SENG 360 - Security Engineering
Web Security - CSRF

Jens Weber

Fall 2021

University of Victoria

Recall from last classes

2021 List

Rank	ID	Name	Score	2020 Rank Change
[1]	CWE-787	Out-of-bounds Write	65.93	+1
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[3]	CWE-125	Out-of-bounds Read	24.9	+1
[4]	CWE-20	Improper Input Validation	20.47	-1
[5]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[7]	CWE-416	Use After Free	16.83	+1
[8]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	CWE-434	Unrestricted Upload of File with Dangerous Type	8.45	+5
[11]	CWE-306	Missing Authentication for Critical Function	7.93	+13
[12]	CWE-190	Integer Overflow or Wraparound	7.12	-1
[13]	CWE-502	Deserialization of Untrusted Data	6.71	+8

Recall from last classes

XSS attacks

➡ *Exploit the fact that the client trusts the Web site*

CSRF attacks

➡ *Exploit the fact that the Web site trusts the client*

University of Victoria

Learning Objectives

At the end of this class you will be able to

- Describe cross-site request forgery vulnerabilities and mitigations

University of Victoria

What is Cross-Site Request Forgery (CSRF)?

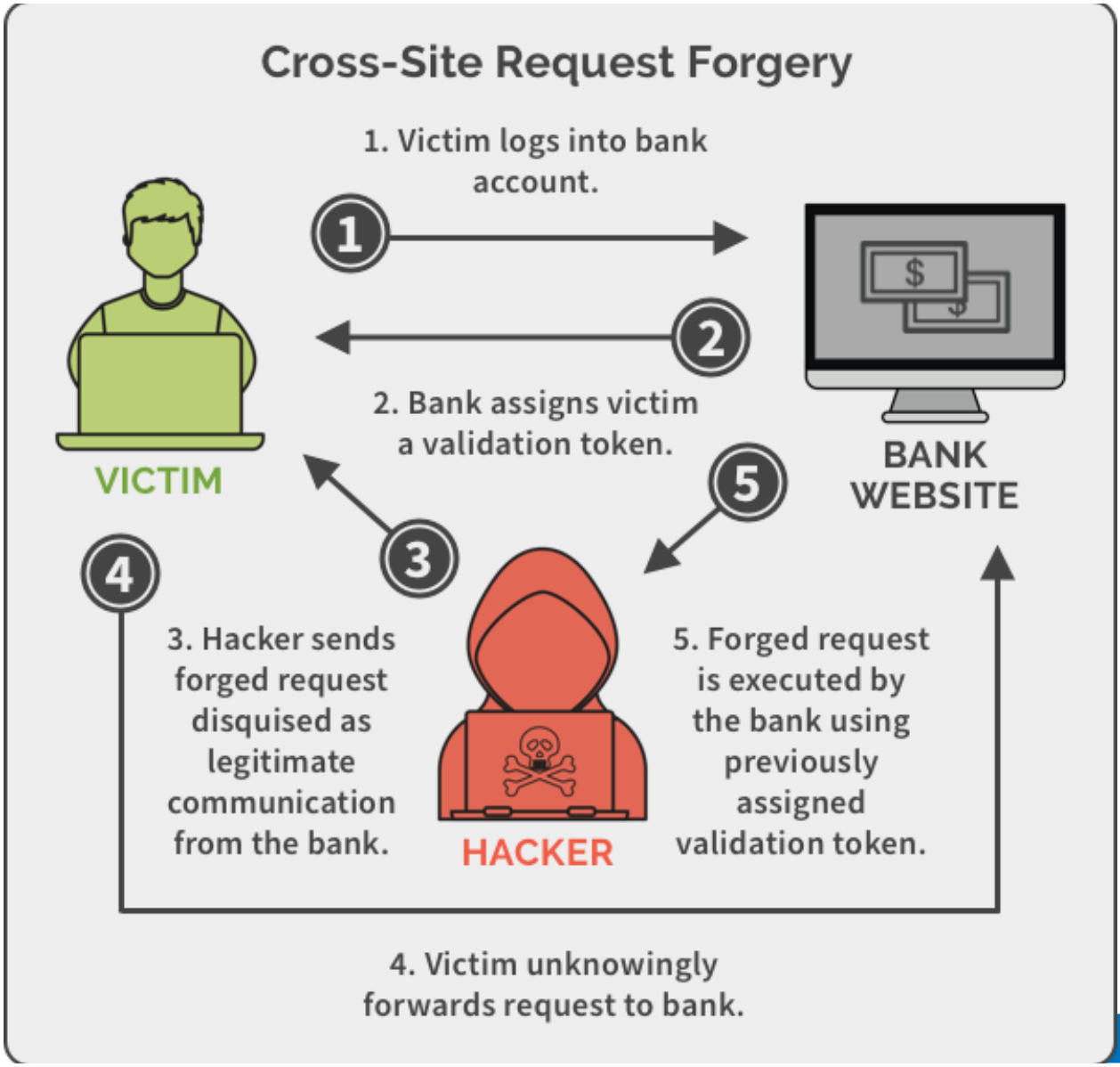
In a CSRF attack, an innocent end user is tricked by an attacker into submitting a web request that they did not intend. [Wikipedia]

University of Victoria

How CSRF works

GET Request Example:

```
<a href="http://bank.com/transfer.do?
acct=MARIA&amount=100000">View my Pictures!</a>
```



also works with POST

```
POST http://bank.com/transfer.do HTTP/1.1

acct=BOB&amount=100
```

```
<form action="http://bank.com/transfer.do" method="POST">

<input type="hidden" name="acct" value="MARIA"/>
<input type="hidden" name="amount" value="100000"/>
<input type="submit" value="View my pictures"/>

</form>
```

```
<body onload="document.forms[0].submit()">

<form...
```

University of Victoria

may also work with PUT and DELETE

```
PUT http://bank.com/transfer.do HTTP/1.1

{ "acct":"BOB", "amount":100 }
```

```
<script>
function put() {
    var x = new XMLHttpRequest();
    x.open("PUT","http://bank.com/transfer.do",true);
    x.setRequestHeader("Content-Type", "application/json");
    x.send(JSON.stringify({"acct":"BOB", "amount":100}));
}
</script>

<body onload="put()">
```

Real world CSRF attack

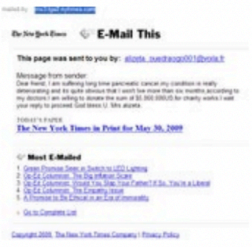
ZDNet | BEST VPNS | CLOUD | SECURITY | AI | ZDNET RECOMMENDS | MORE | EDITION: US | NEWSLETTERS | ALL WRITERS

MUST READ: Sorry, Gen Z isn't going to solve your tech skills crisis

419 scammers using NYTimes.com 'email this feature'

What do Burkina Faso and the New York Times have in common? As of recently, a peak of 419 scams promising you the Moon and asking you for advance-fees via emails sent through the NYTimes.

By Dancho Danchev for Zero Day | June 3, 2009 | Topic: Collaboration



What do Burkina Faso and the New York Times have in common? As of recently, a peak of 419 scams promising you the Moon and asking you for advance-fees via emails sent through the NYTimes.com's 'email this feature' in order to successfully bypass anti-spam filters.

RELATED

3 ways robots won in 2021
Robotics

Tangoe shows work-from-anywhere expense-management package
Collaboration

Microsoft introduces Loop: A

University of Victoria

NY Times Vuln

NYT allowed GET as well as PUT requests

Attacker could easily convert form to GET request

```
<form
action="http://www.nytimes.com/mem/emailthis.html"
method="POST"
enctype="application/x-www-form-urlencoded">
<input type="checkbox" id="copytoself"
name="copytoself" value="Y">
<input id="recipients" name="recipients"
type="text" maxlength="1320" value="">
<input type="hidden" name="state" value="1">
<textarea id="message" name="personalnote"
maxlength="512"></textarea>
<input type="hidden" name="type" value="1">
<input type="hidden" name="url"
value="[...]">
```

CSRF Vuln at ING Direct

1. The attacker creates a checking account

(a) The attacker causes the user's browser to visit ING's "Open New Account" page:

- A GET request to `https://secure.ingdirect.com/myaccount/INGDirect.html?command=gotoOpenOCA`

University of Victoria

CSRF Vuln at ING Direct

1. The attacker creates a checking account

(b) The attacker causes the user's browser to choose a "single" account:

- A POST request to `https://secure.ingdirect.com/myaccount/INGDirect.html` with the parameters:

```
command=ocaOpenInitial&YES, I
WANT TO CONTINUE..x=44&YES, I
WANT TO CONTINUE..y=25
```

University of Victoria

CSRF Vuln at ING Direct

1. The attacker creates a checking account

(c) The attacker chooses an arbitrary amount of money to initially transfer from the user's savings account to the new, fraudulent account:

- A POST request to `https://secure.ingdirect.com/myaccount/INGDirect.html` with the parameters:

```
command=ocaValidateFunding&PRIMARY
CARD=true&JOINTCARD=true&Account
Nickname=[ACCOUNT_NAME]&FROMACCT=
0&TAMT=[INITIAL_AMOUNT]&YES, I
WANT TO CONTINUE..x=44&YES, I
WANT TO
CONTINUE..y=25&XTYPE=4000USD
&XBCRCD=USD
```

University of Victoria

CSRF Vuln at ING Direct

1. The attacker creates a checking account

(d) The attacker causes the user's browser to click the final "Open Account" button, causing ING to open a new checking account on behalf of the user:

- A POST request to `https://secure.ingdirect.com/myaccount/INGDirect.html` with the parameters:

```
command=ocaOpenAccount&Agree
ElectronicDisclosure=yes&AgreeTerms
Conditions=yes&YES, I WANT TO
CONTINUE..x=44&YES, I WANT TO
CONTINUE..y=25&YES, I WANT TO
CONTINUE.=Submit
```

University of Victoria

CSRF Vuln at ING Direct

2. The attacker adds himself as a payee to the user's account

(a) The attacker causes the user's browser to visit ING's "Add Person" page:

- A GET request to `https://secure.ingdirect.com/myaccount/INGDirect.html?command=goToModifyPersonalPayee&Mode=Add&from=displayEmailMoney`

University of Victoria

CSRF Vuln at ING Direct

2. The attacker adds himself as a payee to the user's account

(b) The attacker causes the user's browser to enter the attacker's information:

- A POST request to `https://secure.ingdirect.com/myaccount/INGDirect.html` with the parameters:

```
command=validateModifyPersonalPayee
&from=displayEmailMoney&PayeeName
=[PAYEE_NAME]&PayeeNickname=&chk
Email=on&PayeeEmail=[PAYEE_EMAIL]
&PayeeIsEmailToOrange=true&Payee
OrangeAccount=[PAYEE_ACCOUNT_NUM]&
YES, I WANT TO CONTINUE..x=44
&YES, I WANT TO CONTINUE..y=25
```

University of Victoria

CSRF Vuln at ING Direct

2. The attacker adds himself as a payee to the user's account

(c) The attacker causes the user's browser to confirm that the attacker is a valid payee:

- A POST request to `https://secure.ingdirect.com/myaccount/INGDirect.html` with the parameters:

```
command=modifyPersonalPayee&from=
displayEmailMoney&YES, I WANT TO
CONTINUE..x=44 &YES, I WANT TO
CONTINUE..y=25
```

University of Victoria

CSRF Vuln at ING Direct

3. The attacker transfers funds from the user's account to his own account

(a) The attacker causes the user's browser to enter an amount of money to send to the attacker:

- A POST request to `https://secure.ingdirect.com/myaccount/INGDirect.html` with the parameters:

```
command=validateEmailMoney&CNSPayID
=5000&Amount=[TRANSFER_AMOUNT]
&Comments=[TRANSFER_MESSAGE]&YES,
I WANT TO CONTINUE..x=44 &YES, I
WANT TO
CONTINUE..y=25&show=1&button=Send
Money
```

University of Victoria

CSRF Vuln at ING Direct

3. The attacker transfers funds from the user's account to his own account

(b) The attacker causes the user's browser to confirm that the money should be sent:

- A POST request to `https://secure.ingdirect.com/myaccount/INGDirect.html` with the parameters:

```
command=emailMoney&Amount=
[TRANSFER_AMOUNT]Comments=
[TRANSFER_MESSAGE]&YES, I WANT
TO CONTINUE..x=44&YES, I WANT TO
CONTINUE..y=25
```

University of Victoria

https://youtu.be/5joX1skQtVE

Mitigations against CSRF

The **Synchronizer Token Pattern**

- CSRF tokens (think nonces) generated by server
  - per user session (less secure)
  - per request (more secure)
- communicate secretly (e.g, hidden fields, custom HTTP request headers) (**not** in cookies or URL)

```
<form action="/transfer.do" method="post">
<input type="hidden" name="CSRFToken" value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWEv
[...]
</form>
```

Mitigations against CSRF

The **Double Submit Cookie**

- Server generates nonce and stores in separate cookie (even before authentication)
- Client sends nonce in hidden field for each request
- Server compares the two

Use if Synchronizer Token Pattern is problematic - and stateless is needed

Defense in Depth Techniques

***SameSite* Cookie Attribute**

- defines whether browser sends cookies along with cross-site requests
- Possible values
  - ***Strict*** - no cookies sent from a cross-site context
  - ***Lax*** - cookies sent only for “safe” requests (e.g., no state change)
  - ***None***

Defense in Depth Techniques

not an effective mitigation by themselves

SameSite Cookie Attribute

- defines whether browser sends cookies along with cross-site requests
- Possible values
  - ***Strict*** - no cookies sent from a cross-site context
  - ***Lax*** - cookies sent only for “safe” requests (e.g., no state change)
  - ***None***

Defense in Depth Techniques

not an effective mitigation by themselves

Verify Origin With Standard Headers

- Determine origin of request
- Determine target of request
- Verify that they are the same

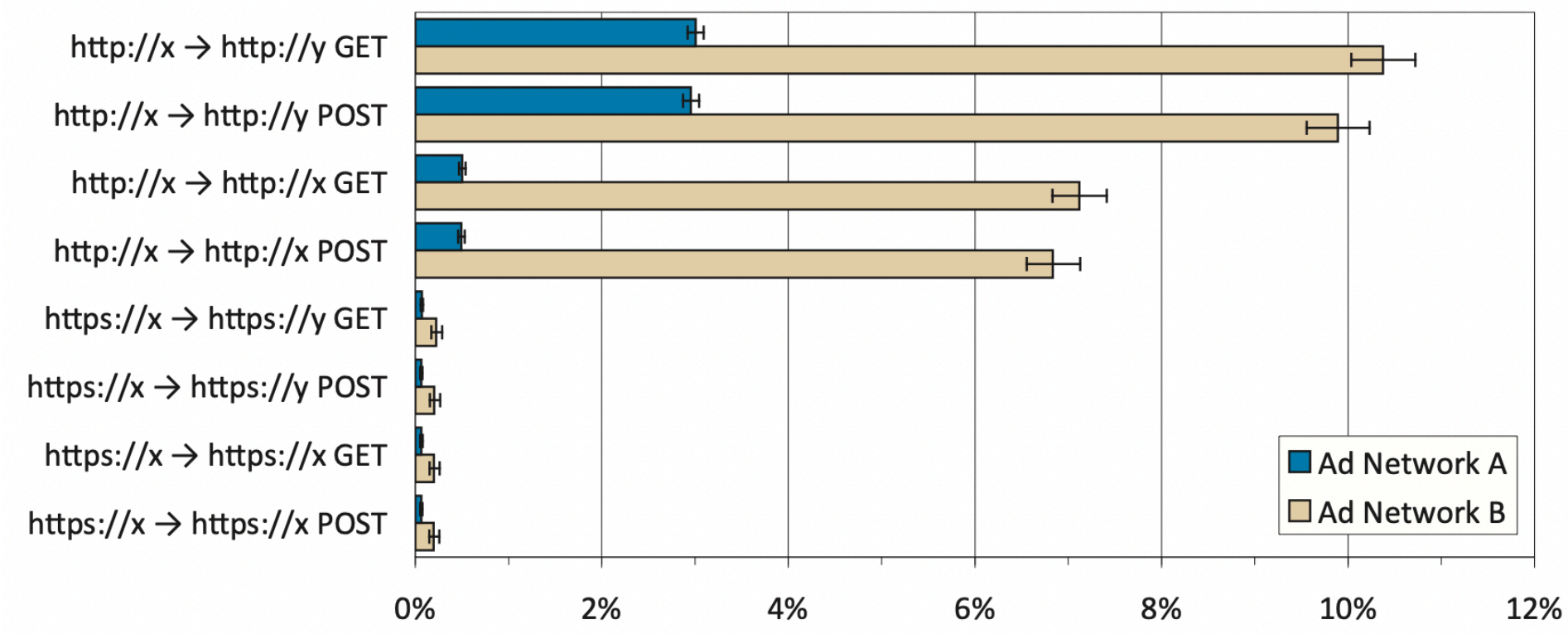
Some privacy contexts may make this difficult

Defense in Depth Techniques

not an effective mitigation by themselves

Use a Custom Request Header

- Particularly well suited for AJAX or API endpoints
- Only JavaScript can be used to add custom headers, and only within its origin



https://seclab.stanford.edu/websec/csrf/csrf.pdf

Defense in Depth Techniques

not an effective mitigation by themselves

User Interaction Based CSRF Defense

- Re-authenticate (password or stronger)
- One-time Token
- CAPTCHA (prefer newer approaches)
- Separate browser for high-assurance apps
- timed log outs
- train users to close tabs :-)

Summary and Outlook

- CSRF vulnerabilities exploit trust of server in client
- Mitigations (CSRF tokens) built into many Web frameworks today
- Defenses in depth are helpful but not by themselves secure

Next class: more high profile Web vulnerabilities

University of Victoria

Questions?

University of Victoria