101

DEFENCE,
CHANGE MY MIND!

ZERO NIGHTS 2018

Egor Karbutov @ShikariSenpai
Sergey Belov @SergeyBelove

for
WebVillage

Who are we?

Yandex & Mail.Ru appsec teams

Agenda

- XSS Contexts
- How to generate CSRF-token
- SSRF
- Impossible to patch
- Let’s play!

XSS Contexts

Escaping vs Sanitizing vs Filtering

- Escaping

```
HTML
&#x20AC;  hexadecimal numeric character reference
&#8364; decimal numeric character reference
&euro;  named character reference

CSS
\20AC   must be followed by a space if the next
character is one of a-f, A-F, 0-9
\0020AC must be 6 digits long, no space needed (but
can be included)
```

Escaping vs Sanitizing vs Filtering

- Sanitizing

```
Hello, <b>test</b><script>alert(1)</script>

to

Hello, <b>test</b>
```

Escaping vs Sanitizing vs Filtering

- Filtering

```
<a href=”javascript:alert(1)”>test</a>

to

<a></a>
```

101

HTML Sanitizer DOM Purify

Dirty HTML

```
><script>alert(1)></script>
<a href="javascript:alert(1);">xss</a>
<a href="https://google.com">google</a>
```

Clean HTML

```
&gt;
<a>xss</a>
<a href="https://google.com">google</a>
```

Where?

- Two options
  - Before saving user's data to database
  - During the rendering
- Template engines
  - During the rendering

- For Django {{|safe}} will lead to XSS

```
<div id="get_title">{{ message.title|safe }}</div>
```

- Client Side validation isn't best way

Escaping special chars

- To mitigate most of the problem with XSS
  - ><&”’
  - < -> <
  - > -> >
  - & -> &
  - " -> "
  - ' -> ' / '
- But what about XSS contexts?

2³ EDITION

101

XSS Contexts

- Don't forget about it
- Super-uber blind vector
- In real life it might not work

Contexts game:
http://polyglot.innerht.ml/

```
javascript:"/*'/*`/*--></noscript></title></textarea>
</style></template></noembed></script><html
\" onmouseover=/*&lt;svg/*/onload=alert()//>
```

```
<div class="{{payload}}"></div>
<div class='{{payload}}'></div>
<title>{{payload}}</title>
<textarea>{{payload}}</textarea>
<style>{{payload}}</style>
<noscript>{{payload}}</noscript>
<noembed>{{payload}}</noembed>
<template>{{payload}}</template>
<frameset>{{payload}}</frameset>
<select><option>{{payload}}</option></select>
<script type="text/template">{{payload}}</script>
<!--{{payload}}-->
<iframe src="{{payload}}"></iframe> " →
<iframe srcdoc="{{payload}}"></iframe> " → < →
<script>"{{payload}}"</script> </script → <\/script
<script>'{{payload}}'</script> </script → <\/script
<script>`{{payload}}`</script> </script → <\/script
<script>//{{payload}}</script> </script → <\/script
<script>/*{{payload}}*/</script> </script → <\/script
<script>"{{payload}}"</script> </script → <\/script " → \"
```

XSS Contexts

```
<b>{{user_input}}</b>
```

- Dangerous special chars are ><

```
<input type="text" name="xss" value="{{user_input}}" />
```

- Dangerous special chars are "

```
<input type='text' name='xss' value='{{user_input}}' />
```

- Dangerous special char is '

XSS Contexts

```
<input type=text name=some value={{user_input}} />
```

- Don't ever do that!

```
<a href="{{user_input}}">cats</a>
```

- Dangerous special chars are " and browser scheme

```
<a href="javascript:alert(1);">cats</a>
```

- Scheme whitelist:
  - mailto:
  - https:
  - http:

XSS Contexts

`<script>{{user_input}}</script>` - difficult case

- Dangerous special chars are >< and ‘” for JSON/var escape

- Don’t forget about DOM XSS
  - Do not allow a user to control parameters for eval functions

EDITION

Another attacks to break SOP

`<object>/<iframe>/<embed>`

`<style>` + CSS

- Do not allow a user to control these tags

- window.opener
- CSS leaks
- “perfect pixel”
- timing attacks

ZERO NIGHTS 2018

2^3 EDITION

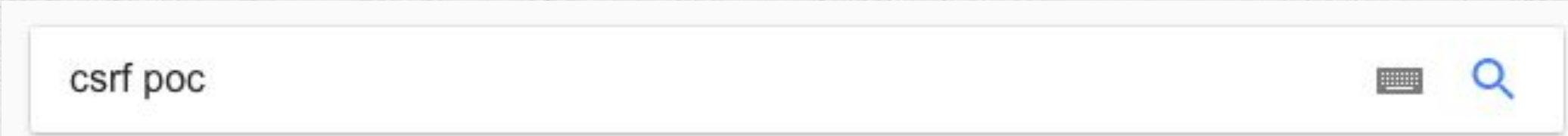
101

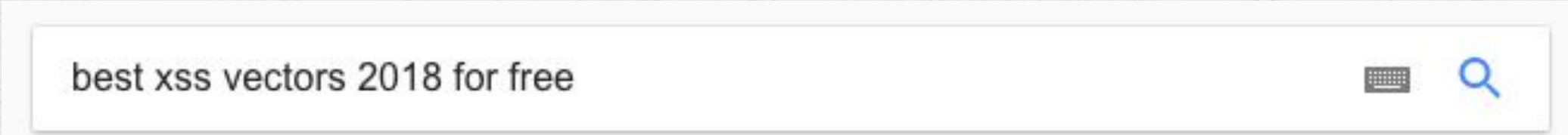
How to generate CSRF-token

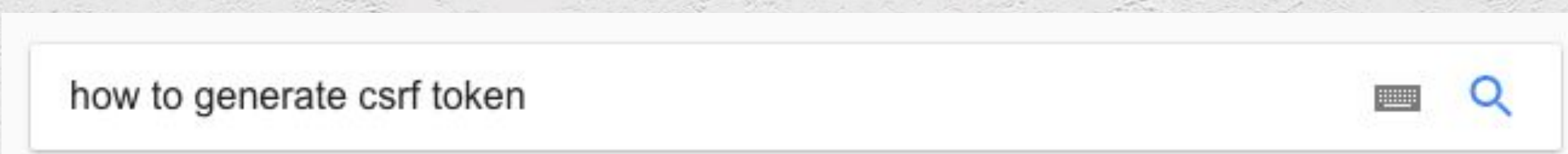
Stateless/ful

- Stateful - easiest
  - Random token
  - A part of session
  - Depends on actions
- Stateless
  - JWT
  - Cookie based (cookie injection problem)
  - and more

Defence dilemma







Defence dilemma

how to generate csrf token

```
$time = time();
$token = md5(SECRET_KEY . $userData . $time) . ':' . $time;
```

Try `base64_encode(openssl_random_pseudo_bytes(16))` . https://github.com/codeguy/php-the-right-way/issues/272#issuecomment-18688498 and I used it for my form example in https://gist.github.com/mikaelz/5668195

This answer is useful

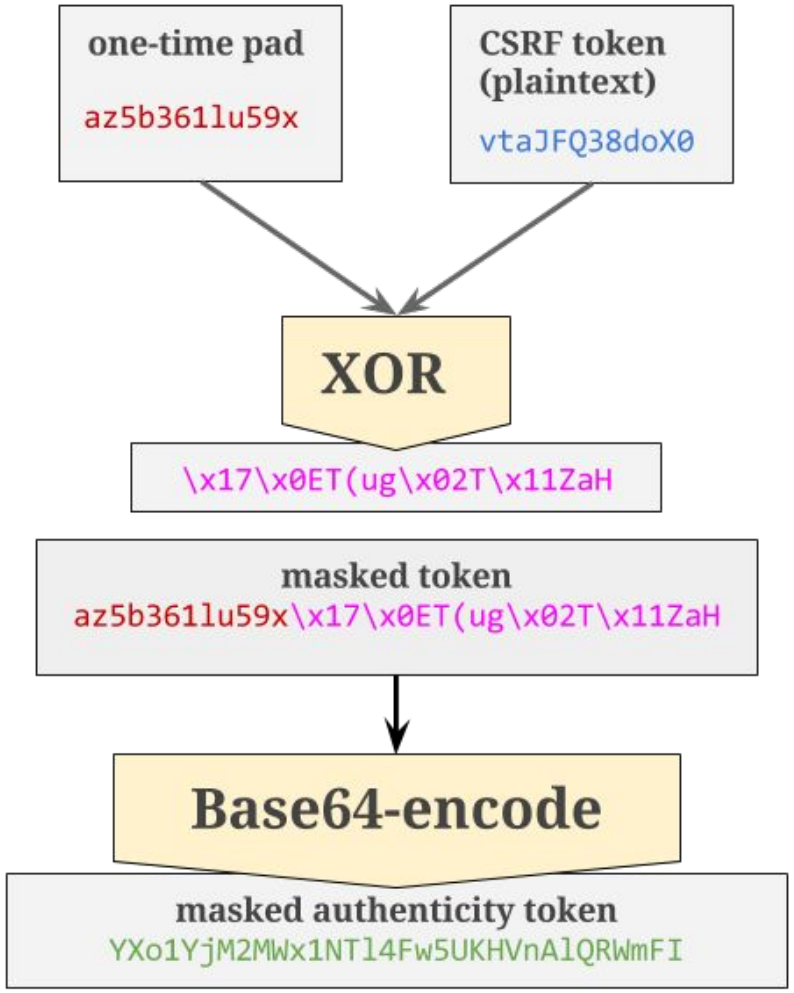
Our CSRF-Token Scheme

```
HMAC_SHA1/256/512(secret_key,"cookie_value:timestamp:action")
```

- Integrity control
- Depending on the time
- Depending on the action
- Secret_key for different application
- Something else?

EDITION 2 3

Ruby on Rails CSRF




```
one_time_pad = SecureRandom.random_bytes(AUTHENTICITY_TOKEN_LENGTH)
encrypted_csrf_token = xor_byte_strings(one_time_pad, raw_token)
masked_token = one_time_pad + encrypted_csrf_token
Base64.strict_encode64(masked_token)
```

2^3 EDITION

Escaping special chars

Lifetimes of popular cryptographic hashes (the rainbow chart)

Function	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
Snefru																												
MD2 (128-bit)[1]																												
MD4																												
MD5															[2]													
RIPEMD															[2]													
HAVAL-128[1]															[2]													
SHA-0																												
SHA-1																												[3]
RIPEMD-160																												
SHA-2 family																		[4]										
SHA-3 (Keccak)																												

Key Didn't exist/not public | Under peer review | Considered strong | Minor weakness | Weakened | Broken | Collision found

Hash table:

http://valerieaurora.org/hash.html

https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

Our CSRF-Token Scheme

```
HMAC_SHA1/256/512(secret_key,"cookie_value:timestamp:action")
```

- HMAC mitigate
  - length extension attack
  - hash collisions*
- Danger:
  - HMAC (user_data, secret_key) – is wrong order leads to simple collision
  - If len(K) > block size: K:=H(K)
  - I can signature message with my user_data - H(user_data)

*https://dankaminsky.com/2015/05/07/the-little-mac-attack/

CSRF-Token

- How to send a CSRF-token?
  - GET parameter
    - Bad options
    - Violation of RFC7231 about GET requests
    - Don’t forget about server logs
    - Referrer leaks your token
  - POST parameter
  - Header
    - For JS Requests
    - Double Submit Cookie Problem with subdomains
- Same-Site Cookie

How to develop good web application:
https://habr.com/company/yandex/blog/265569/

SSRF Problem

Usual mitigation

- I want to download my cats pic from

```
https://cats.mydomain:443/pic?a=1234
```

- SSRF via domain/IPv4 address

```
https://127.0.0.1:443/
```

- SSRF via port

```
127.0.0.1:8080/
```

- SSRF via scheme

```
file://127.0.0.1/
```

Something more

- SSRF via different domain format address

```
https://2130706433:443/
```

- SSRF via IPv6 address

```
https://[::]:443/
```

```
https://[0000::1]:443/
```

- SSRF via different encoding(enclosed alphanumerics and URL encode)

```
https://ⓔⓧⓐⓜⓟⓛⓔ.ⓒⓞⓜ
```

```
https://%65%78%61%6d%70%6c%65%2e%63%6f%6d
```

Usual mitigation

- SSRF via parsing tricks

```
https://1.1.1.1 &@2.2.2.2# @3.3.3.3/
```

```
urllib2 : 1.1.1.1
requests + browsers : 2.2.2.2
urllib : 3.3.3.3
```

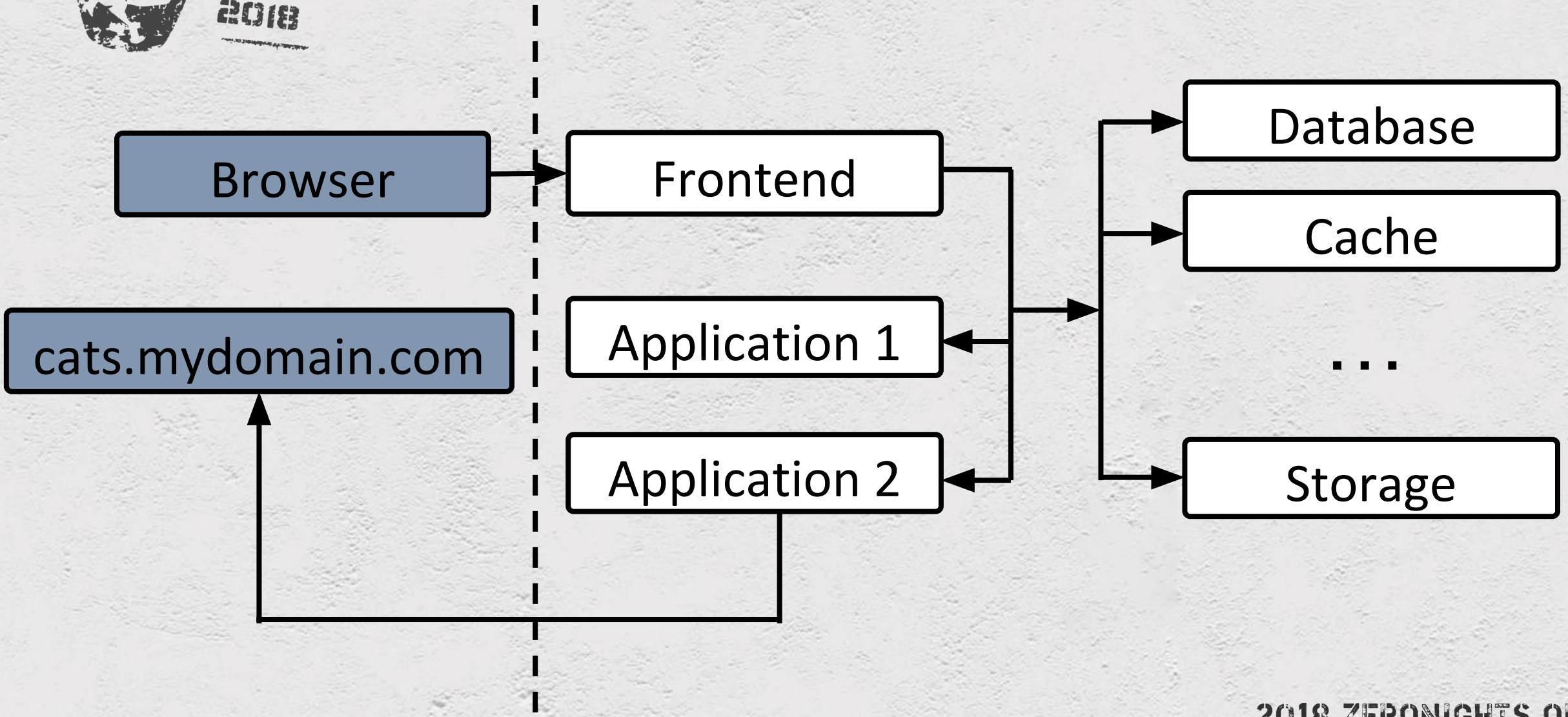
- SSRF DNS A record + sometimes race condition

```
127.0.0.1 with DNS A record:ssrf.mydomain.com
```

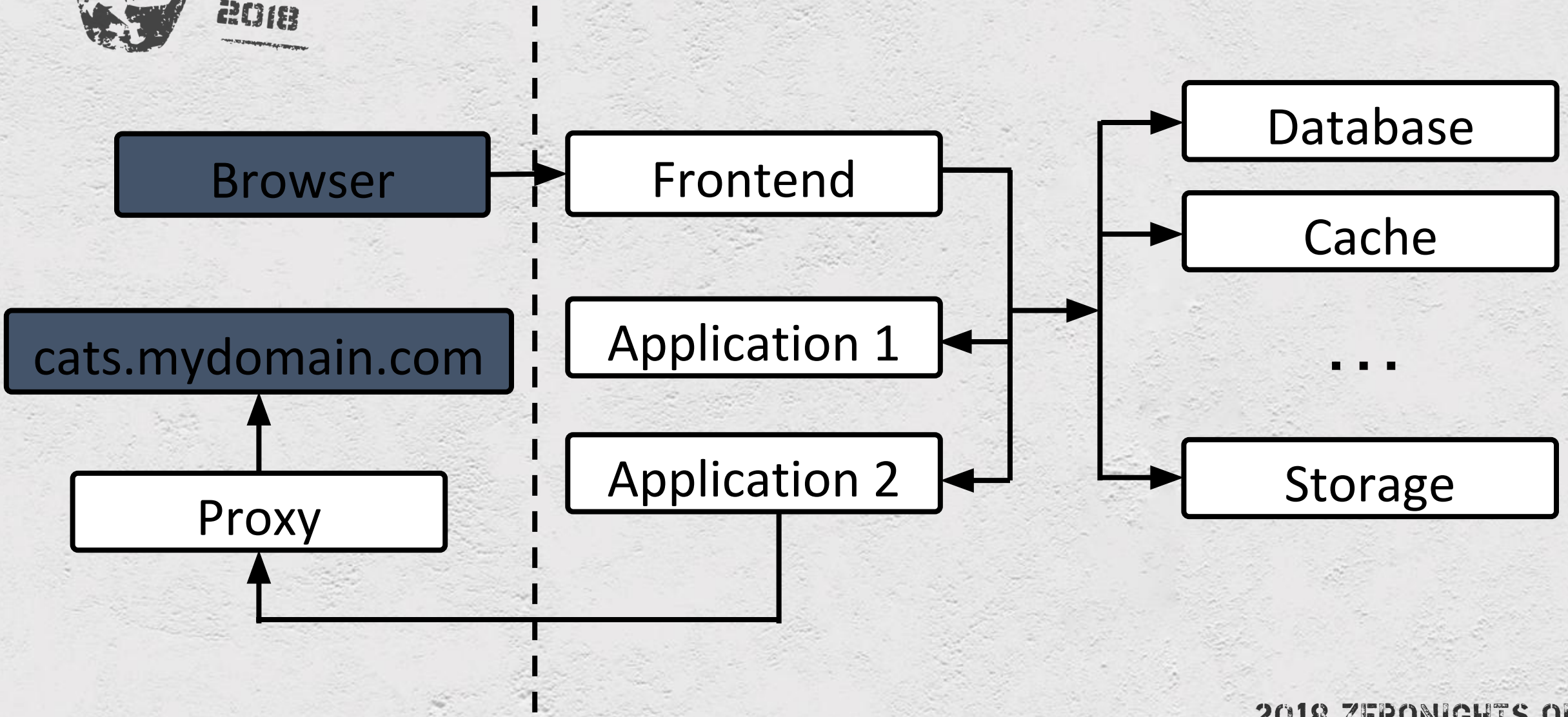
- SSRF via redirects

```
ssrf.mydomain.com 3xx redirect to 127.0.0.1
```

SSRF Scheme


Browser
cats.mydomain.com
Frontend
Application 1
Application 2
Database
Cache
...
Storage

SSRF Proxy


Browser
cats.mydomain.com
Proxy
Frontend
Application 1
Application 2
Database
Cache
...
Storage

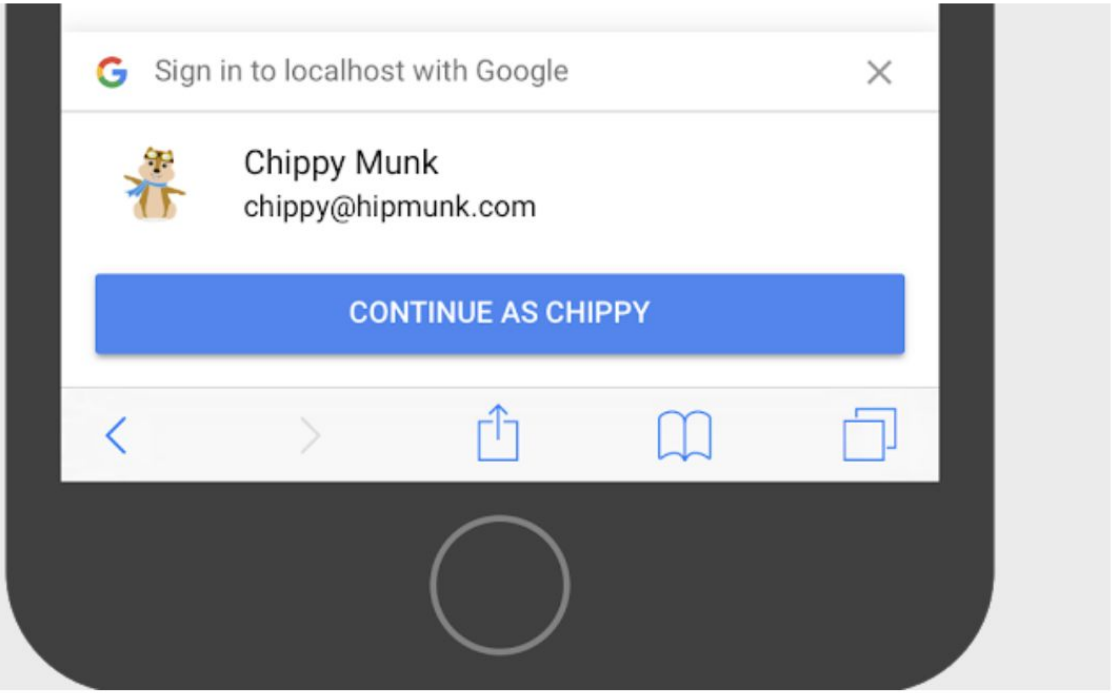
SSRF Proxy

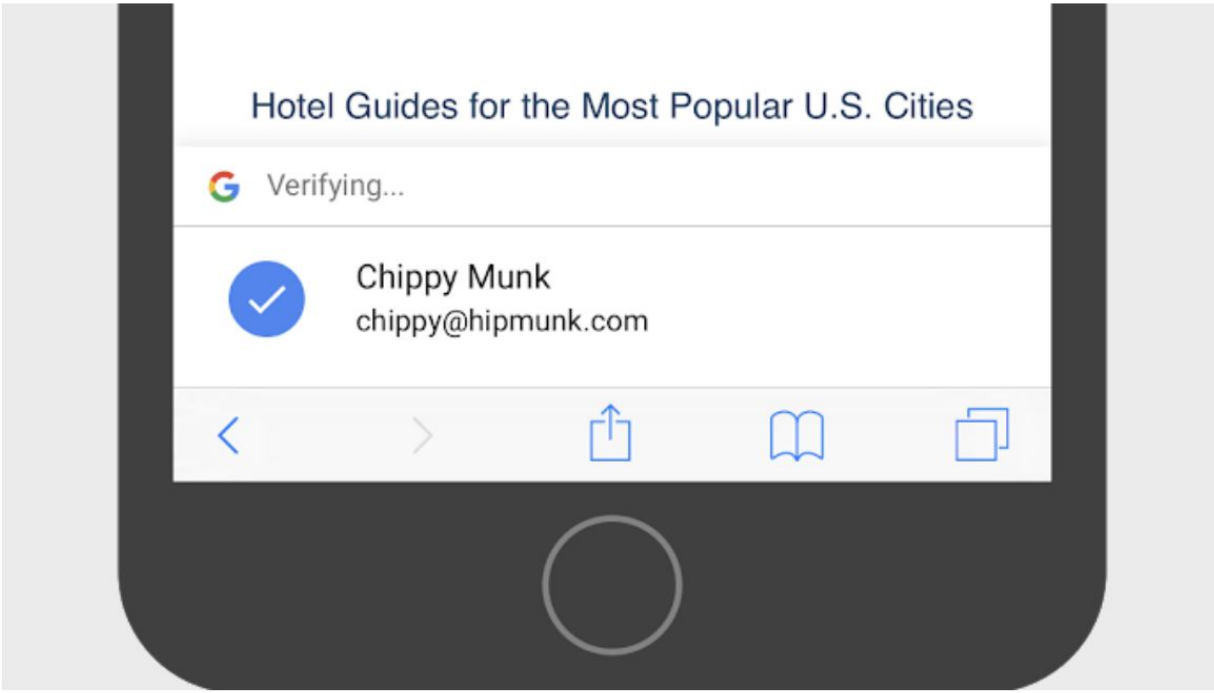
- Don't forget
  - About usual mitigation
  - Extra hardening
- Proxy in docker container make bonus security
- Issues that still hard to restrict in case of RCE:
  - Access to repository
  - Docker hub
  - Monitoring
  - Logs
- Use orchestration for mitigation

Impossible to patch

Impossible to patch

OAuth via iFrame without consent screen - WTF?





https://blog.innerht.ml/google-yolo/

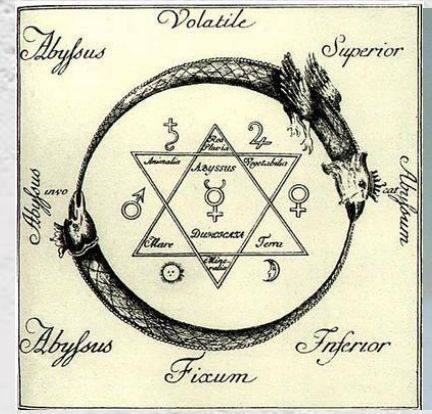
Let's Play

Useful Links

- Contexts game:
  - http://polyglot.innerht.ml/
- XSS contexts payloads:
  - https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/XSS-WITH-CONTEXT-JHADDIX.txt
- Hash Table:
  - http://valerieaurora.org/hash.html
- Best practice for web application:
  - https://habr.com/company/yandex/blog/265569/
- Ruby CSRF Protect:
  - https://medium.com/rubyinside/a-deep-dive-into-csrf-protection-in-rails-19fa0a42c0ef
- Post about CSRF:
  - https://habr.com/post/318748/

THANKS FOR ATTENTION

Egor Karbutov
@ShikariSenpai



Sergey Belov
@SergeyBelove