black hat USA 2023

AUGUST 9-10, 2023
BRIEFINGS

Cookie Crumbles:
Unveiling Web Session Integrity Vulnerabilities

Marco Squarcina
TU Wien
@blueminimal
https://infosec.exchange/@minimalblue
marco.squarcina@tuwien.ac.at

Pedro Adão
IST, Universidade de Lisboa
@pedromigueladao
https://infosec.exchange/@pedroadao
pedro.adao@tecnico.ulisboa.pt

Joint work with **Lorenzo Veronese** and **Matteo Maffei**

Who Are We

- **PhD** @ Ca' Foscari, Venice, IT 🇮🇹
- **Senior Scientist** @ TU Wien, Vienna, AT 🇦🇹
- **Web** & **Mobile** (in)**Security**
- **CTF player / organizer** since 2009
- Founder of **mhackeroni** 🍝
  (5x **DEF CON CTF** finalist)
  Playing with **WE_OWN_YOU** 🇦🇹
- IT security education projects with
  ENISA 🇪🇺, **CSA**, formerly **Cyberchallenge.IT**
- https://minimalblue.com/

Marco Squarcina

Who Are We

Pedro Adão

- **PhD** @ Técnico-Lisboa, PT 🇵🇹
- **Associate Prof.** @ Técnico-Lisboa, PT 🇵🇹
- **Programming Lang** & **Web** (in)**Security**
- **CTF player** since 2013
- Founder of **STT** and **CyberSecurity ChallengePT**
- **Coach Team PT** 🇵🇹 (ECSC 2019-...)
- **Coach Team Europe** 🇪🇺 (ICC 2022, 2023)

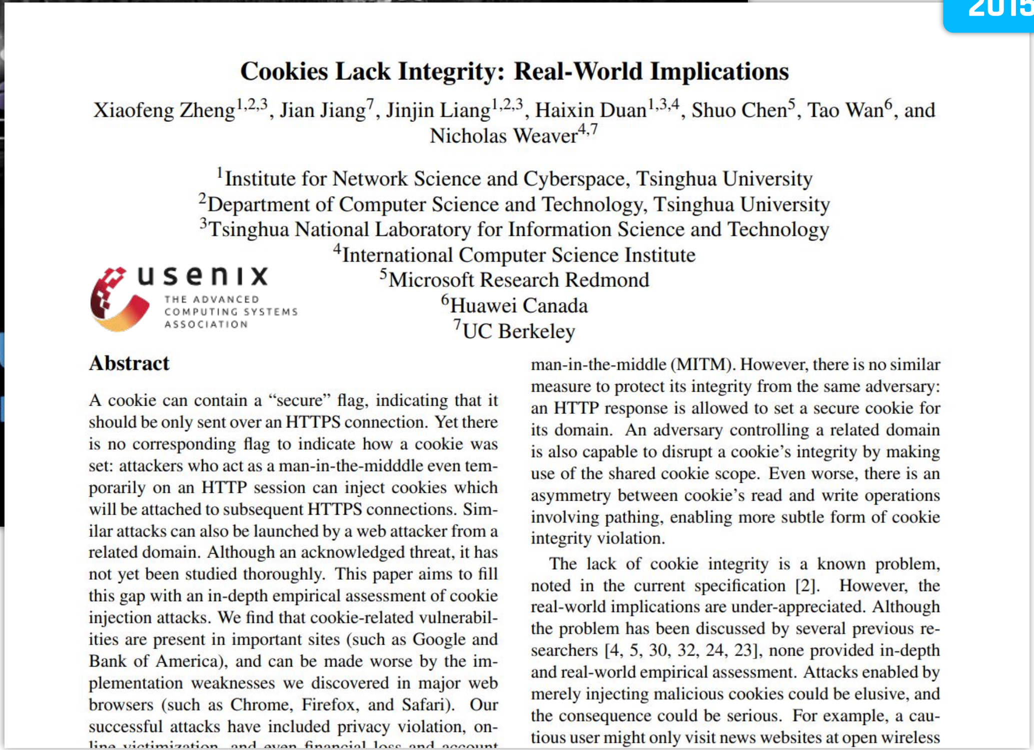
Have Weak Integrity

2013


black hat
EU 2013
THE DEPUTIES ARE STILL CONFUSED
RICH LUNDEEN

2013

2015



Cookies Lack Integrity: Real-World Implications

Xiaofeng Zheng[1,2,3], Jian Jiang[7], Jinjin Liang[1,2,3], Haixin Duan[1,3,4], Shuo Chen[5], Tao Wan[6], and Nicholas Weaver[4,7]

[1]Institute for Network Science and Cyberspace, Tsinghua University
[2]Department of Computer Science and Technology, Tsinghua University
[3]Tsinghua National Laboratory for Information Science and Technology
[4]International Computer Science Institute
[5]Microsoft Research Redmond
[6]Huawei Canada
[7]UC Berkeley

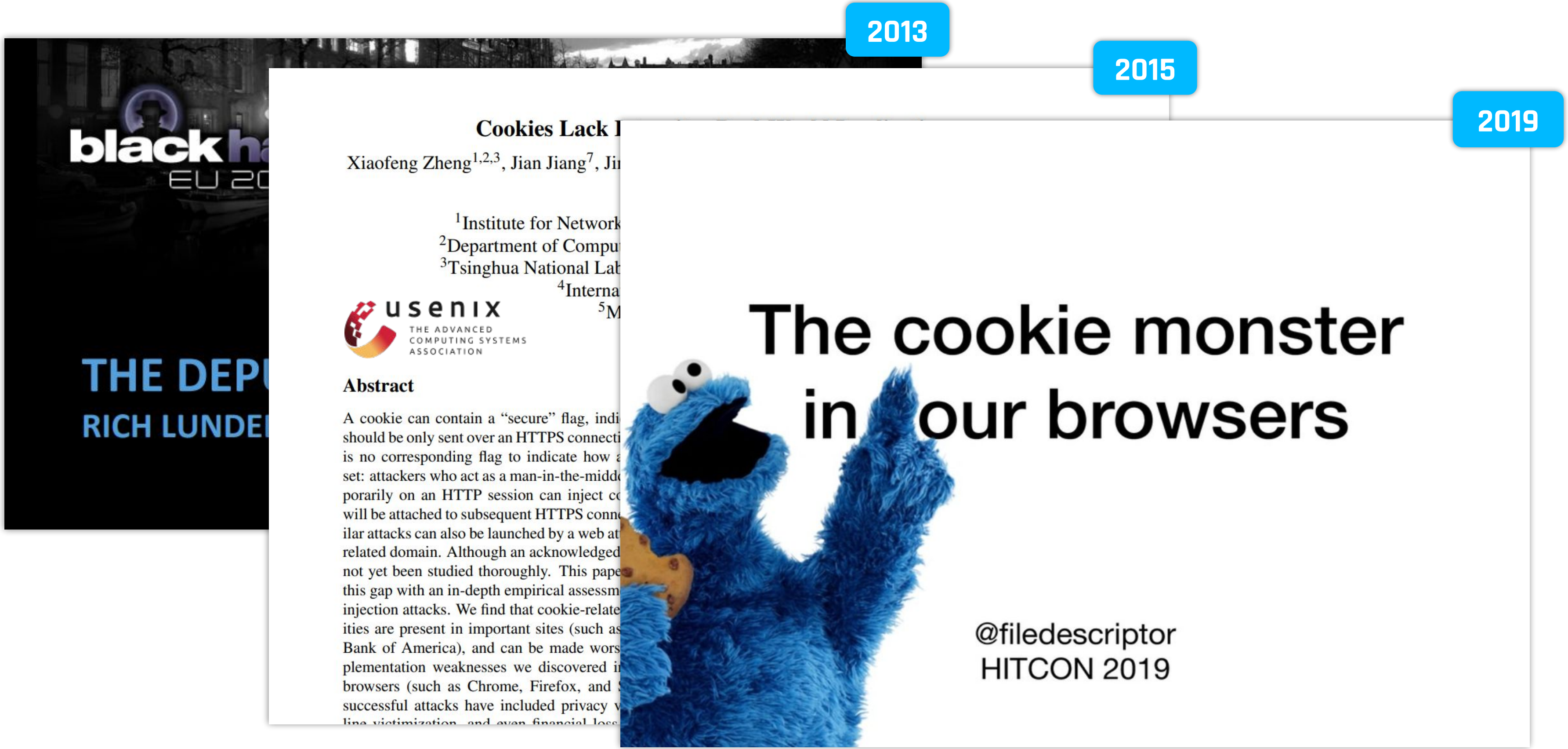
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

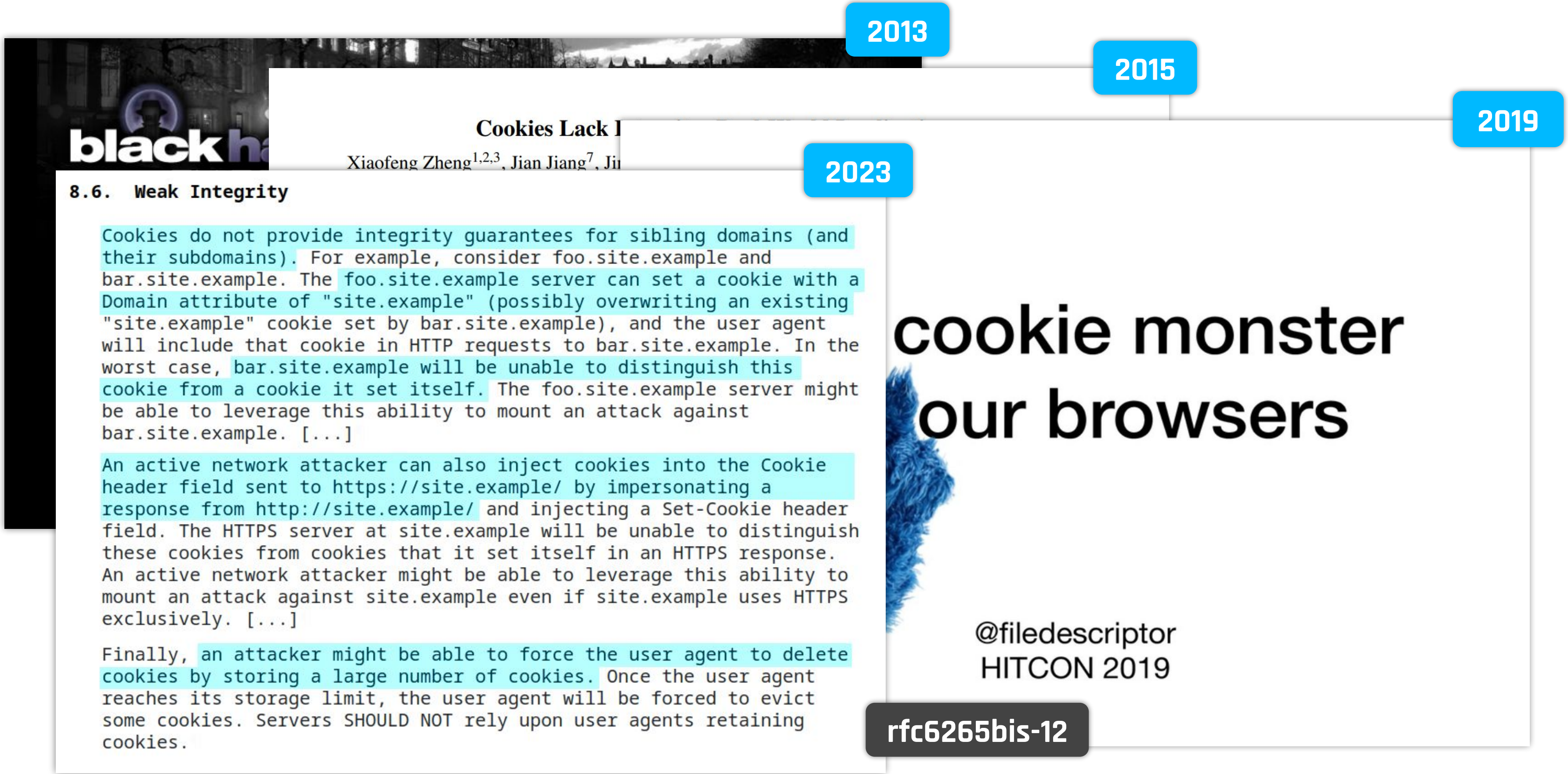
Abstract

A cookie can contain a "secure" flag, indicating that it should be only sent over an HTTPS connection. Yet there is no corresponding flag to indicate how a cookie was set: attackers who act as a man-in-the-midddle even temporarily on an HTTP session can inject cookies which will be attached to subsequent HTTPS connections. Similar attacks can also be launched by a web attacker from a related domain. Although an acknowledged threat, it has not yet been studied thoroughly. This paper aims to fill this gap with an in-depth empirical assessment of cookie injection attacks. We find that cookie-related vulnerabilities are present in important sites (such as Google and Bank of America), and can be made worse by the implementation weaknesses we discovered in major web browsers (such as Chrome, Firefox, and Safari). Our successful attacks have included privacy violation, online victimization, and even financial loss and account

man-in-the-middle (MITM). However, there is no similar measure to protect its integrity from the same adversary: an HTTP response is allowed to set a secure cookie for its domain. An adversary controlling a related domain is also capable to disrupt a cookie's integrity by making use of the shared cookie scope. Even worse, there is an asymmetry between cookie's read and write operations involving pathing, enabling more subtle form of cookie integrity violation.

The lack of cookie integrity is a known problem, noted in the current specification [2]. However, the real-world implications are under-appreciated. Although the problem has been discussed by several previous researchers [4, 5, 30, 32, 24, 23], none provided in-depth and real-world empirical assessment. Attacks enabled by merely injecting malicious cookies could be elusive, and the consequence could be serious. For example, a cautious user might only visit news websites at open wireless


2013
2015
2019
black ha
EU 20
THE DEPU
RICH LUNDE
Cookies Lack I
Xiaofeng Zheng[1,2,3], Jian Jiang[7], Ji
[1]Institute for Network
[2]Department of Compu
[3]Tsinghua National Lab
[4]Interna
[5]M
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION
Abstract
A cookie can contain a "secure" flag, indi
should be only sent over an HTTPS connecti
is no corresponding flag to indicate how a
set: attackers who act as a man-in-the-midd
porarily on an HTTP session can inject co
will be attached to subsequent HTTPS conne
ilar attacks can also be launched by a web at
related domain. Although an acknowledged
not yet been studied thoroughly. This pape
this gap with an in-depth empirical assessm
injection attacks. We find that cookie-relate
ities are present in important sites (such as
Bank of America), and can be made wors
plementation weaknesses we discovered i
browsers (such as Chrome, Firefox, and S
successful attacks have included privacy v
The cookie monster
in our browsers
@filedescriptor
HITCON 2019



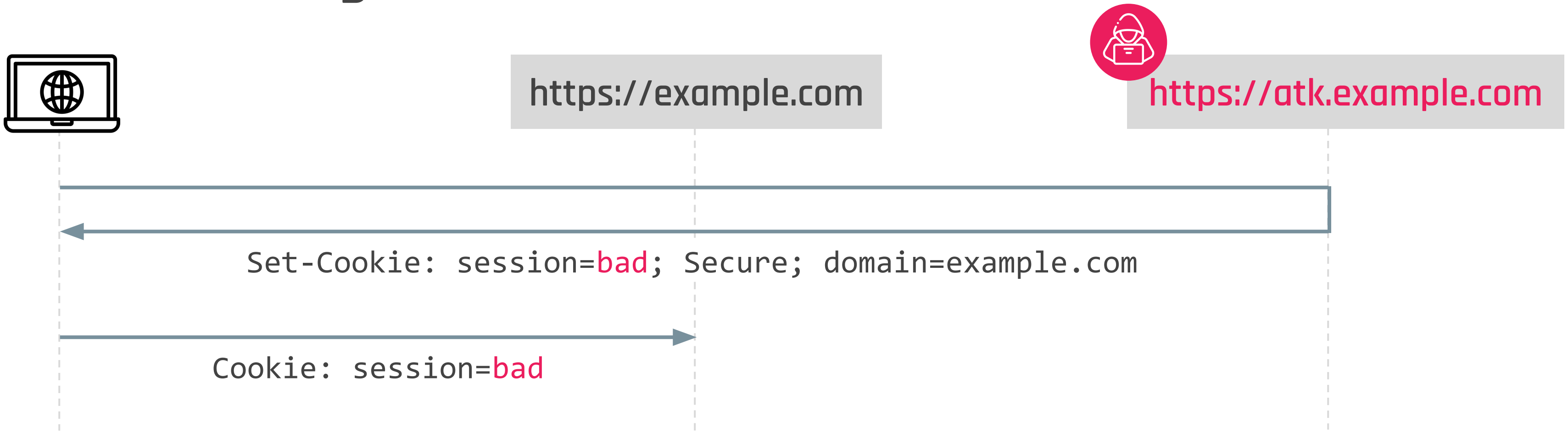
8.6. Weak Integrity

Cookies do not provide integrity guarantees for sibling domains (and their subdomains). For example, consider foo.site.example and bar.site.example. The foo.site.example server can set a cookie with a Domain attribute of "site.example" (possibly overwriting an existing "site.example" cookie set by bar.site.example), and the user agent will include that cookie in HTTP requests to bar.site.example. In the worst case, bar.site.example will be unable to distinguish this cookie from a cookie it set itself. The foo.site.example server might be able to leverage this ability to mount an attack against bar.site.example. [...]

An active network attacker can also inject cookies into the Cookie header field sent to https://site.example/ by impersonating a response from http://site.example/ and injecting a Set-Cookie header field. The HTTPS server at site.example will be unable to distinguish these cookies from cookies that it set itself in an HTTPS response. An active network attacker might be able to leverage this ability to mount an attack against site.example even if site.example uses HTTPS exclusively. [...]

Finally, an attacker might be able to force the user agent to delete cookies by storing a large number of cookies. Once the user agent reaches its storage limit, the user agent will be forced to evict some cookies. Servers SHOULD NOT rely upon user agents retaining cookies.

Cookie Tossing (Same-site Attacker)



Attributes					Flags	
Expires	Max-Age	Domain	Path	SameSite	Secure	HttpOnly

Path useful to prioritize cookies

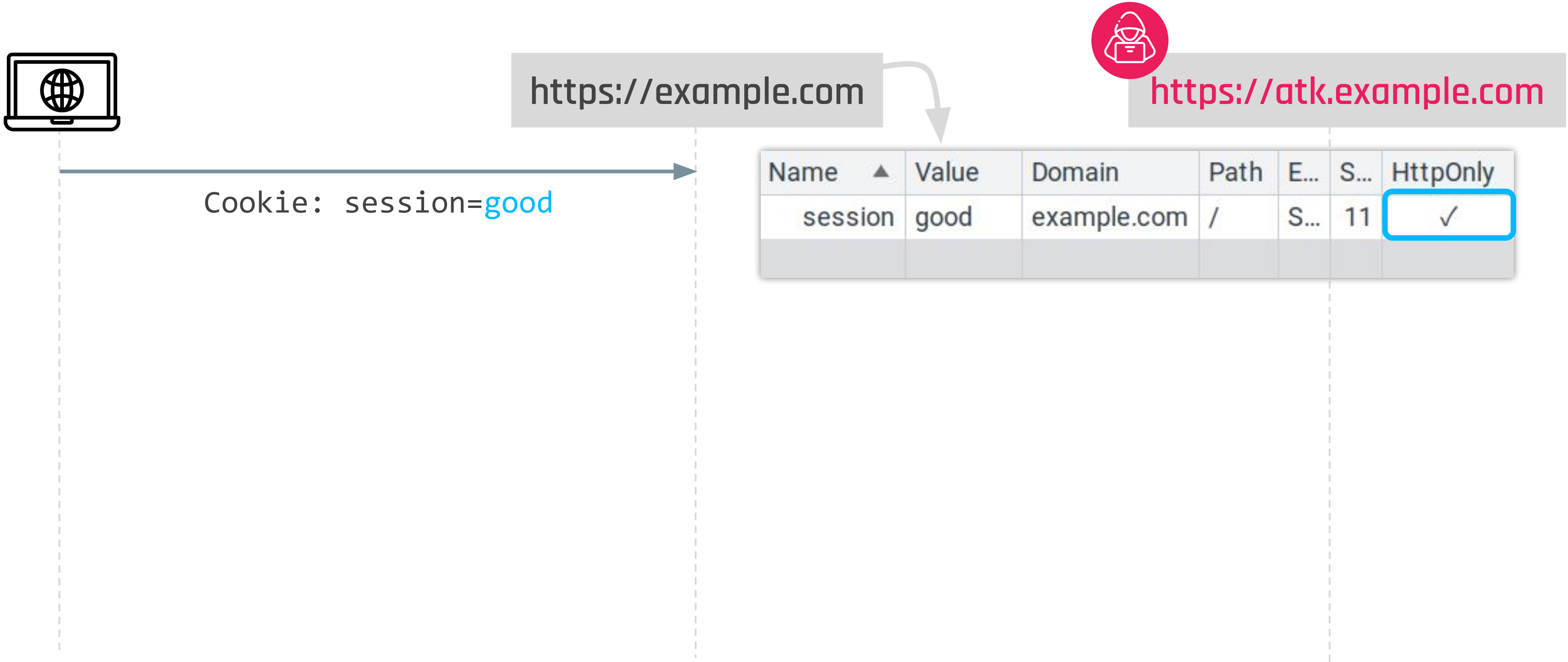
SameSite does not matter here!

Cookie Tossing (Network Attacker)

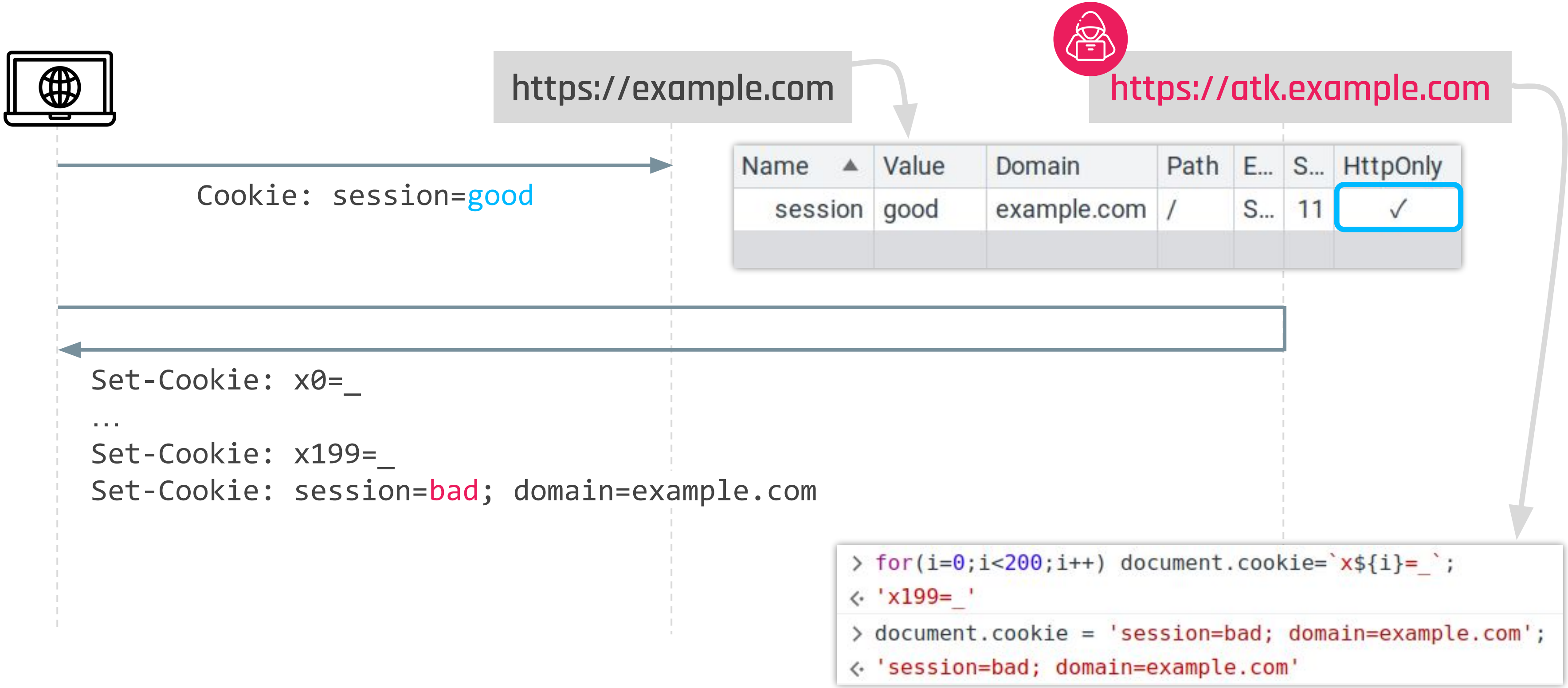
Cookies do not follow the **Same Origin Policy**

Can also be a subdomain over HTTP and the forged request contains a **domain cookie**

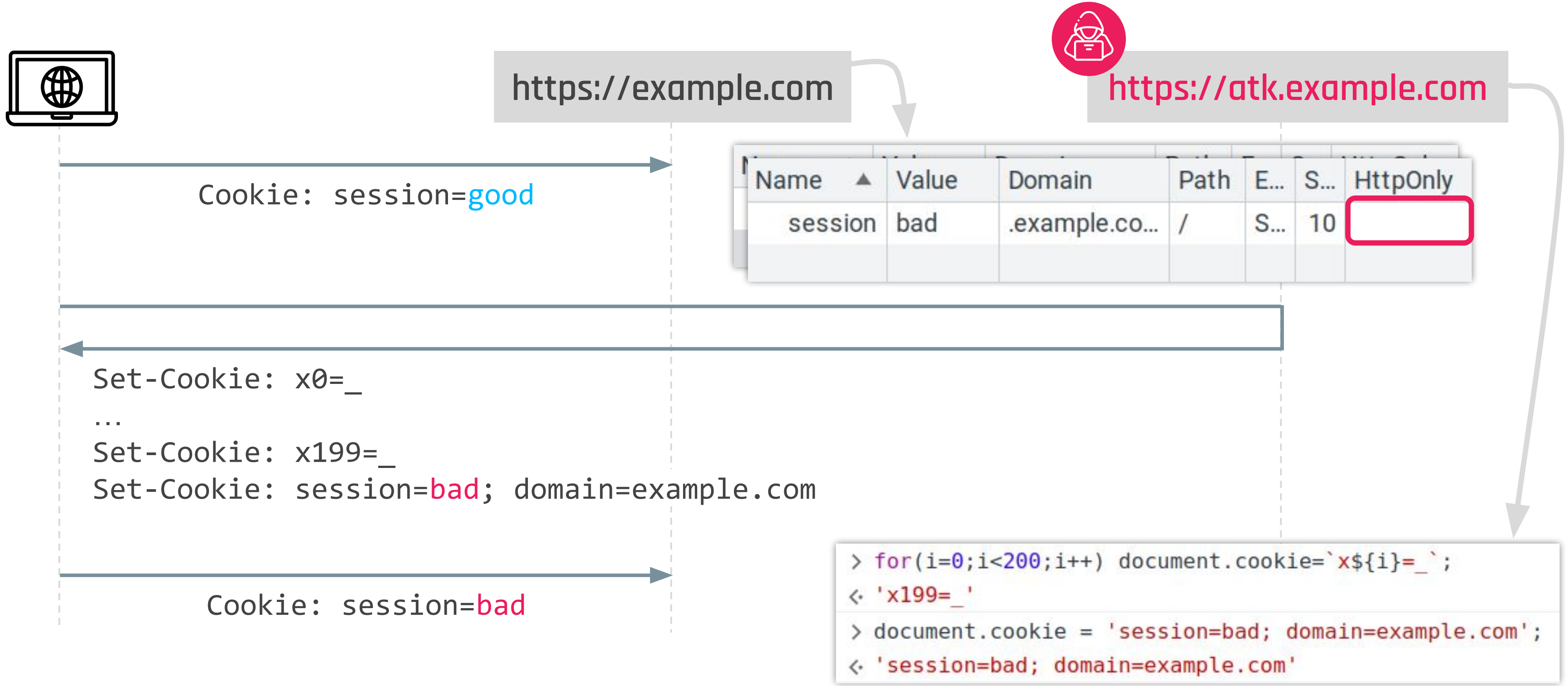
Cookie Eviction (Same-site & Network Attacker)



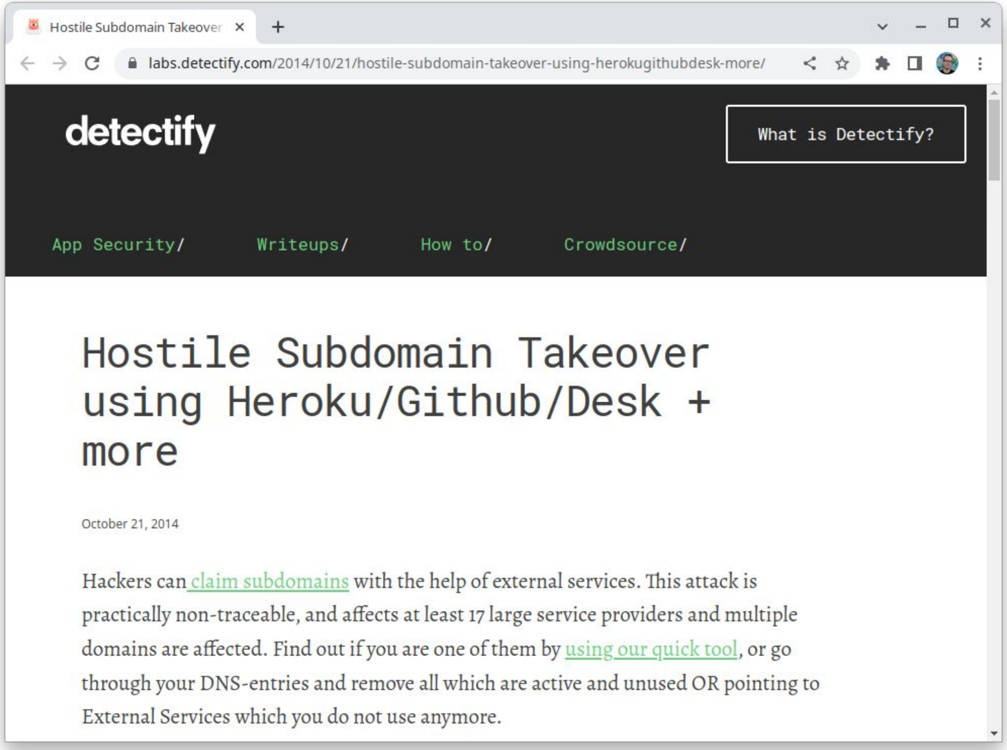
Cookie Eviction (Same-site & Network Attacker)



Cookie Eviction (Same-site & Network Attacker)



Threat Models



Dangling DNS Records

Discontinued Services

Threat Models

1520 vulnerable subdomains

Can I Take Your Subdomain? Exploring Same-Site Attacks in the Modern Web

Marco Squarcina[1] Mauro Tempesta[1] Lorenzo Veronese[1] Stefano Calzavara[2] Matteo Maffei[1]
[1] *TU Wien* [2] *Università Ca' Foscari Venezia & OWASP*

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

2021

Abstract

Related-domain attackers control a sibling domain of their target web application, e.g., as the result of a subdomain takeover. Despite their additional power over traditional web attackers, related-domain attackers received only limited attention from the research community. In this paper we define and quantify for the first time the threats that related-domain attackers pose to web application security. In particular, we first clarify the capabilities that related-domain attackers can acquire through different attack vectors, showing that different instances of the related-domain attacker concept are worth attention. We then study how these capabilities can be abused to compromise web application security by focusing on different angles, including cookies, CSP, CORS, postMessage, and domain relaxation. By building on this framework, we report on a large-scale security measurement on the top 50k domains from the Tranco list that led to the discovery of vulnerabilities in 887 sites, where we quantified the threats posed by related-domain attackers to popular web applications.

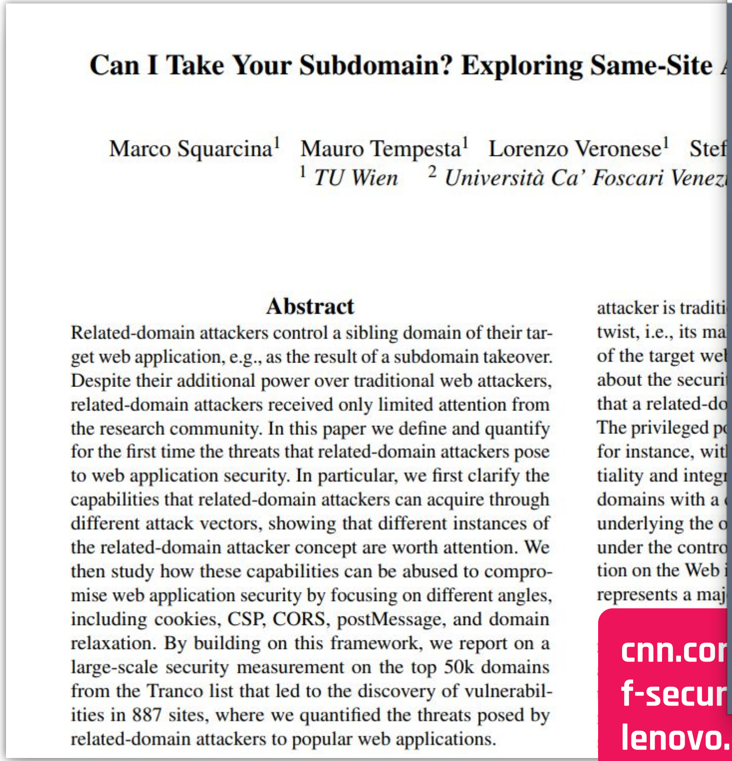
attacker is traditionally defined as a web attacker with an extra twist, i.e., its malicious website is hosted on a sibling domain of the target web application. For instance, when reasoning about the security of `www.example.com`, one might assume that a related-domain attacker controls `evil.example.com`. The privileged position of a related-domain attacker endows it, for instance, with the ability to compromise cookie confidentiality and integrity, because cookies can be shared between domains with a common ancestor, reflecting the assumption underlying the original Web design that related domains are under the control of the same entity. Since client authentication on the Web is mostly implemented on top of cookies, this represents a major security threat.

cnn.com, **nih.gov**, **cisco.com**, **f-secure.com**, **harvard.edu**, **lenovo.com**, ...

Dangling DNS Records

- **Discontinued Services**
- **Corporate Networks**
- **Expired Domains**
- **Roaming Services**
- **Deprovisioned Cloud Instances**
- **Dynamic DNS Providers**

Threat Models


Can I Take Your Subdomain? Exploring Same-Site

Marco Squarcina[1] Mauro Tempesta[1] Lorenzo Veronese[1] Stef
[1] *TU Wien* [2] *Università Ca' Foscari Venez*

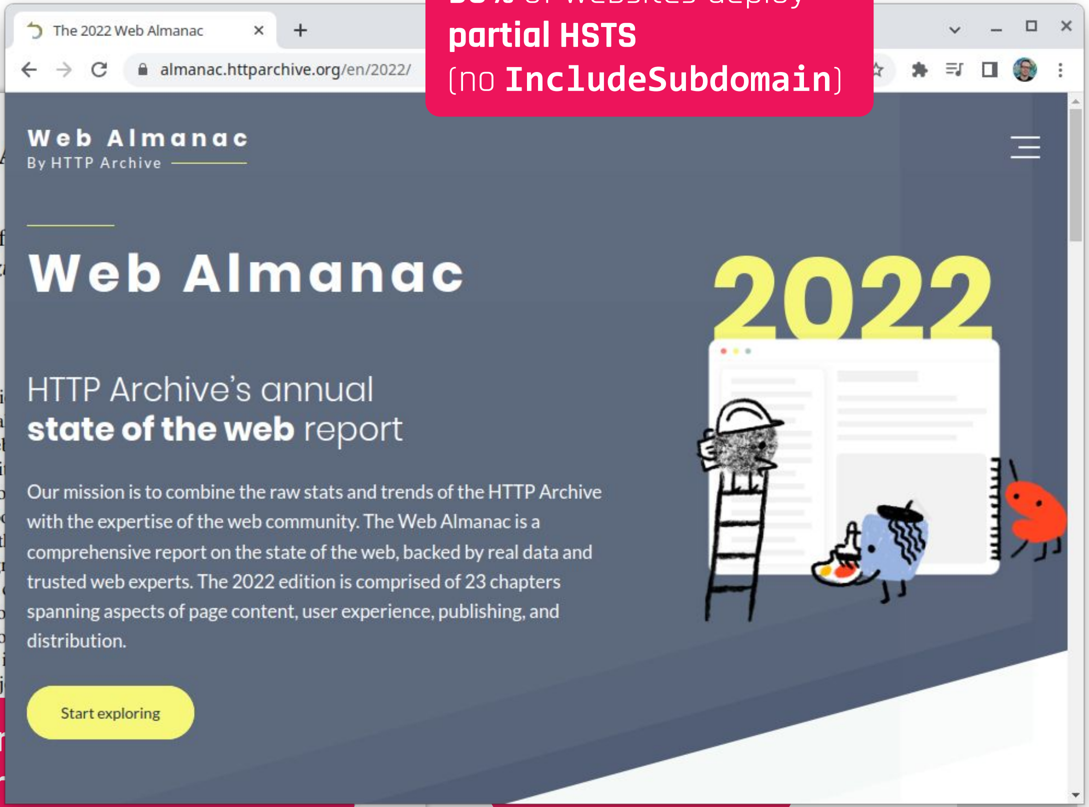
Abstract

Related-domain attackers control a sibling domain of their target web application, e.g., as the result of a subdomain takeover. Despite their additional power over traditional web attackers, related-domain attackers received only limited attention from the research community. In this paper we define and quantify for the first time the threats that related-domain attackers pose to web application security. In particular, we first clarify the capabilities that related-domain attackers can acquire through different attack vectors, showing that different instances of the related-domain attacker concept are worth attention. We then study how these capabilities can be abused to compromise web application security by focusing on different angles, including cookies, CSP, CORS, postMessage, and domain relaxation. By building on this framework, we report on a large-scale security measurement on the top 50k domains from the Tranco list that led to the discovery of vulnerabilities in 887 sites, where we quantified the threats posed by related-domain attackers to popular web applications.

attacker is traditi
twist, i.e., its ma
of the target we
about the securi
that a related-do
The privileged p
for instance, wit
tiality and integ
domains with a
underlying the
under the contro
tion on the Web
represents a maj

cnn.cor
f-secur
lenovo.com, ...

90% of websites deploy **partial HSTS** (no **IncludeSubdomain**)



Session Fixation & Login CSRF

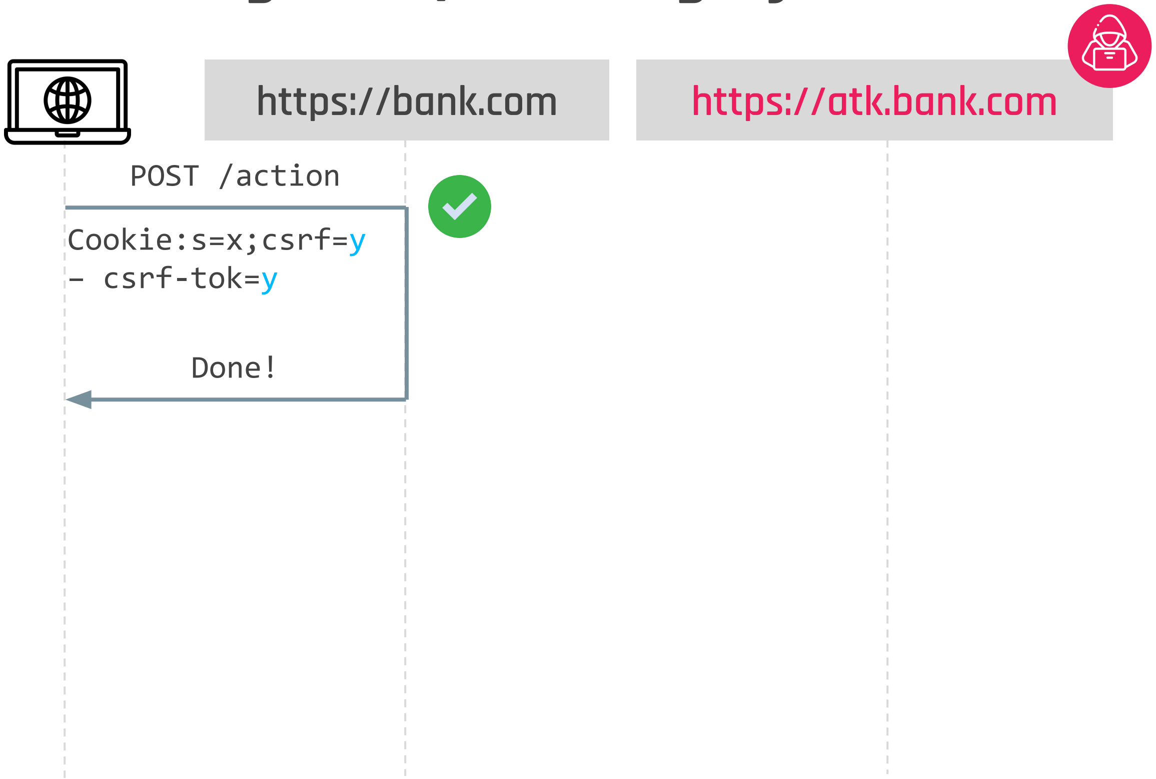
Session Fixation

- `bank.com` **does not refresh the session ID after login**
- Attacker obtains a pre-session `sid=s1` and tosses that cookie into Bob's browser
- Bob authenticates, promoting `sid=s1` to an authenticated session
- **Attacker hijacks Bob's session** using `s1`

Login CSRF

- Attacker has an account on `bank.com`, with cookie `sid=s2`
- Attacker tosses that cookie into Bob's browser
- When Bob visits `bank.com`, Bob is **authenticated as the attacker**, leaking sensitive information that can be later accessed by the attacker

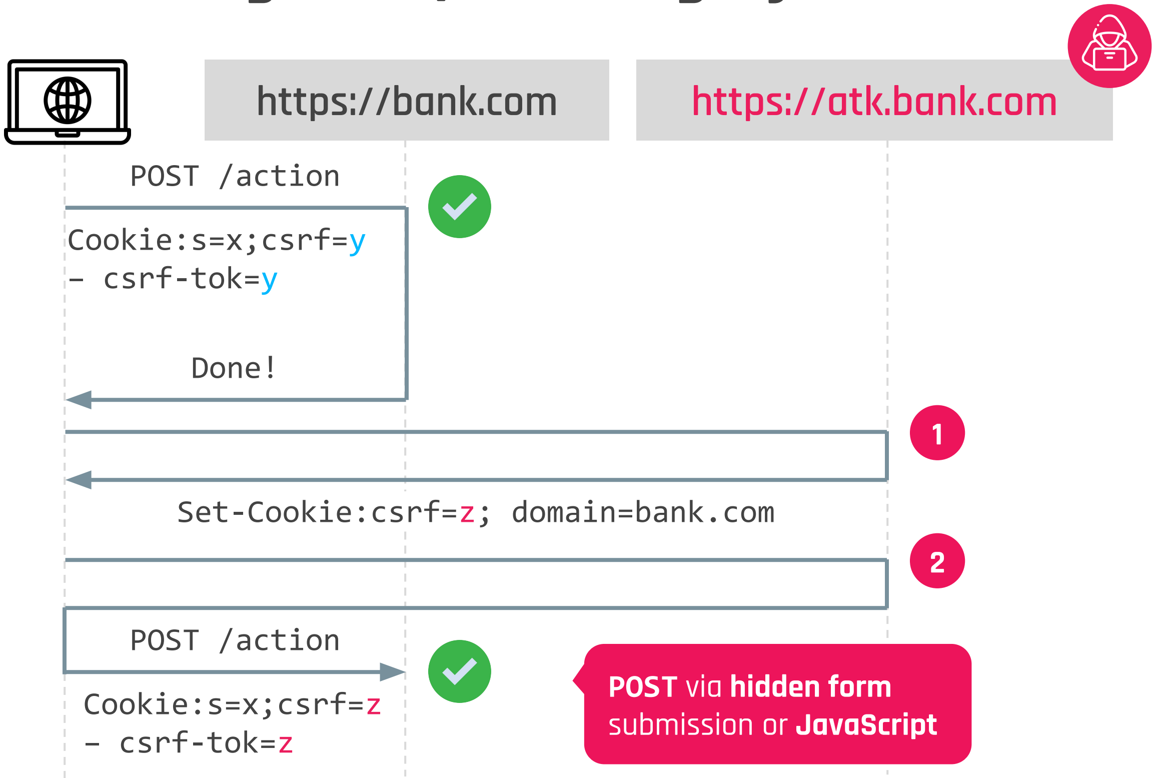
Cross-Origin Request Forgery (CORF)



Double-Submit

```
if cookie(csrf)==POST(csrf-tok):
    return True
return False
```

Cross-Origin Request Forgery (CORF)



Double-Submit

```
if cookie(csrf)==POST(csrf-tok):
     return True
return False
```

Wrong assumption: attacker can only manipulate the token, but not the cookie!

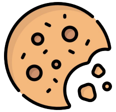
Trivially **vulnerable** against same-site attackers, just **toss** and **submit**!

Synchronizer Token Pattern

- Fixes Double Submit problems by **binding the CSRF token to the session**
- Store a **CSRF secret in the session** and use it to **generate CSRF tokens**

generate_func(**CSRF_secret**, params...) = **CSRF_token**

Attached to HTTP requests via hidden form field

Session  := <**id**, **CSRF_secret**>

Stored in the session

Verify := generate_func(**CSRF_secret**, params...) == **CSRF_token**

- Overwrite the session cookie? Deauth the user, **NO CORF**, attacker sad :'(

Synchronizer Token Pattern (Flask-login + Flask-WTF)

Flask web development, one drop at a time

https://bank.com

1 GET /login

<input csrf_token=**t0** type="hidden">

Set-Cookie: session={csrf=**s**, _id=**None**}#sign

2 POST /login

Cookie: session={csrf=**s**, _id=**None**}#sign
- user=bob&password=s3cur3&csrf_token=**t0**

Hi Bob <input csrf_token=**t1** type="hidden">

Set-Cookie: session={csrf=**s**, _id=**bob**}#sign

3 POST /action

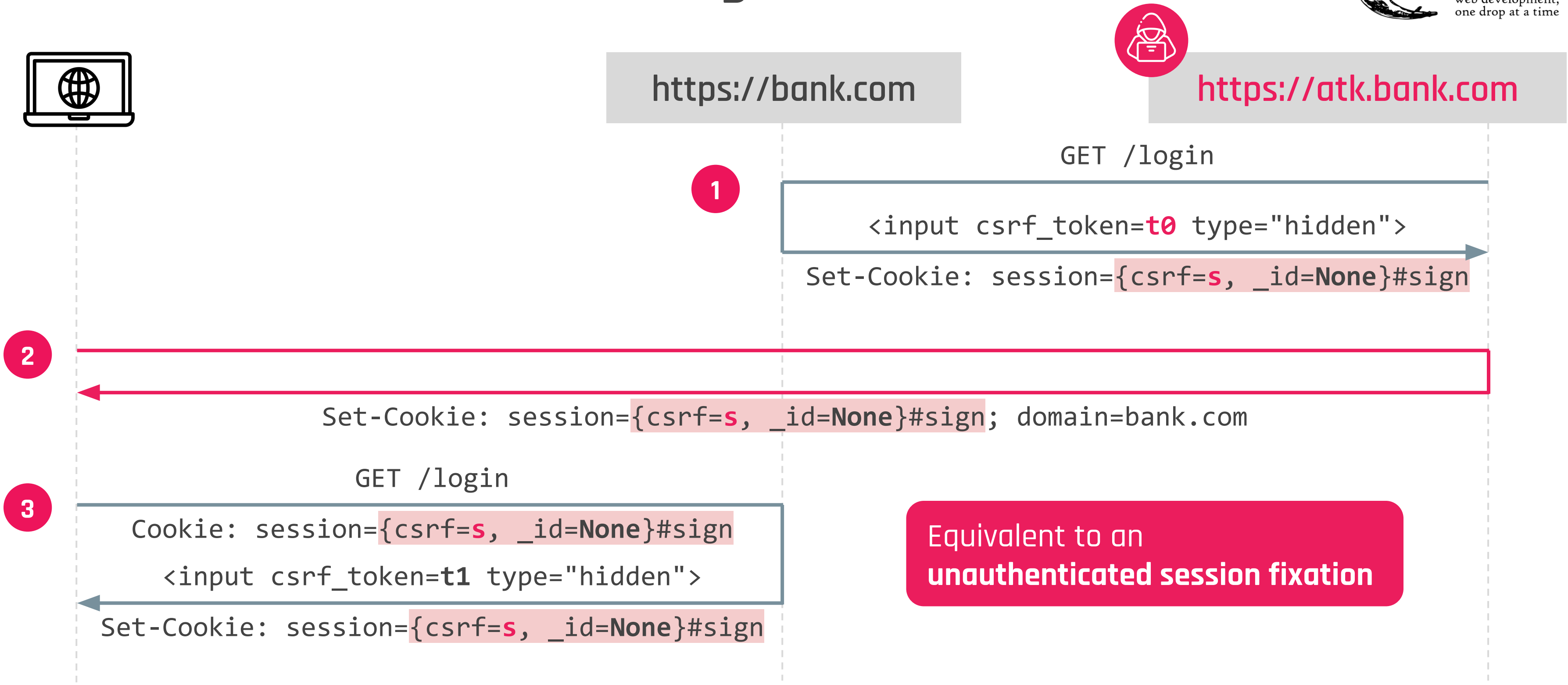
Cookie: session={csrf=**s**, _id=**bob**}#sign
- csrf_token=**t1**

```
s  = sha1(os.urandom(64)).hexdigest()

t0 = exp_time0##HMAC(SECRET,s#exp_time0)

t1 = exp_time1##HMAC(SECRET,s#exp_time1)

Verification:
  exp_time, hmac = token.split(“##”)
  if hmac == HMAC(SECRET, s#exp_time):
    return True
  return False
```

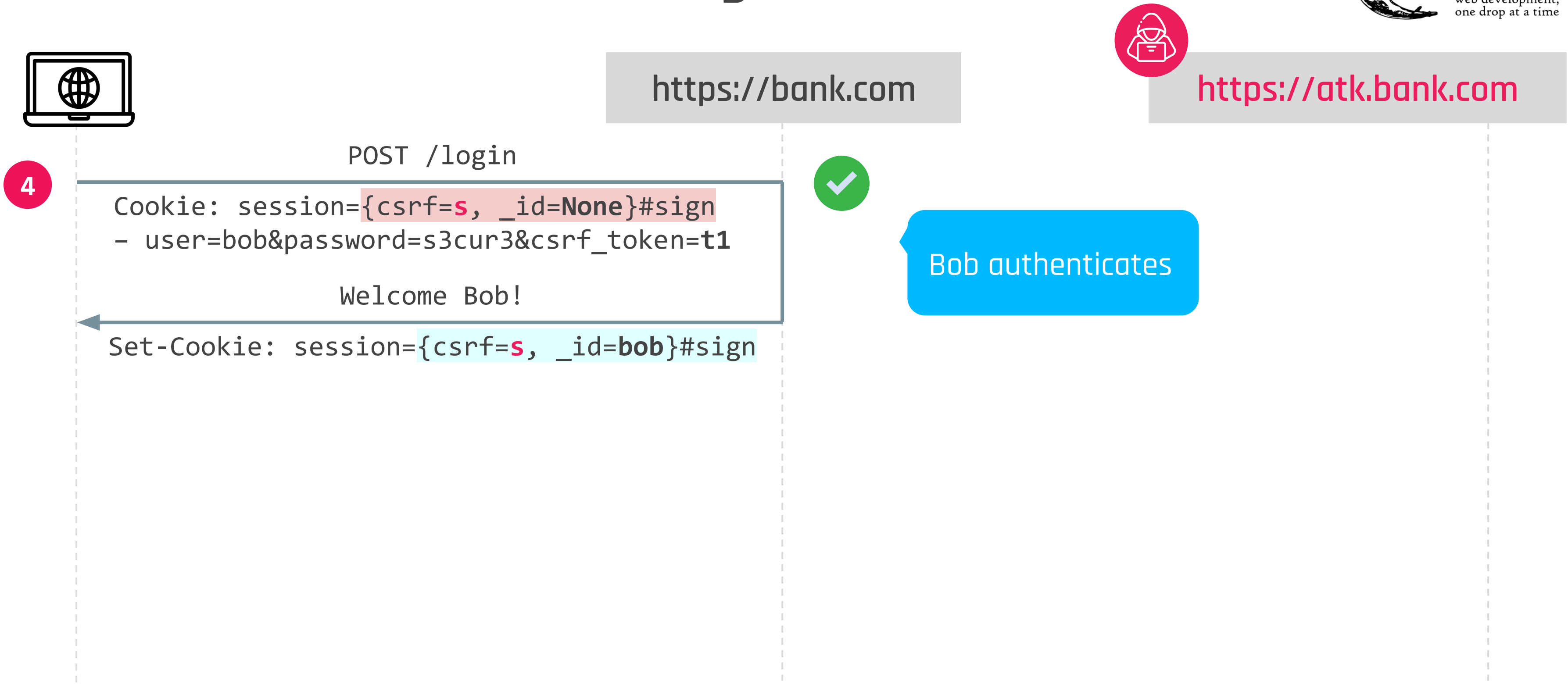
Synchronizer Token Pattern (Flask-login + Flask-WTF)


```
s  = sha1(os.urandom(64)).hexdigest()

t0 = exp_time0##HMAC(SECRET,s#exp_time0)

t1 = exp_time1##HMAC(SECRET,s#exp_time1)

Verification:
  exp_time, hmac = token.split("##")
  if hmac == HMAC(SECRET, s#exp_time):
    return True
  return False
```

CORF Token Fixation (Flask-login + Flask-WTF)



CORF Token Fixation (Flask-login + Flask-WTF)



CORF Token Fixation (Flask-login + Flask-WTF)

Flask
web development, one drop at a time

https://bank.com

https://atk.bank.com

4

POST /login

Cookie: session={csrf=**s**, _id=**None**}#sign
- user=bob&password=s3cur3&csrf_token=**t1**

Welcome Bob!

Set-Cookie: session={csrf=**s**, _id=**bob**}#sign

Bob authenticates

5

POST /action

Cookie: session={csrf=**s**, _id=**bob**}#sign
- csrf_token=**t0**

The **CSRF secret s** is not refreshed during login!
The **CSRF token t0** known by the attacker is valid for Bob's session!

CORF Token Fixation

- Bypasses faulty implementations of the **Synchronizer Token Pattern**
- Caused by the **CSRF secret** in the session **not being renewed** upon login
- The attacker does not need to know the CSRF secret, but only an **unauthenticated session id** and a **valid CSRF token** for that session
- Works against **server-side** and **client-side** session handling implementations
- User already logged-in? No problem, **force a deauth** and toss the attacker's pre-session, either via eviction or request to `/logout` endpoint

CORF Token Fixation (CodeIgniter4)

CodeIgniter

https://bank.com

1

GET /login

<input csrf_token=**t0** type="hidden">

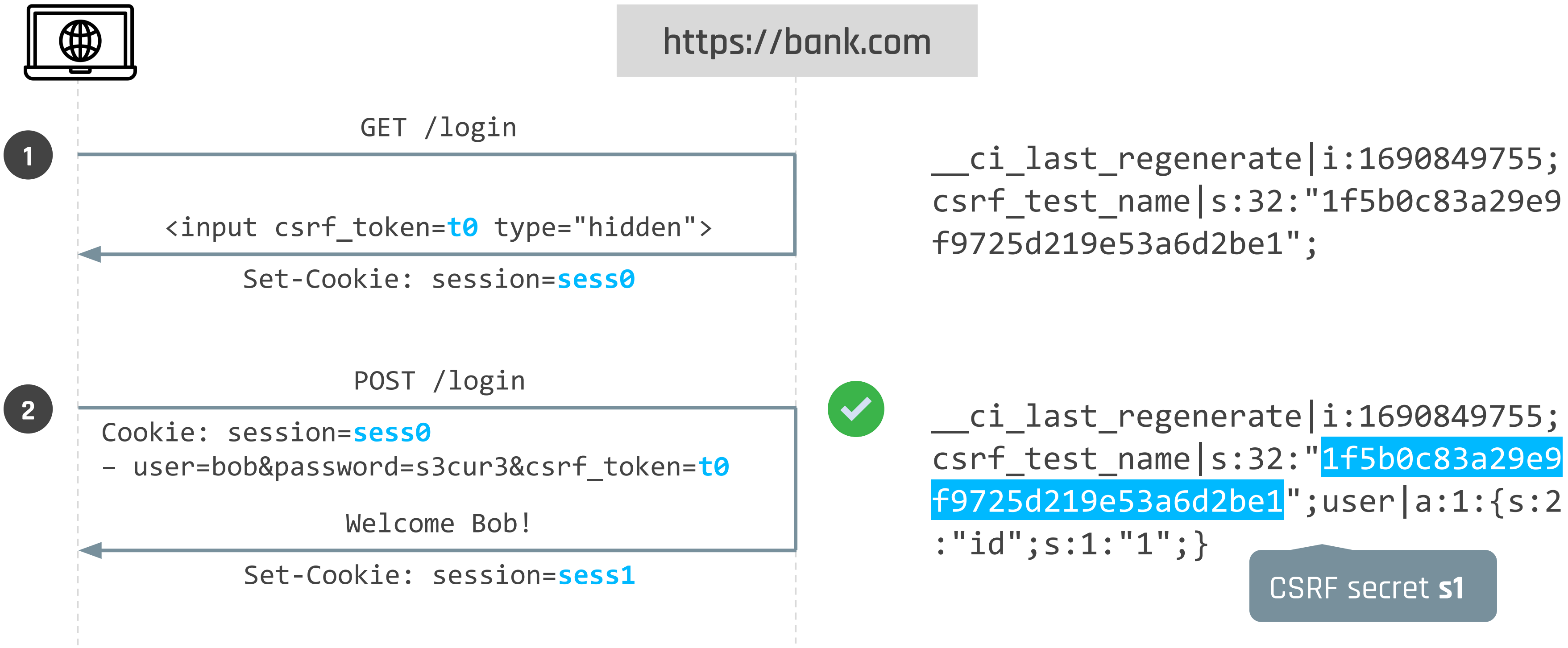
Set-Cookie: session=**sess0**

```
__ci_last_regenerate|i:1690849755;
csrf_test_name|s:32:"47be9758fe558
98f1958bd201764a0be";
```

CSRF secret **s0**

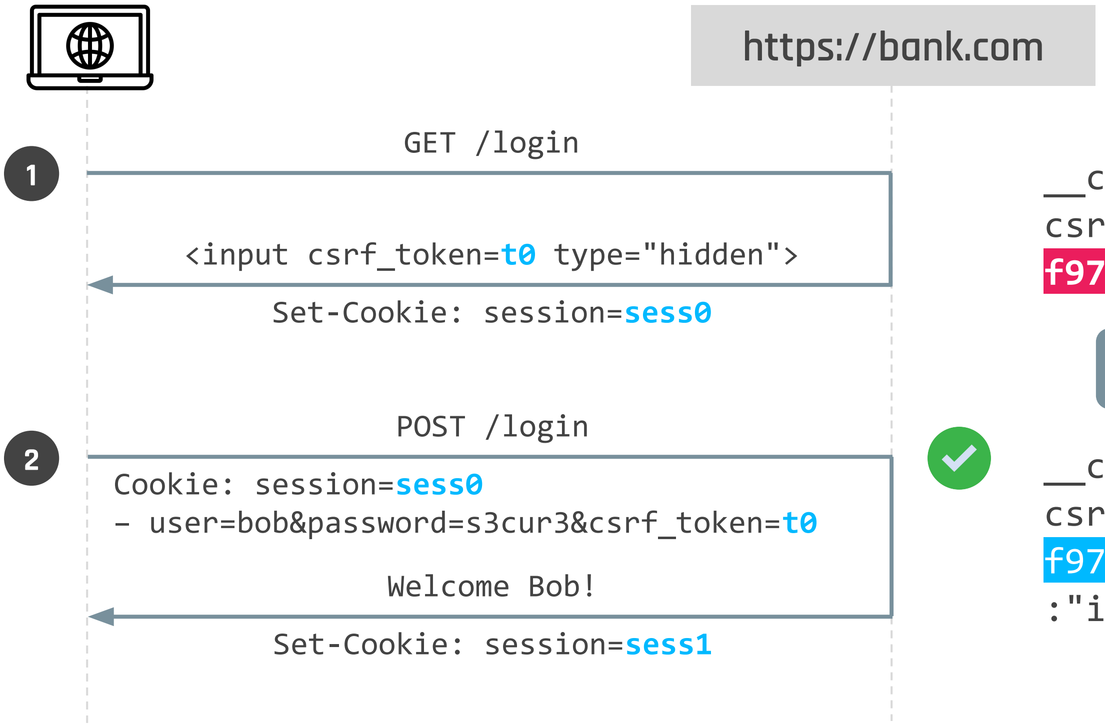
CORF Token Fixation (CodeIgniter4)

CodeIgniter



CORF Token Fixation (CodeIgniter4)

CodeIgniter




```
__ci_last_regenerate|i:1690849755;
csrf_test_name|s:32:"1f5b0c83a29e9
f9725d219e53a6d2be1";
```

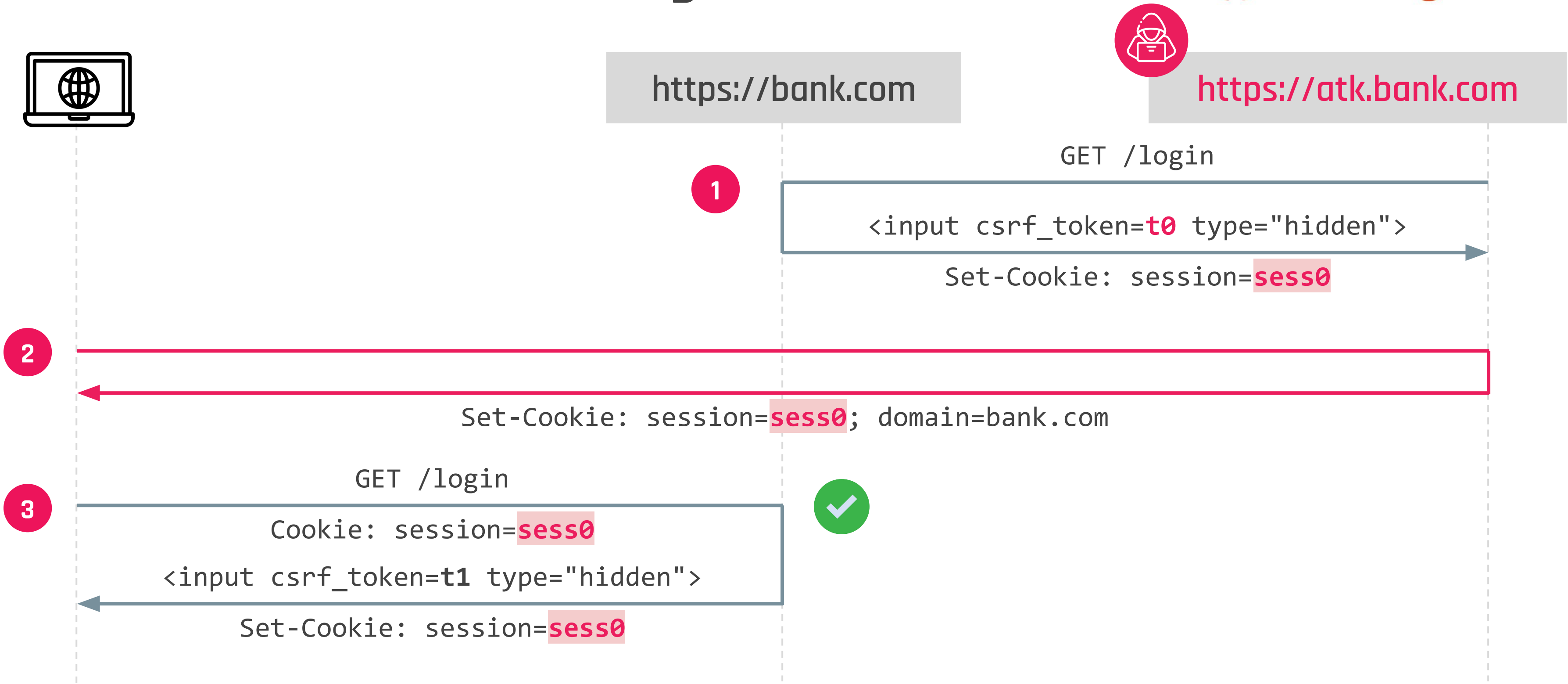
CSRF secret **s1**

```
__ci_last_regenerate|i:169084975 ;
csrf_test_name|s:32:"1f5b0c83a29e9
f9725d219e53a6d2be1";user|a:1:{s:2
:"id";s:1:"1";}
```

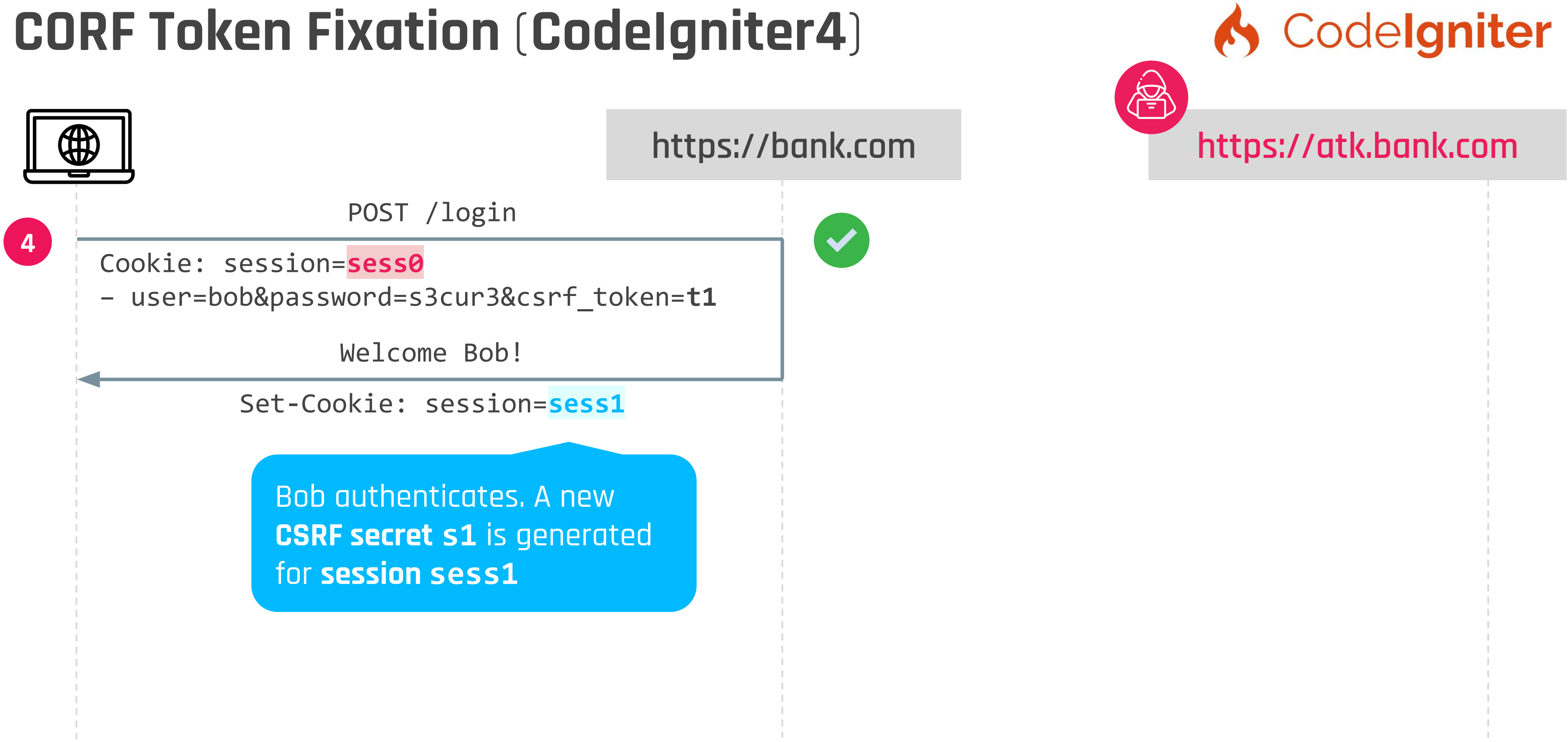
CSRF secret **s1**

CORF Token Fixation (CodeIgniter4)

CodeIgniter



CORF Token Fixation (CodeIgniter4)



CORF Token Fixation (CodeIgniter4)

CodeIgniter

https://bank.com

https://atk.bank.com

4

POST /login

Cookie: session=sess0
- user=bob&password=s3cur3&csrf_token=t1

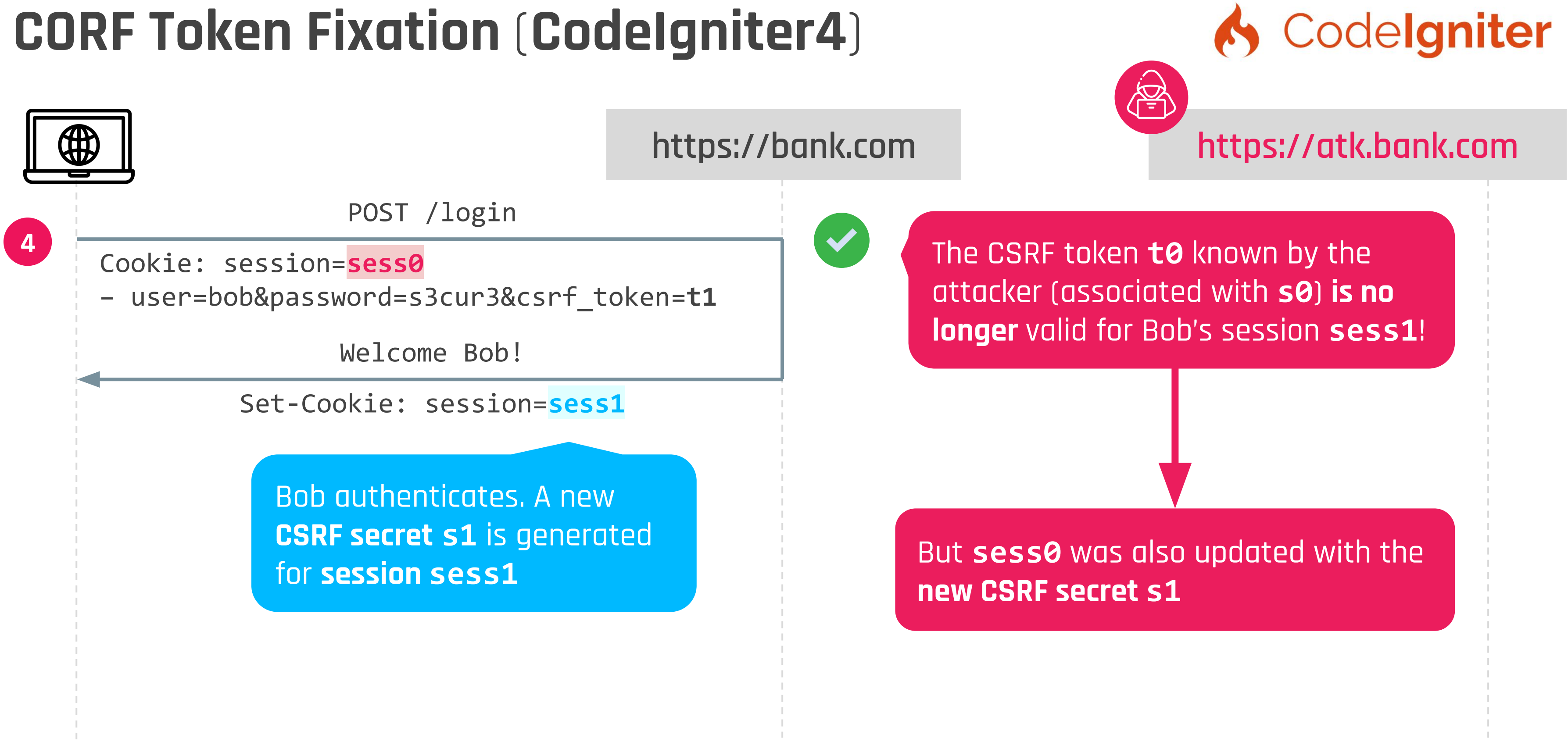
Welcome Bob!

Set-Cookie: session=sess1

Bob authenticates. A new **CSRF secret s1** is generated for **session sess1**

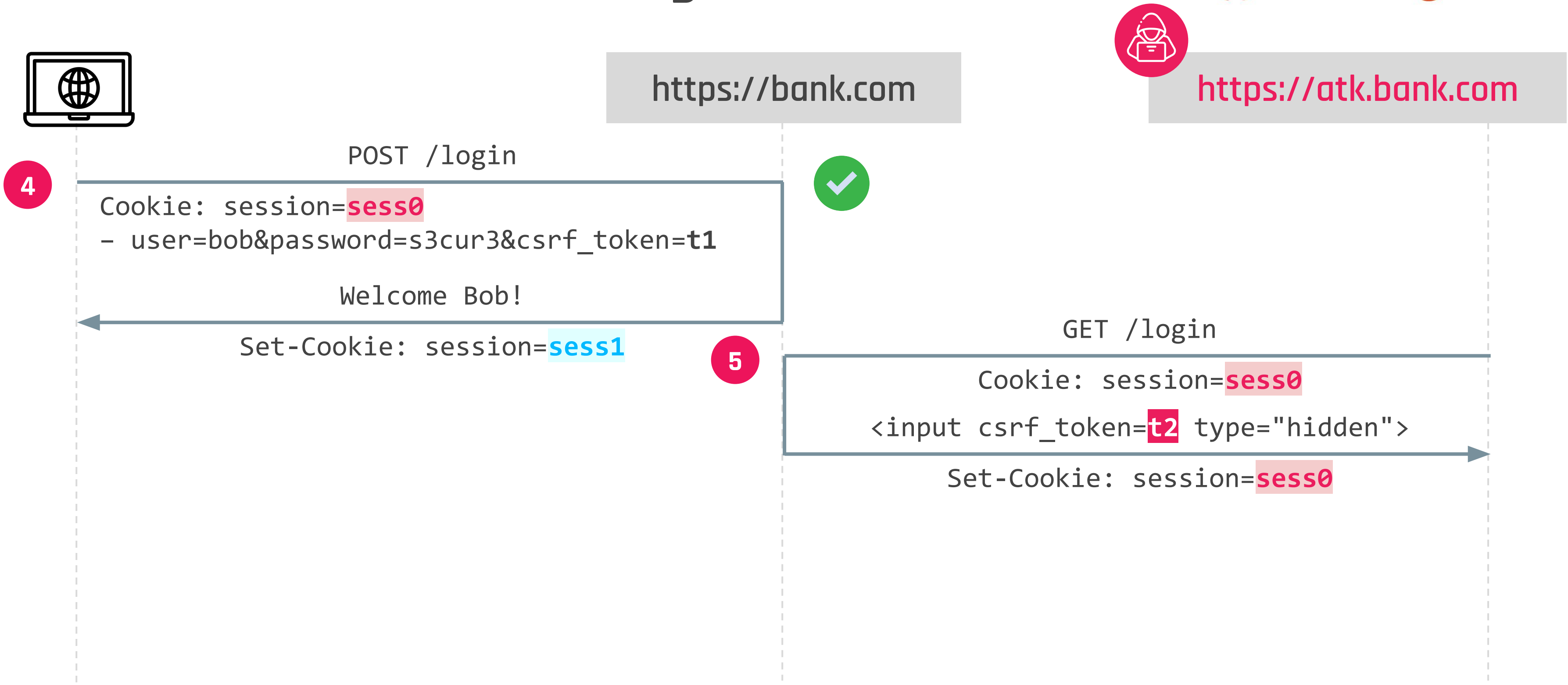
The CSRF token **t0** known by the attacker (associated with **s0**) **is no longer** valid for Bob's session **sess1**!

CORF Token Fixation (CodeIgniter4)

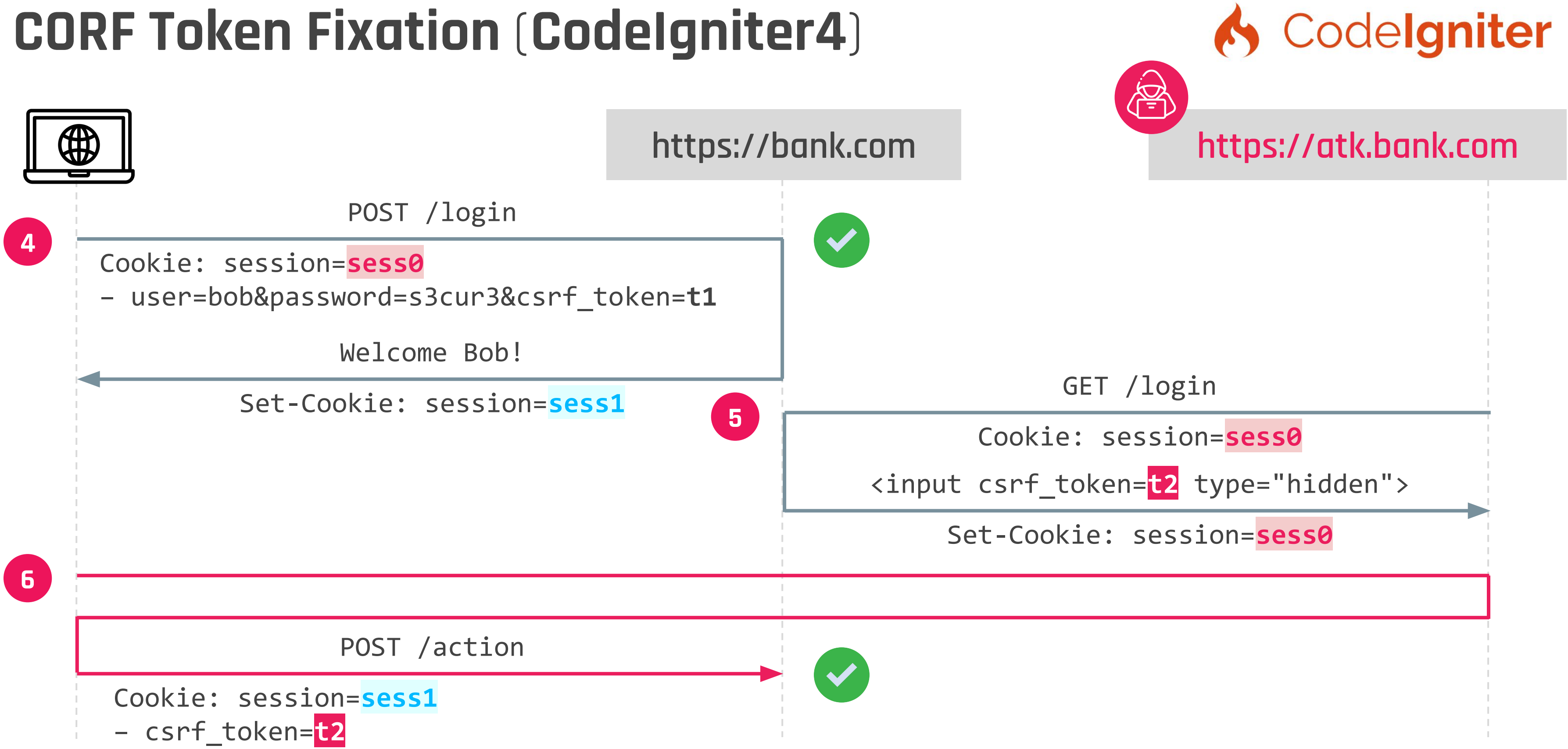


CORF Token Fixation (CodeIgniter4)

CodeIgniter



CORF Token Fixation (CodeIgniter4)



Web Frameworks Analysis

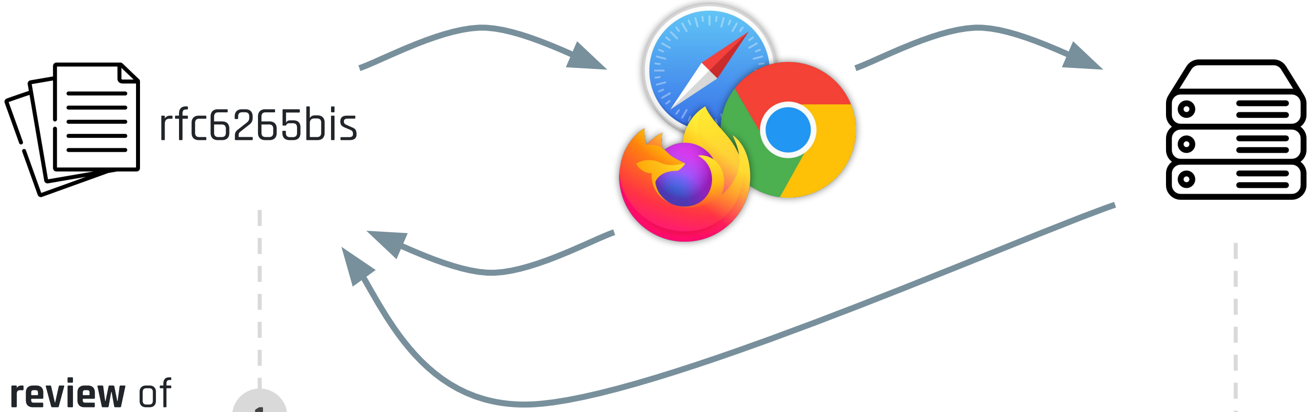
Framework (9/13 vulnerable)	Broken STP	Default DS	Session Fixation	
Express (passport + csurf)	●		●	CVE-2022-25896
Koa (koa-passport + csrf)	●			
Fastify (fastify/passport + csrf-protection)	●	●	●	CVE-2023-29020 CVE-2023-27495 CVE-2023-29019
Sails* (csurf)	●		●	
Flask (flask-login+flask-wtf)	●			
Tornado		●		
Symfony (security-bundle)	●			CVE-2022-24895
CodeIgniter4 (shield)	●	●		CVE-2022-35943
Yii2		●		

*affects the bootstrap template app

Are  Getting Better?

Methodology

2 **Browser testing suite** for tossing & eviction: check **discrepancies**



Manual **review** of the cookie standard 1

Simple **differential fuzzing** to test server implementations 3

Strict Secure


```
HTTP Working Group                                              M. West
Internet-Draft                                              Google, Inc
Updates: 6265 (if approved)                           September 5, 2016
Intended status: Standards Track
Expires: March 9, 2017


   Deprecate modification of 'secure' cookies from non-secure origins
                    draft-ietf-httpbis-cookie-alone-01
```

Browsers now **block setting a cookie without the Secure flag** if there is already a secure cookie in that site with the same name.

Prevents tossing from network attackers. Also **eviction doesn't work** as secure cookies are partitioned separately from non-secure cookies.

Prefixes


```
HTTP Working Group                                          M. West
Internet-Draft                                          Google, Inc
Updates: 6265 (if approved)                       February 23, 2016
Intended status: Standards Track
Expires: August 26, 2016
```

Cookie Prefixes
draft-ietf-httpbis-cookie-prefixes-00

__Secure- cookies must be set from a secure origin and include the Secure attribute.

__Host- cookies, additionally, must **NOT be set with the Domain** attribute and **Path=/**.

__Host- cookies are **high-integrity cookies** even against same-site attackers!

Collisions

Werkzeug <2.2.3

Set-Cookie:	Cookie:	Key	Value	Server <key, value>
foo=	foo=	foo		<foo, >
=foo	foo		foo	
=foo=	foo=		foo=	
==foo	=foo		=foo	
foo	foo		foo	

Collisions

Werkzeug <2.2.3

Set-Cookie:	Cookie:	Key	Value	Server <key, value>
foo=	foo=	foo		<foo, >
=foo	foo		foo	<foo, >
=foo=	foo=		foo=	<foo, >
==foo	=foo		=foo	<foo, >
foo	foo		foo	<foo, >

Collisions

[RFC6265bis] Accept nameless cookies. (#1018)

Browse files

This patch alters the cookie parsing algorithm to treat `Set-Cookie: token` as creating a cookie with an empty name and a value of "token". It also rejects cookies with neither names nor values (e.g. `Set-Cookie:` and `Set-Cookie: =`.

Closes #159.

main (#1018)

draft-ietf-httpbis-unprompted-auth-02 ... b68e4ff

committed on Jan 10, 2020

1 parent c43cdae commit 0178223

Werkzeug <2.2.3

Set	Server <key, value>
foo	<foo, >
=fo	<foo, >
=fo	<foo, >
==f	<foo, >
foo	<foo, >

Bypassing __Host-

http://atk.bank.com

https://bank.com

```
Set-Cookie: __Host-session=good;
            Secure; Path=/
```

Bypassing __Host-

http://atk.bank.com

https://bank.com

```
Set-Cookie: __Host-session=good;
            Secure; Path=/
```

```
Set-Cookie: =__Host-session=bad; Path=/app;
            domain=bank.com
```

Bypassing __Host-

http://atk.bank.com

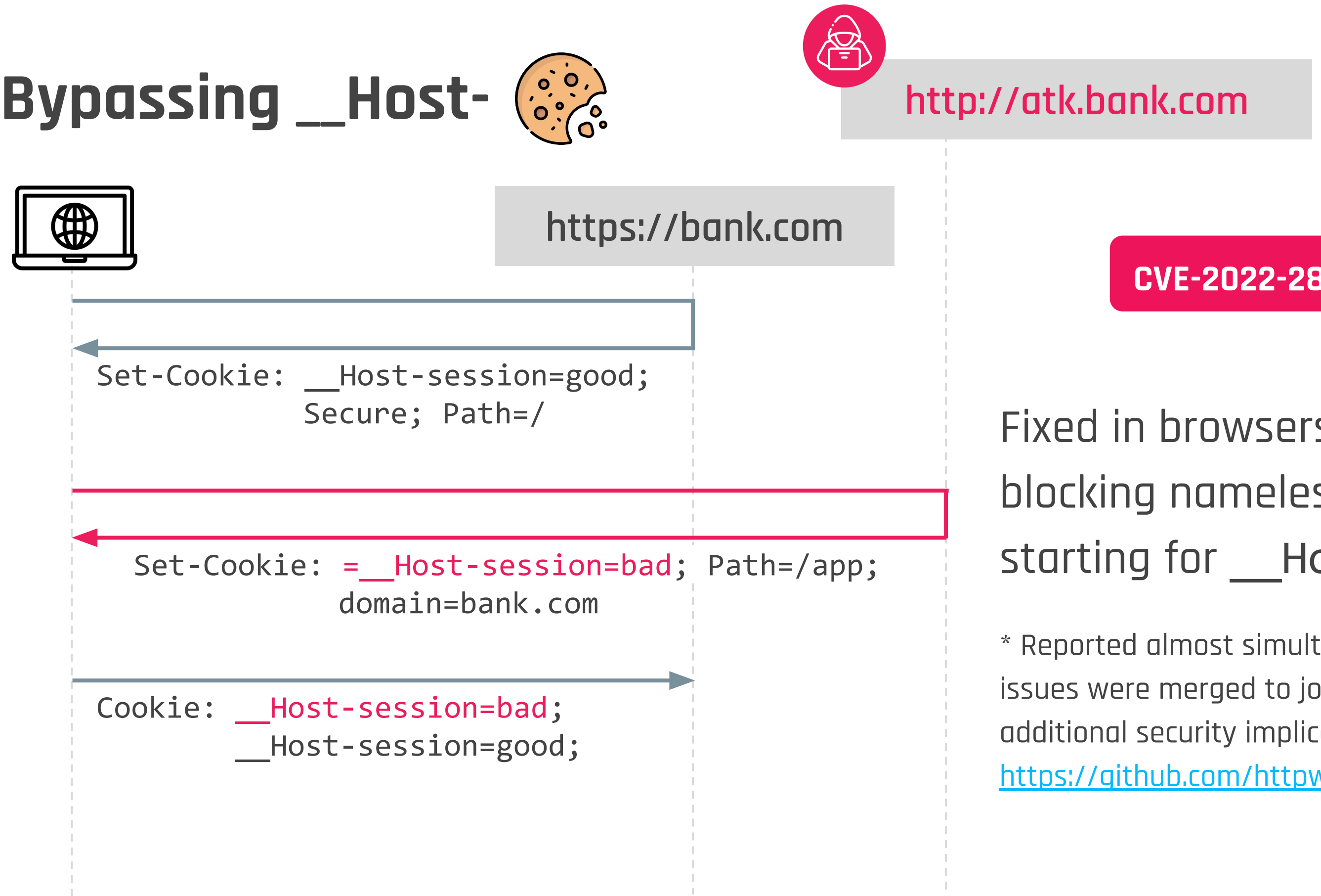
https://bank.com

```
Set-Cookie: __Host-session=good;
            Secure; Path=/
```

```
Set-Cookie: =__Host-session=bad; Path=/app;
            domain=bank.com
```

```
Cookie: __Host-session=bad;
        __Host-session=good;
```

Bypassing __Host-



CVE-2022-2860* **CVE-2022-40958***

Fixed in browsers and rfc6265bis by blocking nameless cookies with value starting for `__Host-` or `__Secure-`

* Reported almost simultaneously with **Axel Chong**, our issues were merged to jointly discuss mitigations and additional security implications. See also https://github.com/httpwg/http-extensions/issues/2229

Bypassing __Host- (after the **fix**)

Amazon API Gateway

- **Serialization collisions** could still be used to bypass `__Host-` against chains of pairs
- Fixed in **AWS Lambda proxy integration for HTTP APIs** after our report

CVE-2022-2860* CVE-2022-40958*

Fixed in browsers and rfc6265bis by blocking nameless cookies with value starting for `__Host-` or `__Secure-`

* Reported almost simultaneously with **Axel Chong**, our issues were merged to jointly discuss mitigations and additional security implications. See also https://github.com/httpwg/http-extensions/issues/2229

Bypassing Strict Secure

Still working!

Set-Cookie: =session=bad

Name	Value	Domain	Path	E...	S...	H...	Secure
session	good	bank.com	/	S...	11		✓
	session=bad	.bank.com	/app	S...	11		

Bypassing __Host- (with the help of the **server**)

- Popular programming languages / Web frameworks **diverge from the spec**
- Client / server inconsistencies. Security implications?

Werkzeug <2.2.3

```
Cookie: __Host-sess=bad
Cookie: =__Host-sess=bad
Cookie: ========__Host-sess=bad
```

Bypassing __Host- (with the help of the **server**)

- Popular programming languages / Web frameworks **diverge from the spec**
- Client / server inconsistencies. Security implications?

Werkzeug <2.2.3

CVE-2023-23934

```
Cookie:  __Host-sess=bad
Cookie:  =__Host-sess=bad
Cookie:  ========__Host-sess=bad
```

Parsed as the **same cookie**

Leading '=' are stripped out while parsing the cookie string!

Bypass with, e.g.,

```
Set-Cookie: ==__Host-sess=bad
```

Bypassing __Host- (with the help of the **server**)

- Popular programming languages / Web frameworks **diverge from the spec**
- Client / server inconsistencies. Security implications?

PHP <8.1.11

```
Cookie: __Host-sess=bad
Cookie: _ Host-sess=bad
Cookie: ..Host-sess=bad
```

Parsed as the **same cookie**

Bypassing __Host- (with the help of the **server**)

- Popular programming languages / Web frameworks **diverge from the spec**
- Client / server inconsistencies. Security implications?

PHP <8.1.11

CVE-2022-31629

```
Cookie: __Host-sess=bad
Cookie: _Host-sess=bad
Cookie: ..Host-sess=bad
```

Parsed as the **same cookie**

`register_globals` heritage:
‘ ’ . [are replaced by _ in the
`$_COOKIE` superglobal array

Did you know? `Cookie: a[b]=c`
Parsed as `{"a":{"b":"c"}}`

Desynchronization Issues

1. `https://bank.com` set a secure 🍪
   `Set-Cookie: sess=good; Secure`
2. `http://bank.com` sets a non-secure 🍪 via JS
   `document.cookie = 'sess=bad'`

CVE-2023-29547

Fixed in Firefox 112

Caused by restrictions imposed by the FF implementation of **Site Isolation** (**Project Fission**)

EXPECTATION	`sess=bad` is **not set** (Strict Secure 🍪)
REALITY	Cookie not set, but `document.cookie` at `http://bank.com` returns `sess=bad`

Desynchronization Issues

1 `https://atk.bank.com`

Fixed in Firefox 115

```
>> for(let i=0; i<400; i++) document.cookie = `a${i}=_; domain=bank.com`
⚠ ▶ Some cookies are misusing the recommended "SameSite" attribute  400
← "a399=_; domain=bank.com"
>> document.cookie.split('; ').length
← 400
>> window.open("https://bank.com")
```

Could introduce vulnerabilities in frontends trusting **`document.cookie`** to set **custom HTTP headers** like **ASP.NET** and **Angular**

2 **Delete** 🍪 via `Set-Cookie` (exp. date), `Clear-Site-Data` header, or manually

3 The first 240 🍪 are still in `Document.cookie` in the original and opened window (survives reloads and schemeful navigations)

Takeaways

- Many battle-tested Web frameworks and libraries had **concerning session integrity vulnerabilities**. Causes & consequences?
- **Legacy design** is still cursing modern applications: can we **move on without breaking the Web**?
- Developers are falling behind in **keeping track of Web standards**
- Composition issues or lack of understanding of the threat models? Apps in the wild?
- Backward compatibility issues? Is it possible to make deployment easier without trading on security?
- Lack of cohesiveness between browser vendors, developers, and authors of Web standards? Web platform changing too fast?

… and that's the way the cookie crumbles!

icons by **flaticon**

Thank You! Questions?

Marco Squarcina (TU Wien)
@blueminimal
https://infosec.exchange/@minimalblue
marco.squarcina@tuwien.ac.at

Pedro Adão (IST, Universidade de Lisboa)
@pedromigueladao
https://infosec.exchange/@pedroadao
pedro.adao@tecnico.ulisboa.pt

Paper available at https://github.com/SecPriv/cookiecrumbles