Where We Stand (or Fall): An Analysis of CSRF Defenses in Web Frameworks

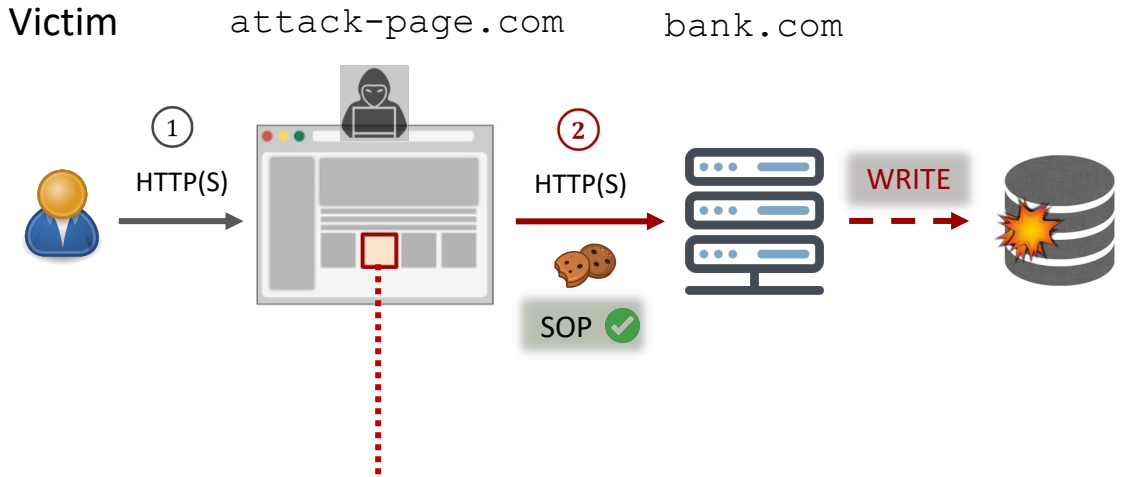
Xhelal Likaj †, Soheil Khodayari * and Giancarlo Pellegrino *

† Saarland University

* CISPA Helmholtz Center for Information Security

24th Symposium on Research in Attacks, Intrusions and Defenses
October 6-8, 2021

Cross-Site Request Forgery (CSRF)




```
<script
src="https://bank.com/transfer?amo
unt=2m&to=attacker">
```

Robust anti-CSRF defenses are well-known.

- Custom HTTP Headers
- Hard-to-guess Tokens
- SameSite Cookies
- …

Steep increasing number of reported CSRF instances every year[1]

Are CSRF defenses **implemented correctly** in practice?

[1]**Source:** https://nvd.nist.gov/vuln/search/statistics?results_type=statistics&query=CSRF

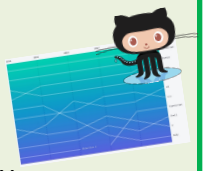
Research Questions

- Little knowledge about CSRF defense implementations in web frameworks.

- **Objective:** Studying CSRF defense implementations
  - **(RQ1)** Existing CSRF defenses? Usage in practice?
  - **(RQ2)** Threats to CSRF defenses and their prevalence?
  - **(RQ3)** Web developers' challenges when using the CSRF defenses?

Approach Overview and Methodology

- **(RQ1)** Existing CSRF defenses? Usage in practice?
  - Systematically surveyed exiting literature, identified **16** defenses
  - Studied defenses' usage in **44** most-popular web frameworks



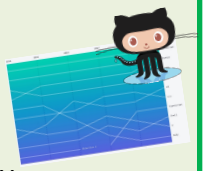
Framework Selection Criteria:
- GitHub Stars, Forks, and Used By
- Downloads (PIP, Packagist, etc)
- StackOverflow questions

- **(RQ2)** Threats to CSRF defenses and their prevalence?
  - Studied academic and non-academic resources, identified **18** threats
  - Detection of security risks by manual code review and dynamic testing

- **(RQ3)** Web developers' challenges when using the CSRF defenses?
  - Documentation and API abstraction review
  - Developer's feedback from vulnerability disclosure

Approach Overview and Methodology

- **(RQ1)** Existing CSRF defenses? Usage in practice?
  - Systematically surveyed exiting literature, identified **16** defenses
  - Studied defenses' usage in **44** most-popular web frameworks



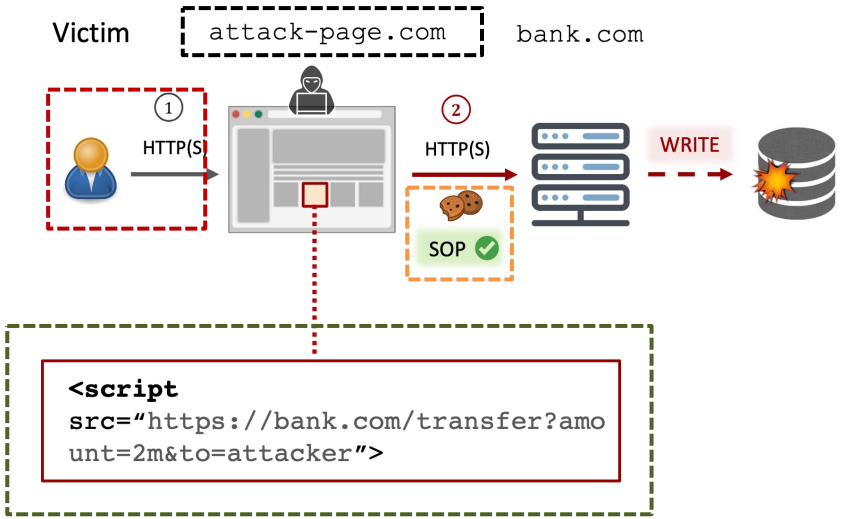
Framework Selection Criteria:
- GitHub Stars, Forks, and Used By
- Downloads (PIP, Packagist, etc)
- StackOverflow questions

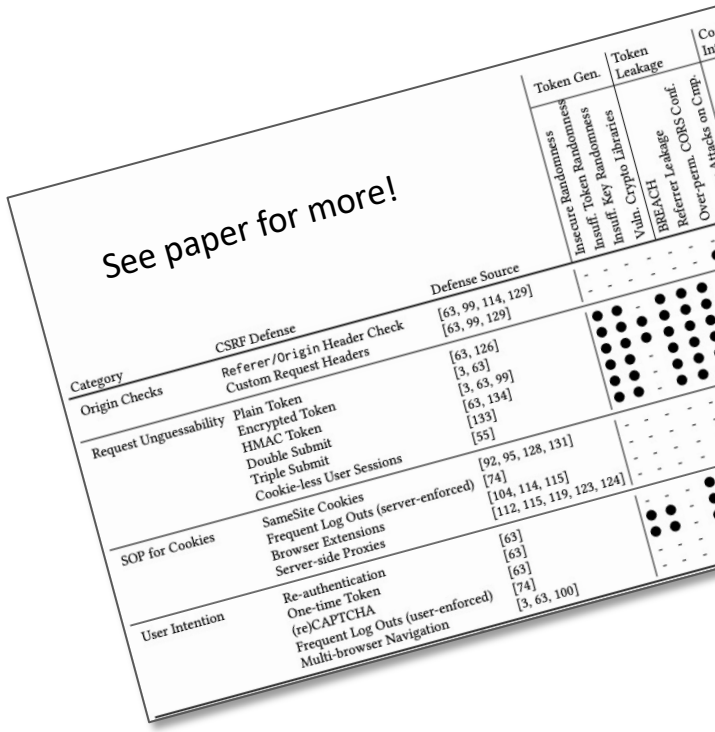
- **(RQ2)** Threats to CSRF defenses and their prevalence?
  - Studied academic and non-academic resources, identified **18** threats
  - Detection of security risks by manual code review and dynamic testing

- **(RQ3)** Web developers' challenges when using the CSRF defenses?
  - Documentation and API abstraction review
  - Developer's feedback from vulnerability disclosure

RQ1: CSRF Defenses

- Comprehensive survey in the literature, idenfitied 16 distinct defense




See paper for more!

Category	CSRF Defense	Defense Source
Origin Checks	Referer/Origin Header Check	[63, 99, 114, 129]
	Custom Request Headers	[63, 99, 129]
Request Unguessability	Plain Token	[63, 126]
	Encrypted Token	[3, 63]
	HMAC Token	[3, 63, 99]
	Double Submit	[63, 134]
	Triple Submit	[133]
	Cookie-less User Sessions	[55]
SOP for Cookies	SameSite Cookies	[92, 95, 128, 131]
	Frequent Log Outs (server-enforced)	[74]
	Browser Extensions	[104, 114, 115]
	Server-side Proxies	[112, 115, 119, 123, 124]
User Intention	Re-authentication	[63]
	One-time Token	[63]
	(re)CAPTCHA	[63]
	Frequent Log Outs (user-enforced)	[74]
	Multi-browser Navigation	[3, 63, 100]

Insecure Randomness	Insuff. Token Randomness	Insuff. Key Libraries	Vuln. Crypto Libraries	BREACH	Referrer Leakage	CORS Conf.	Over-perm.	Attacks on Cmp.	Co In
Token Gen.				Token Leakage					

Origin Checks

Referrer/Origin Check

Custom Request Headers

Req. Unguessability

Plain Token

HMAC Token

Double/Triple Submit

Cookie-less User Sessions

SOP for Cookies

SameSite Cookies

Freq. Log Outs (server)

Browser Extensions

Server-side Proxies

User Intention

Re-authentication

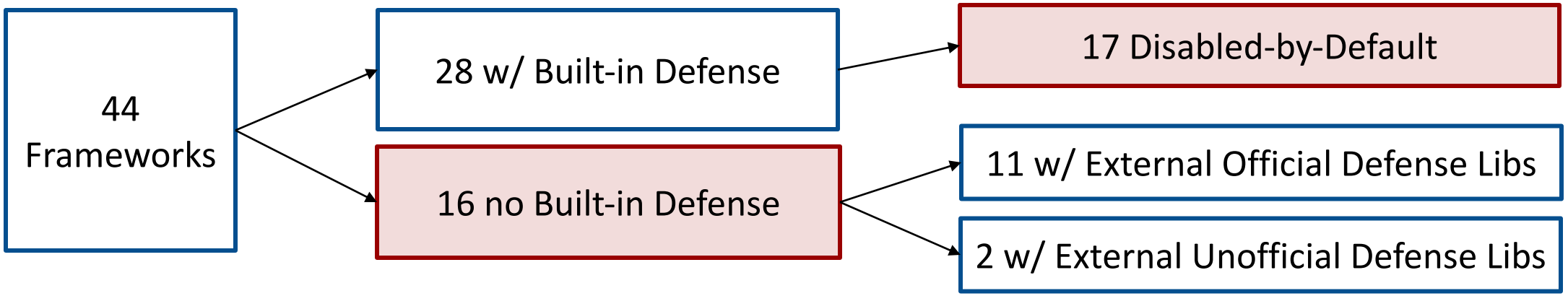
One-Time Token

(re)CAPTCHA

Multi-browser Navigation

RQ1: Demographics of CSRF defenses

- We studied the top 44 frameworks across 5 programming languages.



- **Most popular:** Double Submit Cookie is the most popular defense, followed by Plain Token.
- **Least popular:** Cookie-less user sessions, used only by Meteor framework.

RQ1: Demographics of CSRF defenses (Cont'd)

- **Defense-in-Depth**
  - Web frameworks may implement multiple CSRF defenses at the same time.
  - Almost half of the frameworks (i.e., 19) enforce two or more defenses in sequence.
  - Most common pairs:
    - Double Submit and HMAC Token (12 frameworks)
    - Double Submit and SameSite cookies (6 frameworks)
- **SameSite Cookies**
  - Only 10 frameworks provide built-in support for SameSite cookies

	Ref./Orig. Header	Plain Token	Encrypted Token	HMAC Token	Double Submit	Triple Submit	SameSite Cookies	Cust. Req. Hdr.	Cookie-less Usr Sess.	One-time Token	(re)CAPTCHA	Frequent Log outs	Re-authentication	Browser extensions	Server-side Proxies	Multi-browser Nav.
Ref./Orig. Header	4	0	0	2	3	0	2	0	0	0	0	0	0	0	0	0
Plain Token	0	18	0	0	0	0	3	0	0	0	0	0	0	0	0	0
Encrypted Token	0	0	4	4	4	0	1	0	0	0	0	0	0	0	0	0
HMAC Token	2	0	4	12	12	0	4	0	0	0	0	0	0	0	0	0
Double Submit	3	0	4	12	22	0	6	0	0	0	0	0	0	0	0	0
Triple Submit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SameSite Cookies	2	3	1	4	6	0	10	0	0	0	0	0	0	0	0	0
Cust. Req. Hdr.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cookie-less Usr Sess.	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
One-time Token	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(re)CAPTCHA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Frequent Log Outs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Re-authentication	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Browser Extensions	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Server-side Proxies	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Multi-browser Nav.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Approach Overview and Methodology

- **(RQ1)** Existing CSRF defenses? Usage in practice?
  - Systematically surveyed exiting literature, identified **16** defenses
  - Studied defenses' usage in **44** most-popular web frameworks

Framework Selection Criteria:
- GitHub Stars, Forks, and Used By
- Downloads (PIP, Packagist, etc)
- StackOverflow questions

- **(RQ2)** Threats to CSRF defenses and their prevalence?
  - Studied academic and non-academic resources, identified **18** threats
  - Detection of security risks by manual code review and dynamic testing

- **(RQ3)** Web developers' challenges when using the CSRF defenses?
  - Documentation and API abstraction review
  - Developer's feedback from vulnerability disclosure

RQ2: Security Risks

- **Identified 157 security risks in 37 frameworks**
  - Directly exploitable: 17
  - Conditional exploitability: 140

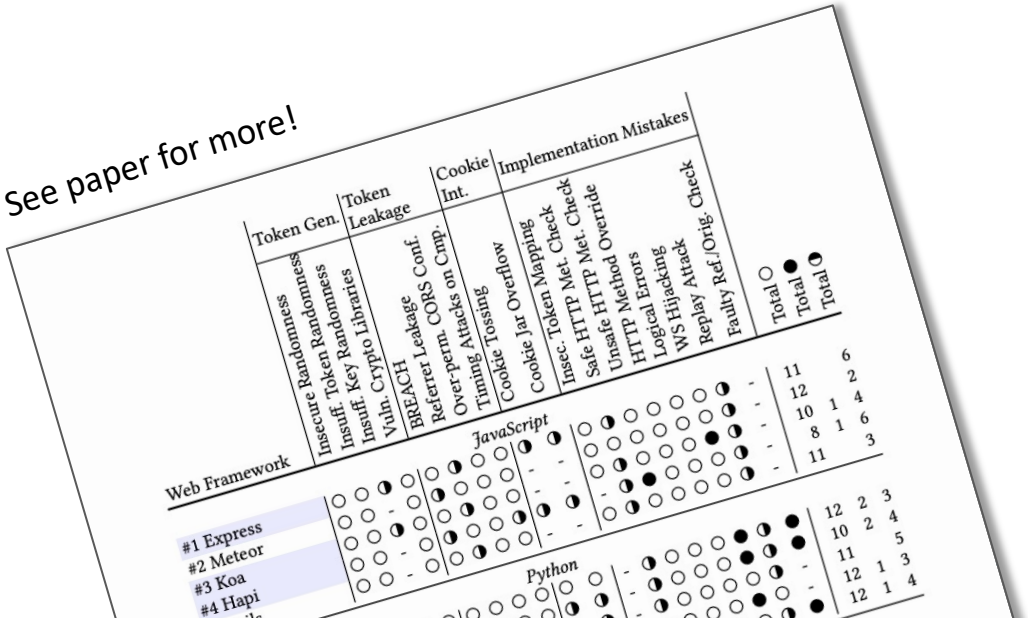
- Most common: implementation mistakes
- Least common: cookie integrity
- Haven't found any: weak token generators

In this presentation

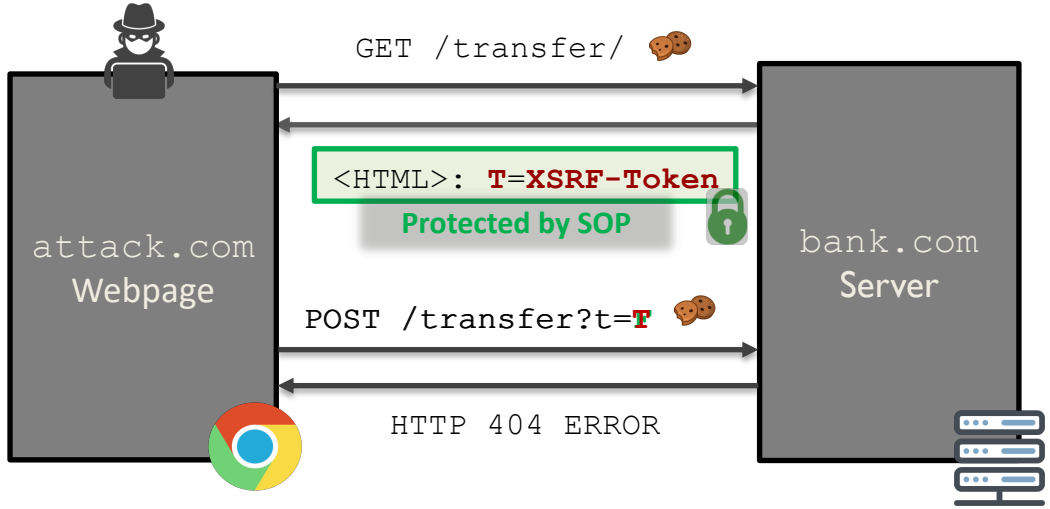
Threats	# Vuln.	# Exploitable
Token Gen.	10	0
Token Leakage	37	0
Cookie Integrity	30	0
Impl. Mistake	80	17
Total	157	17

89% (35/39) of frameworks with a CSRF defense are vulnerable to at least one threat.

See paper for more!

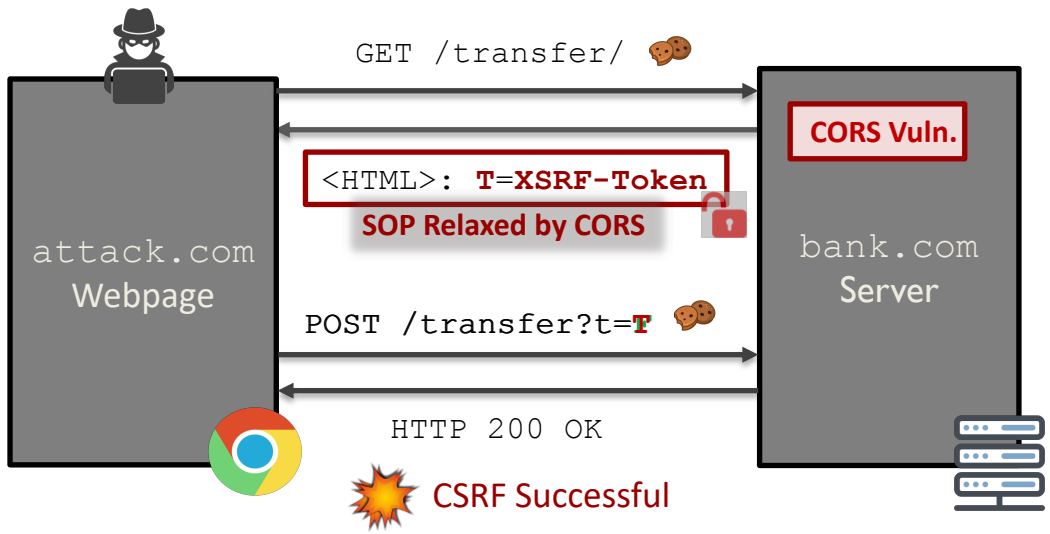


RQ2: Token Leakage



RQ2: Token Leakage

- CSRF tokens can be leaked as a result of:
  - CORS misconfigurations
  - Cross-domain referrer leakage
  - Side-channel attacks affecting CSRF token comparison



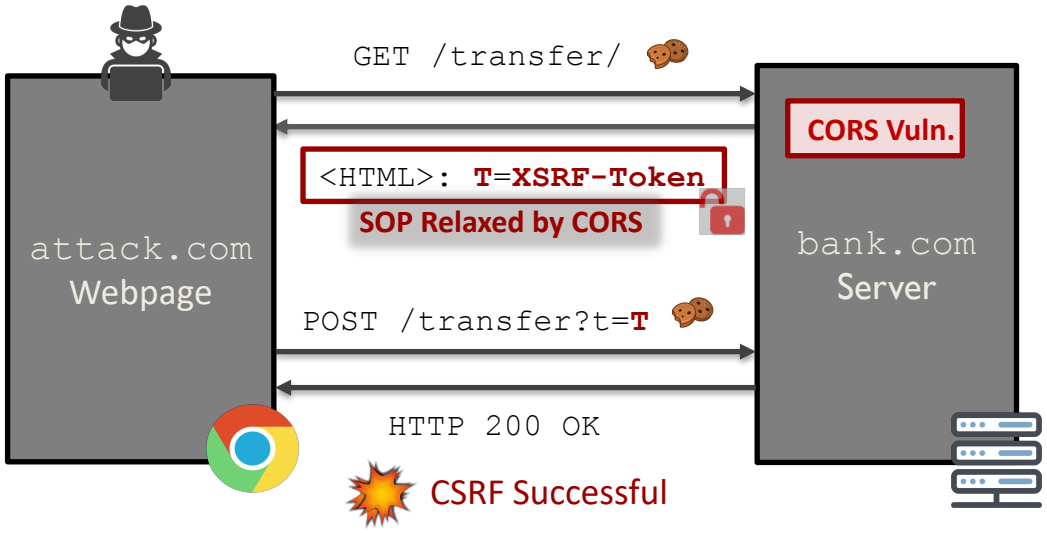
- Security risks
  - Identified instances of each of the three above-mentioned threats

A total of 37 token leakage vulnerabilities affecting 34 frameworks.

Token Leakage Example: CORS Misconfiguration In Play Framework

- Vulnerability in CORS module (when enabled)
  - *Access-Control-Allow-Origin* response header: reflects origin.
  - *Access-Control-Allow-Credentials* response header: set to true by default

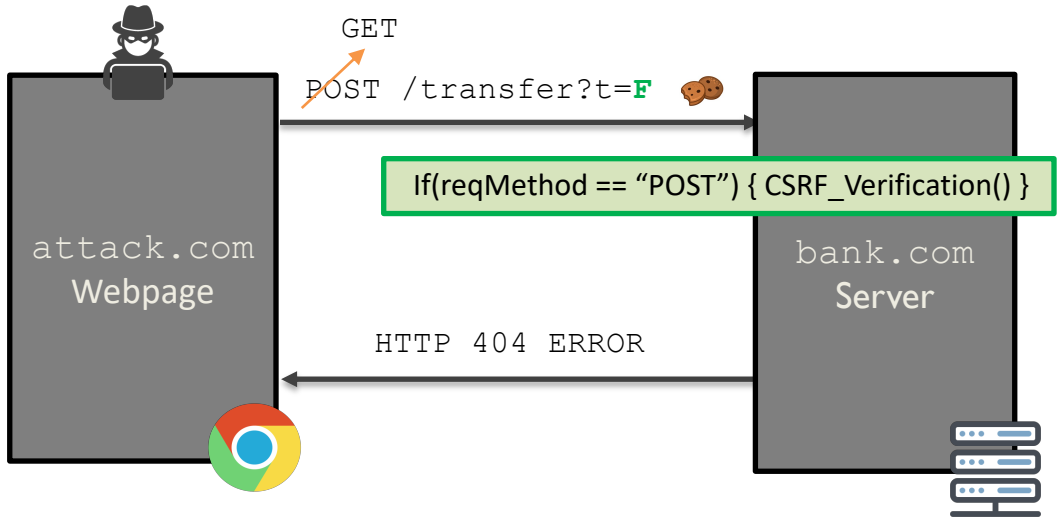
- Exploitation
  1. GET request to retrieve a webpage with a valid CSRF token
  2. Use the token in the actual state-changing request



Play Framework: reference.conf, CORS config

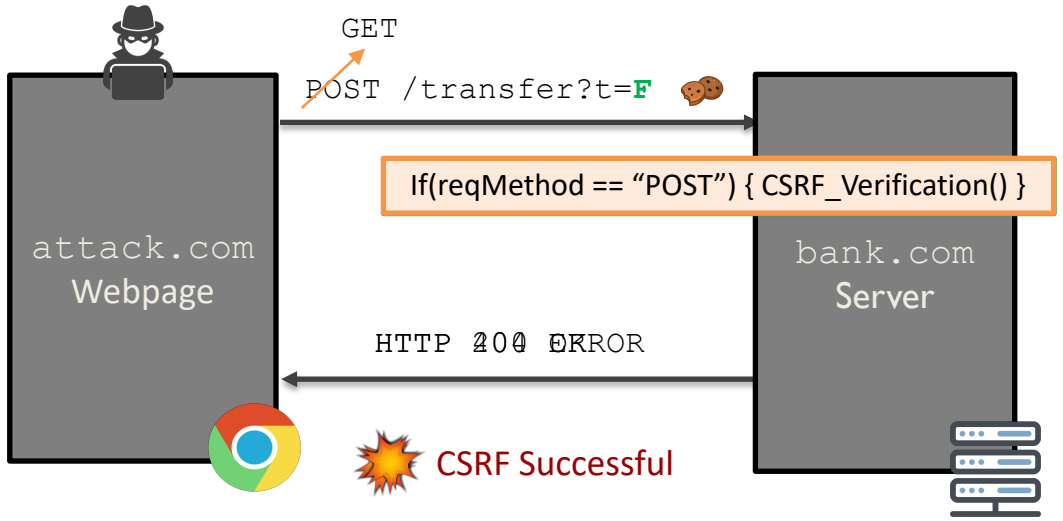
```
# CORS filter configuration
cors {

    # The allowed origins.
    # If null, all origins are allowed
    allowedOrigins= null

    # The allowed HTTP methods.
    # If null, all methods are allowed
    allowedHttpMethods= null

    # The allowed HTTP headers.
    # If null, all headers are allowed
    allowedHttpHeaders= null

    # The exposed headers
    exposedHeaders = []

    # Whether to support credentials
    supportsCredentials = true
}
```

RQ2: Implementation Mistakes



RQ2: Implementation Mistakes

- Mistakes during CSRF verification
  - Missing CSRF checks on HTTP methods
  - Logical mistakes
  - Reusable tokens (i.e., replay attacks)
  - Cookie-based authentication for WebSockets



Note: GET-based state-changing requests are still frequently used in practice [Khodayari et. al., S&P 2022]

⚠ A total of 80 implementation mistakes affecting 37 frameworks.

Example: CakePHP Critical Vulnerability (CVE-2020-35239)

- Vulnerability in CSRF verification process

1. CSRF verification is performed only on unsafe HTTP request methods

CakePHP Framework: CsrfProtectionMiddleware.php, process() function

```
// check if the request method is an unsafe, or has a body
$hasData = in_array($method, ['PUT', 'POST', 'DELETE', 'PATCH'], true)
if ($hasData) {
    // function that compares the token in the request against the token in the CSRF cookie.
    $this->_validateToken($request);
    $request = $this->_unsetTokenField($request); // removes the CSRF token from the request's body.
}
```

2. HTTP Method Override: change the request method → CRSF verification will not be performed

```
<input id="_method" type="hidden" value="any" />
```

3. route() module vulnerability: no check on input string to be a valid HTTP request method

Example: Vert.x-Web Critical Vulnerability (CVE-2020-35217)

- Vulnerability in CSRF verification process
  1. Generated token is stored in three places:
     - (i) HTML form, (ii) CSRF cookie, and (iii) Server-side user session object.
  2. On a state-changing request
     - Compares token in CSRF cookie vs token in session object.

Vulnerability: token in HTML form is ignored.

Vulnerable as victim's cookies are always sent automatically during a CSRF attack.

Vert.x-Web: CSRFHandlerImpl.java, validateRequest() function

```
// gets the CSRF cookie from the request
final Cookie cookie = ctx.getCookie(cookieName);

// get the CSRF token from session storage
String challenge = getTokenFromSession(ctx);

// compare
if (challenge == null || !challenge.equals(cookie.getValue())) {
  return false;
}
```

Approach Overview and Methodology

- **(RQ1)** Existing CSRF defenses? Usage in practice?
  - Systematically surveyed exiting literature, identified **16** defenses
  - Studied defenses' usage in **44** most-popular web frameworks

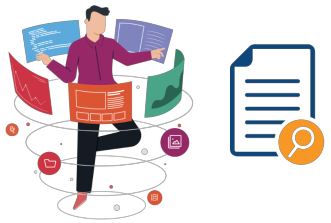
Framework Selection Criteria:
- GitHub Stars, Forks, and Used By
- Downloads (PIP, Packagist, etc)
- StackOverflow questions

- **(RQ2)** Threats to CSRF defenses and their prevalence?
  - Studied academic and non-academic resources, identified **18** threats
  - Detection of security risks by manual code review and dynamic testing

- **(RQ3)** Web developers' challenges when using the CSRF defenses?
  - Documentation and API abstraction review
  - Developer's feedback from vulnerability disclosure

RQ3: Documentation Review

- Incorrect use of implemented CSRF defenses can also compromise webapps' security
- Review of CSRF documentation based on six quality criteria



1. Defense name and/or description
2. Cryptographic Guarantees
3. API specification
4. Code Example
5. Configuration
6. General Security Considerations

- Findings
  - 4.5% of the frameworks included information for all six criteria 
  - 29.6% frameworks fulfill only one of the six criteria 
  - 61.7% of the frameworks do not fulfill half of the documentation quality criteria 

The state of the documentation of web frameworks is far from adequate.

RQ3: API Abstraction Analysis

- Incorrect use of defenses can also arise from variety in the semantics of APIs
- The majority of frameworks (i.e., 39) implement token-based defenses
- **API specification Analysis:** Semantics and operations of APIs diverge. For example:

Token Generation

1. Token generation function call
2. Template engine pseduo-variables
3. Framework special form objects

Token validation

1. CSRF verification function call
2. Method decorators
3. Automatic

No established consensus in the way unguessable request defenses are exposed to developers

RQ3: Developers‘ Feedback

- All 157 vulnerabilities reported
  - While disclosing, we learned interesting aspects about CSRF defenses

- **Observation:** A same threat was patched in some frameworks but not in others.
  - Example 1: Replay attacks, patched by Vert.x-Web and Slim but not by Spring and Django (risk acceptance)
  - Example 2: BREACH, patched by CakePHP and Vert.x-Web but not by Apache Struts (webapp vs framework’s responsibility)

What is a threat?

Who should fix ?

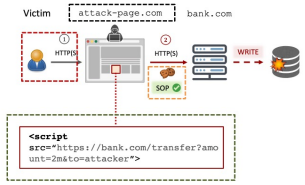
Inconsistent threat model + divergent expectations on who is responsible to fix

Conclusion

Thank You!

RQ1: CSRF Defenses

- Comprehensive survey in the literature, idenfitied 16 distinct defense



Origin Checks	Req. Unguessability	SOP for Cookies	User Intention
Referrer/Origin Check	Plain Token	SameSite Cookies	Re-authentication
Custom Request Headers	HMAC Token	Freq. Log Outs (server)	One-Time Token
	Double/Triple Submit	Browser Extensions	(re)CAPTCHA
	Cookie-less User Sessions	Server-side Proxies	Multi-browser Navigation

RQ2: Security Risks

- **Identified 157 security risks in 37 frameworks**
  - Directly exploitable: 17
  - Conditional exploitability: 140
- Most common: implementation mistakes
- Least common: cookie integrity
- Haven't found any: weak token generators

In this presentation

Threats	# Vuln.	# Exploitable
Token Gen.	10	0
Token Leakage	37	0
Cookie Integrity	30	0
Impl. Mistake	80	17
Total	157	17

89% (35/39) of frameworks with a CSRF defense are vulnerable to at least one threat.

See paper for more!

RQ3: Developers' Challenges

- Incorrect use of implemented CSRF defenses can also compromise webapps' security
- Review of CSRF documentation based on six quality criteria
  1. Defense name and/or description
  2. Cryptographic Guarantees
  3. API specification
  4. Code Example
  5. Configuration
  6. General Security Considerations
- Findings
  - 4.5% of the frameworks included information for all six criteria
  - 29.6% frameworks fulfill only one of the six criteria
  - 61.7% of the frameworks do not fulfill half of the documentation quality criteria

The state of the documentation of web frameworks is far from adequate.

RQ3: Developers' Feedback

- All 157 vulnerabilities reported
  - While disclosing, we learned interesting aspects about CSRF defenses
- **Observation:** A same threat was patched in some frameworks but not in others.
  - Example 1: Replay attacks, patched by Vert.x-Web and Slim but not by Spring and Django (risk acceptance)
  - Example 2: BREACH, patched by CakePHP and Vert.x-Web but not by Apache Struts (webapp vs framework's responsibility)

What is a threat?

Who should fix ?

Inconsistent threat model + divergent expectations on who is responsible to fix

@Soheil__K