the `document.cookie` JavaScript API. Lastly, we specify the `Secure` attribute, which means the cookie is only sent over HTTPS.

GitHub Pages used to be on `*.github.com`

SD Killing 🐦 with 🐛 - Speaker X +

https://speakerdeck.com/filedescriptor/killing-with Incognito (2)

SD Killing 🐦 with 🐛

Killing 🐦 with 🐛

A journey from subdomain #SELF XSS to site-wide
#CSRF @Twitter

filedescriptor August 25, 2016

Technology 2 2.9k

Scenario

- Had an XSS on ton.twitter.com where contents are static
- twitter.com uses *auth_token* for session ID and *_twitter_sess* for storing CSRF token
- Could modify *_twitter_sess* with an attacker-known value and have site-wide CSRF
- However it's protected by HttpOnly

HttpOnly

- Cookies with this flag cannot be read/write from JavaScript API
- Safari **before version 12** has a bug that allows writing to HttpOnly cookies with JavaScript API
- Cookie Tossing can also help “bypass” this flag, as you can create a cookie with the same name but different key tuple

Expectation

Name	Value	Domain
_twitter_sess	original	
_twitter_sess	attacker's	.twitter.com



```
POST /i/tweet/create HTTP/1.1
```

```
[...]
```

```
Cookie: _twitter_sess=attackers; _twitter_sess=original
```

```
authenticity_token=attacker-known
```

Reality

Name	Value	Domain
_twitter_sess	original	
_twitter_sess	attacker's	.twitter.com



```
POST /i/tweet/create HTTP/1.1
```

```
[...]
```

```
Cookie: _twitter_sess=original; _twitter_sess=attackers;
```

```
authenticity_token=attacker-known
```

2. The user agent SHOULD sort the cookie-list in the following order:
 - * Cookies with longer paths are listed before cookies with shorter paths.
 - * Among cookies that have equal-length path fields, cookies with earlier creation-times are listed before cookies with later creation-times.

-RFC 6265 (5.4)

Precedence matters

- Specs do not mention how to handle duplicate cookies
- Most servers accept the first occurrence of cookies with the same name (think of HPP)
- Most browsers place cookies created earlier first

2. The user agent SHOULD sort the cookie-list in the following order:
 - * Cookies with longer paths are listed before cookies with shorter paths.
 - * Among cookies that have equal-length path fields, cookies with earlier creation-times are listed before cookies with later creation-times.

-RFC 6265 (5.4)

Revised Attack

Name	Value	Domain	Path
_twitter_sess	original		/
_twitter_sess	attacker's	.twitter.com	/i/



```
POST /i/tweet/create HTTP/1.1
```

```
[...]
```

```
Cookie: _twitter_sess=attackers; _twitter_sess=original
```

```
authenticity_token=attacker-known
```

Practical user agent implementations have limits on the number and size of cookies that they can store. General-use user agents SHOULD provide each of the following minimum capabilities:

- o At least 4096 bytes per cookie (as measured by the sum of the length of the cookie's name, value, and attributes).
- o At least **50 cookies per domain**.

-RFC 6265 (6.1)

Overflowing Cookie Jar

- Another way to “overwrite” a HttpOnly cookie is to remove it
- Browsers have a limitation on how many cookies a domain can have
- When there is no space, older cookies will get deleted
- Drawback: it’s not always easy to know how many cookies a victim has (tracking cookies are unpredictable)

More Cookie Tossing Application

The screenshot shows a report on the HackerOne platform. The report details a vulnerability titled "H1514 DOMXSS on Embedded SDK via Shopify.API.setWindowLocation abusing cookie Stuffing". The reporter is listed as "filedescriptor (filedescriptor)". Key statistics include a Reputation of 5655 (89th percentile), a Signal of 6.42 (95th percentile), and an Impact of 37.50 (99th percentile). The state is "Resolved (Closed)" and the severity is "High (8.1)". The bounty offered is \$5,000. The report was disclosed on April 17, 2019, at 10:40pm +0800. It was reported to Shopify. The weakness is described as "Cross-site Scripting (XSS) - DOM". The summary by Shopify states: "During H1-514, @filedescriptor reported an XSS issue in our Embedded App SDK that allowed for attacking legitimate apps through our platform, due to a missing protocol check on the Shopify.API.setWindowLocation. Since this issue would have allowed realistic attacks against apps using the Embedded App SDK, we decided to award \$2500 for this issue. As part of the event, @filedescriptor was also awarded a bonus of \$2500 for chaining multiple bugs." The timeline shows the reporter submitting the report to Shopify on Oct 10th (10 months ago) and providing a detailed description of the exploit.

Self-XSS to full XSS

Selectively forcing attacker's session cookie on certain paths



The screenshot shows a HackerOne report page for a bug titled "H1514 Session Fixation on multiple shopify-built apps on *.shopifycloud.com and *.shopifyapps.com". The report was submitted by user "filedescriptor" (#423136) and is currently "Resolved (Closed)". It was disclosed on April 25, 2019, at 10:39am +0800. The bug was reported to Shopify and identified as a "Session Fixation" weakness with a bounty of \$5,000. The severity is listed as "No Rating (---)". The summary notes that the report was submitted during a live hacking event, which had an expanded scope compared to the public bug bounty program. The timeline shows the reporter's initial message and a follow-up message detailing the issue.

#423136 H1514 Session Fixation

HackerOne, Inc. [US] | https://hackerone.com/reports/423136

SIGN IN | SIGN UP

Incognito (2)

filedescriptor (filedescriptor)

FOR BUSINESS FOR HACKERS HACKTIVITY COMPANY TRY HACKERONE

128 #423136 H1514 Session Fixation on multiple shopify-built apps on *.shopifycloud.com and *.shopifyapps.com

Share: [f](#) [t](#) [g+](#) [in](#) [Y](#) [e](#)

State: Resolved (Closed)

Severity: No Rating (---)

Disclosed: April 25, 2019 10:39am +0800

Participants:

Reported To: Shopify

Visibility: Disclosed (Full)

Weakness: Session Fixation

Bounty: \$5,000

Collapse

SUMMARY BY SHOPIFY

Note: This report was submitted during our [H1-514 live hacking event](#), which had an expanded scope compared to our public bug bounty program. Some of the apps mentioned in this report are not currently in scope for our public program.

@filedescriptor noticed that several add-on applications built by Shopify were vulnerable to session fixation because they did not generate a fresh session cookie during the OAuth2 callback phase. This could have allowed an attacker to gain access to a victim's app session if certain conditions were met.

TIMELINE

filedescriptor submitted a report to Shopify. Oct 13th (10 months ago)
Hi team!,
I'm reporting a Session Fixation issue on multiple shopify-built apps hosted on *.shopifycloud.com and *.shopifyapps.com. Normally

Session Fixation

Forcing attacker's session cookie with a subdomain
XSS

https://script-editor.shopifycloud.com

Force a session cookie scoped to .shopifycloud.com using XSS

document.cookie='__flow_session=attackers;domain=.shopifycloud.com'

https://victim.myshopify.com/admin/oauth/authorize?client_id=flow

OAuth redirect with authorization code

GET /oauth/callback?code=victims HTTP/1.1

Host: flow.shopifycloud.com

Cookie: __flow_session=attackers

Implementation Discrepancy

Multiple Cookies at Once?

- We can only set one cookie at a time in a single Set-Cookie header
- However, the older specs allow setting multiple in a single Set-Cookie header

The screenshot shows a web browser window with the title bar "XSS by tossing cookies - WeSecureApp". The URL in the address bar is <https://wesecureapp.com/2017/07/10/xss-by-tossing-cookies/>. The page content includes the WeSecureApp logo, a navigation bar with links like Services, Solutions, Vertical, Blog, Careers, About, and Strobes, and a main article titled "XSS by tossing cookies" dated July 10, 2017. The article discusses various XSS vulnerabilities, including one involving Outlook and Twitter, and ends with a section on Outlook Client Side Stored Cross Site Scripting Vulnerability.

XSS by tossing cookies

JULY 10, 2017 | BY ESHWARGETENV@WSA | UNCATEGORIZED

All cross site scripting vulnerabilities cannot be exploited easily and would need a vulnerability chain to exploit them

For example a self XSS that only executes in your profile, here is how whitton used minor OAuth flaws to exploit a cross site scripting in Uber
<https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

How about a XSS that needs a lot of user interaction?

This is how Sasi used a clicking vulnerability to successfully exploit a XSS in Google
<http://sasi2103.blogspot.in/2016/09/combination-of-techniques-lead-to-dom.html>

What about a Cross site scripting that needs an arbitrary cookie?

Here is how we found cross site scripting vulnerabilities in Outlook and Twitter by tossing cookies in Safari browser.

Outlook Client Side Stored Cross Site Scripting Vulnerability

There was a simple cross site scripting on outlook.live.com, a value from cookie was directly reflected back in the source without any filtering.

Request

Privacy & Cookies Policy

Cookie based XSS

Exploiting limited Cookie Injection with Safari

“Informally, the Set-Cookie response header comprises the token Set-Cookie:, followed by a comma-separated list of one or more cookies.”

-RFC 2109 (4.2.2)

```
Set-Cookie: foo=123; path=/admin; HttpOnly; bar=456; Secure
```

Works in Safari before version 10

```
GET /admin HTTP/1.1  
[ ... ]  
Cookie: foo=123; bar=456
```

https://outlook.live.com/owa/?realm=hotmail.com;, ClientId='-alert(2)-'



HTTP/1.1 200 OK

[...]

Set-Cookie: realm=hotmail.com; , ClientId='-alert(2)-'



GET / HTTP/1.1

[...]

Cookie: realm=hotmail.com; ClientId='-alert(2)-'

Safari sets 2 cookies



window.clientId = '-alert(2)-';

The screenshot shows a report on the HackerOne platform. The report details a CSRF protection bypass found on mobile.twitter.com. The reporter is Sergey Bobrov (bobrov), who has a reputation of 5910 and is ranked 86th. The signal percentile is 6.34, and the impact percentile is 15.97. The report was disclosed on May 5, 2015, at 12:09am +0800. It was reported to Twitter and identified as a Cross-Site Request Forgery (CSRF). The state is Resolved (Closed). The severity is No Rating (---). Participants include bobrov, Twitter, and Google. The visibility is Disclosed (Full). The timeline shows bobrov submitting the report and explaining the steps to create a PoC, followed by a detailed technical explanation of cookie injection via Google Analytics and its parsing peculiarities.

Sergey Bobrov (bobrov)

FOR BUSINESS FOR HACKERS HACKTIVITY COMPANY TRY HACKERONE

1 #14883 [mobile.twitter.com / twitter.com] CSRF protection bypass

State: Resolved (Closed) Severity: No Rating (---)
Disclosed: May 5, 2015 12:09am +0800 Participants: bobrov, Twitter, Google
Reported To: Twitter Visibility: Disclosed (Full)
Weakness: Cross-Site Request Forgery (CSRF)

TIMELINE

bobrov submitted a report to Twitter.
I shall explain all the steps to create the final PoC in order to be more clear. Jun 3rd (5 years ago)

Part 1. Cookie Injection via Google Analytics

1) Google Analytics sets the cookie to track user source:
123456.123456789.11.2.utmcsrc=[HOST]|utmccn=(referral)|utmcmd=referral|utmccct=[PATH]
For example:
123456.123456789.11.2.utmcsrc=blackfan.ru|utmccn=(referral)|utmcmd=referral|utmccct=/path/
2) User fully controls path in Referer and it is not filtered before being put in __utmz

Part 2. Cookie parsing peculiarities by different web servers

1) A typical Cookie sent by a web browser looks like this:

CSRF Cookie Injection

Server accepting comma separated cookies

“For backward compatibility, the separator in the Cookie header is semi-colon (;) everywhere. A server **SHOULD** also accept comma (,) as the separator between cookie-values for future compatibility.”

-RFC 2965 (3.3.4)

http://blackfan.ru/r/,m5_csrf_tkn=x;domain=.twitter.com;path=/

Cookie set by Google Analytics on translation.twitter.com scoped to .twitter.com

__utmz=123456.123456789.11.2.utmcst=blackfan.ru|utmccn=(referral)|utmccct=/r/,m5_csrf_tkn=x

POST /messages/follow HTTP/1.1

[. . .]

Twitter's server parses it as 2 cookies

Cookie: __utmz=123456.123456789.11.2.utmcst=blackfan.ru|utmccn=(referral)|utmccct=/r/,m5_csrf_tkn=x

m5_csrf_tkn=x

Defense

Cookie Prefixes

- Cookies prefixed with `_Host-` cannot have Domain attribute
- This prevents (sub)domains from forcing a cookie the current domain doesn't want
- Cookies intended for (sub)domains are still vulnerable to Cookie Tossing
- Use a separate domain for user generated assets

```
> document.cookie = '__Host-foo=bar; domain=example.com'  
< " __Host-foo=bar; domain=example.com"  
> document.cookie  
< ""
```

**Servers must only
follow RFC 6265**

Intent to Implement and Ship: Cookies with SameSite by default

https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/AknSSyQTGYs/gf1ReqJdBAAJ

Groups

Search for messages

Sign in

Groups

Home

Click on a group's star icon to add it to your favorites

Recently viewed

blink-dev

Privacy - Terms of Service

1 of 99+ < > ⚙️ 🔍

blink-dev >
Intent to Implement and Ship: Cookies with SameSite by default
29 posts by 13 authors

Lily Chen May 25

Other recipients: morl...@chromium.org, mk...@chromium.org

Contact emails chl...@chromium.org

Explainer <https://tools.ietf.org>
<https://web.dev/samesite-cookies-explained>

Design doc/Spec
See [spec changes proposed in explainer](#).
[TAG review request](#).

Summary
Treat cookies as SameSite=Lax by default if no SameSite attribute is specified. Developers would be able to opt-into the status quo of unrestricted use by explicitly asserting SameSite=None.

Motivation
"SameSite" is a reasonably robust defense against some classes of cross-site request forgery (CSRF) attacks, but developers currently need to opt-into its protections by specifying a SameSite attribute, i.e., developers are vulnerable to CSRF attacks by default. This change would allow developers to be protected by default, while allowing sites that require state in cross-site requests to opt-in to the status quo's less-secure model. In addition, forcing sites to opt-in to SameSite=None gives the user agent the ability to provide users more transparency and control over tracking.

Risks

Interoperability and Compatibility
Some sites relying on third-party cookies may break temporarily until developers add "SameSite=None". Most browsers (except for Safari; bug filed [here](#)) that do not yet support SameSite=None should just ignore that attribute and apply the status quo default of "no restriction", which has the same effect as SameSite=None.

Edge: Public support
Firefox: [In development](#)

PSA: CSRF & others will be dead in 2020 😢

Q&A

find me on Twitter @filedescriptor

