XSS and UI Attacks

CS 161 Fall 2022 - Lecture 14

Announcements

- Congratulations on finishing the midterm!
  - Midterm grades and solutions will be released by the end of the week
- Homework 4 will be released tonight and is due **Friday, October 21st**
  - No lab on this one :)
- Project 2 will be released tonight
  - The design doc checkpoint is due **Friday, October 28th**
  - The final design doc and code are due **Friday, November 11th**
  - *Start this project early!*

Last Time: Cookies

- Cookie: a piece of data used to maintain state across multiple requests
  - Set by the browser or server
  - Stored by the browser
  - Attributes: Name, value, domain, path, secure, HttpOnly, expires
- Cookie policy
  - Server with domain X can set a cookie with domain attribute Y if the domain attribute is a **domain suffix** of the server's domain, and the domain attribute Y is not a top-level domain (TLD)
  - The browser attaches a cookie on a request if the domain attribute is a **domain suffix** of the server's domain, and the path attribute is a **prefix** of the server's path

Last Time: Session Authentication

- Session authentication
  - Use cookies to associate requests with an authenticated user
  - First request: Enter username and password, receive session token (as a cookie)
  - Future requests: Browser automatically attaches the session token cookie
- Session tokens
  - If an attacker steals your session token, they can log in as you
  - Should be randomly and securely generated by the server
  - The browser should not send tokens to the wrong place

Last Time: CSRF

- Cross-site request forgery (CSRF or XSRF): An attack that exploits cookie-based authentication to perform an action as the victim
  - User authenticates to the server
    - User receives a cookie with a valid session token
  - Attacker tricks the victim into making a malicious request to the server
  - The server accepts the malicious request from the victim
    - Recall: The cookie is automatically attached in the request
- Attacker must trick the victim into creating a request
  - GET request: click on a link
  - POST request: use JavaScript

Last Time: CSRF Defenses

- CSRF token: A secret value provided by the server to the user. The user must attach the same value in the request for the server to accept the request.
  - The attacker does not know the token when tricking the user into making a request
- Referer Header: Allow same-site requests, but disallow cross-site requests
  - Header may be blank or removed for privacy reasons
- Same-site cookie attribute: The cookie is sent only when the domain of the cookie exactly matches the domain of the origin
  - Not implemented on all browsers

Today: XSS

- XSS
  - Websites use untrusted content as control data
  - Stored XSS
  - Reflected XSS
  - Defense: HTML sanitization
  - Defense: Content Security Policy (CSP)
- UI attacks
  - Clickjacking
  - Phishing

Cross-Site Scripting (XSS)

Textbook Chapter 22

Top 25 Most Dangerous Software Weaknesses (2020)

Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
[16]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
[17]	CWE-94	Improper Control of Generation of Code ('Code Injection')	6.53

Review: Same-Origin Policy

- Two webpages with different origins should not be able to access each other's resources
  - Example: JavaScript on **`http://evil.com`** cannot access the information on **`http://bank.com`**

Review: JavaScript

- **JavaScript**: A programming language for running code in the web browser
- JavaScript is **client-side**
    - Code sent by the server as part of the response
    - Runs in the browser, not the web server!
- Used to manipulate web pages (HTML and CSS)
    - Makes modern websites interactive
    - JavaScript can be directly embedded in HTML with **`<script>`** tags
- Most modern webpages involve JavaScript
    - JavaScript is supported by all modern web browsers
- You don’t need to know JavaScript syntax
    - However, knowing common attack functions helps

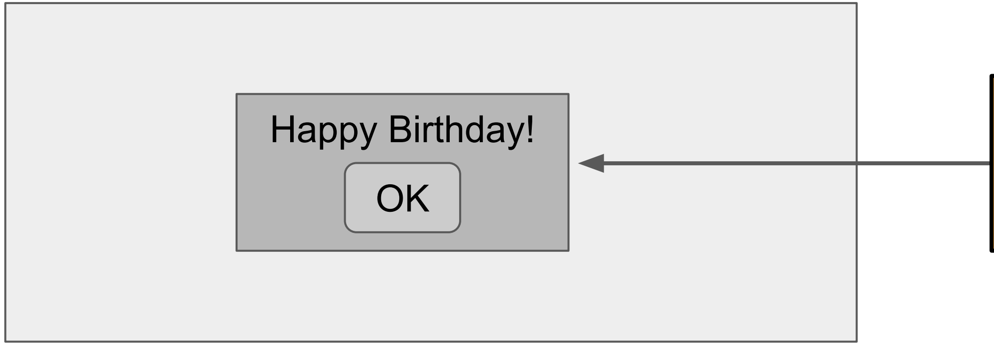
Review: JavaScript

- JavaScript can create a pop-up message

HTML (with embedded JavaScript)

```
<script>alert("Happy Birthday!")</script>
```

Webpage



When the browser loads this HTML, it will run the embedded JavaScript and cause a pop-up to appear.

A Go HTTP Handler

Handler

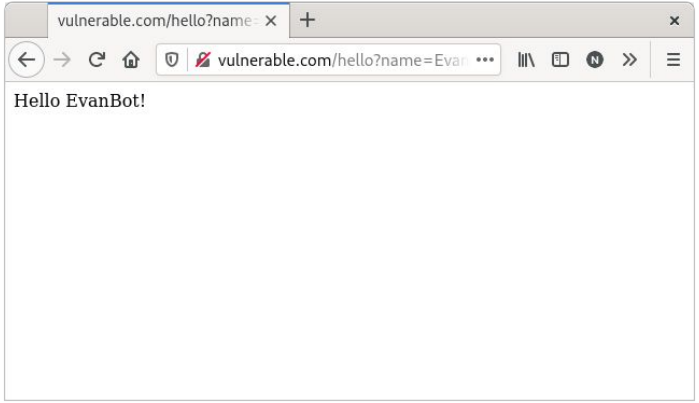
```
func handleSayHello(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query()["name"][0]
    fmt.Fprintf(w, "<html><body>Hello %s!</body></html>", name)
}
```

URL

```
https://vulnerable.com/hello?name=EvanBot
```

Response

```
<html><body>Hello EvanBot!</body></html>
```



Hello EvanBot!

A Go HTTP Handler

Handler

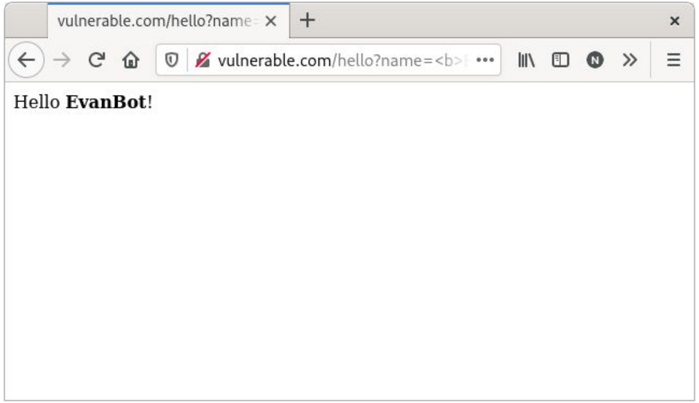
```
func handleSayHello(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query()["name"][0]
    fmt.Fprintf(w, "<html><body>Hello %s!</body></html>", name)
}
```

URL

```
https://vulnerable.com/hello?name=<b>EvanBot</b>
```

Response

```
<html><body>Hello <b>EvanBot</b>!</body></html>
```



Hello **EvanBot**!

A Go HTTP Handler

Problem: This input represents control data (HTML), not just text!

Handler

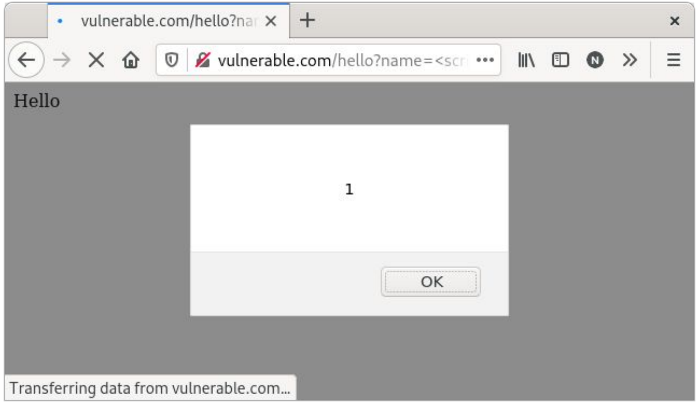
```
func handleSayHello(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query()["name"][0]
    fmt.Fprintf(w, "<html><body>Hello %s!</body></html>", name)
}
```

URL

```
https://vulnerable.com/hello?name=<script>alert(1)</script>
```

Response

```
<html><body>Hello <script>alert(1)</script>!</body></html>
```



vulnerable.com/hello?na
vulnerable.com/hello?name=<scr
Hello
1
OK
Transferring data from vulnerable.com...

A Go HTTP Handler

Not just %s: It can happen with any string manipulation

Handler

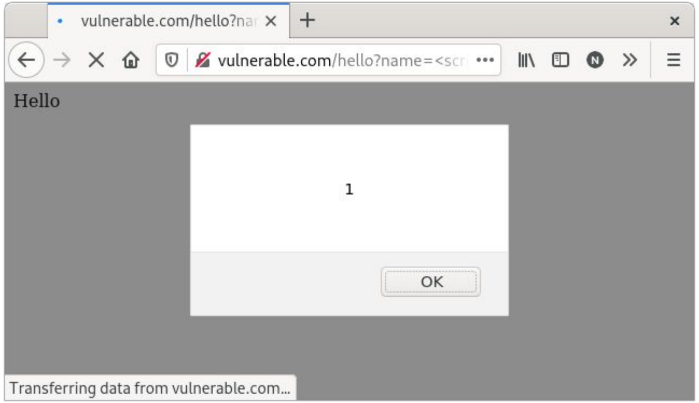
```
func handleSayHello(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query()["name"][0]
    content := "<html><body>Hello "+name+"!</body></html>"
    fmt.Fprint(w, content)
}
```

URL

```
https://vulnerable.com/hello?name=<script>alert(1)</script>
```

Response

```
<html><body>Hello <script>alert(1)</script>!</body></html>
```


vulnerable.com/hello?na
vulnerable.com/hello?name=<scr
Hello
1
OK
Transferring data from vulnerable.com...

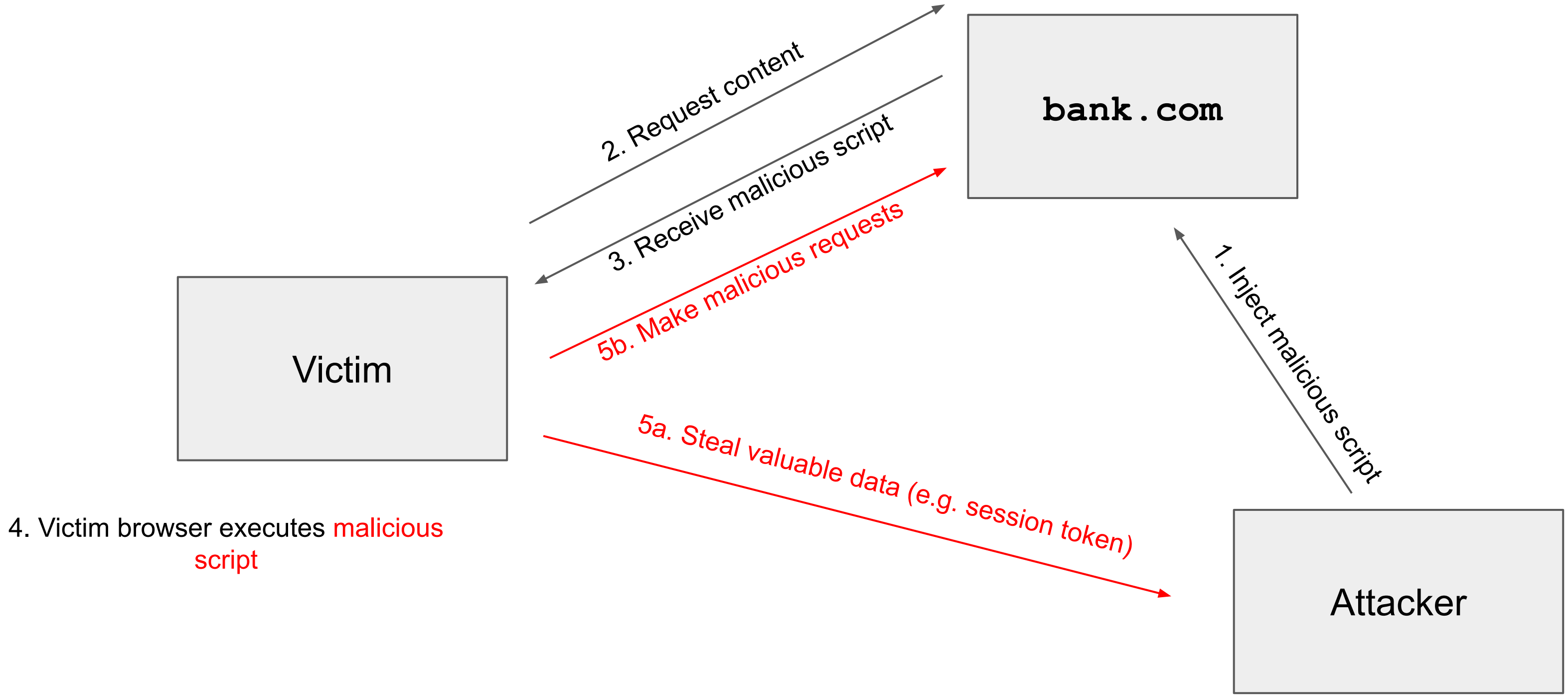
Cross-Site Scripting (XSS)

- Idea: The attacker adds malicious JavaScript to a legitimate website
  - The legitimate website will send the attacker’s JavaScript to browsers
  - The attacker’s JavaScript will run with the origin of the legitimate website
  - Now the attacker’s JavaScript can access information on the legitimate website!
- **Cross-site scripting** (**XSS**): Injecting JavaScript into websites that are viewed by other users
  - Cross-site scripting subverts the same-origin policy
- Two main types of XSS
  - Stored XSS
  - Reflected XSS

Stored XSS

- **Stored XSS** (**persistent XSS**): The attacker's JavaScript is **stored** on the legitimate server and sent to browsers
- Classic example: Facebook pages
  - Anybody can load a Facebook page with content provided by users
  - An attacker puts some JavaScript on their Facebook page
  - Anybody who loads the attacker's page will see JavaScript (with the origin of Facebook)
- Stored XSS requires the victim to load the page with injected JavaScript

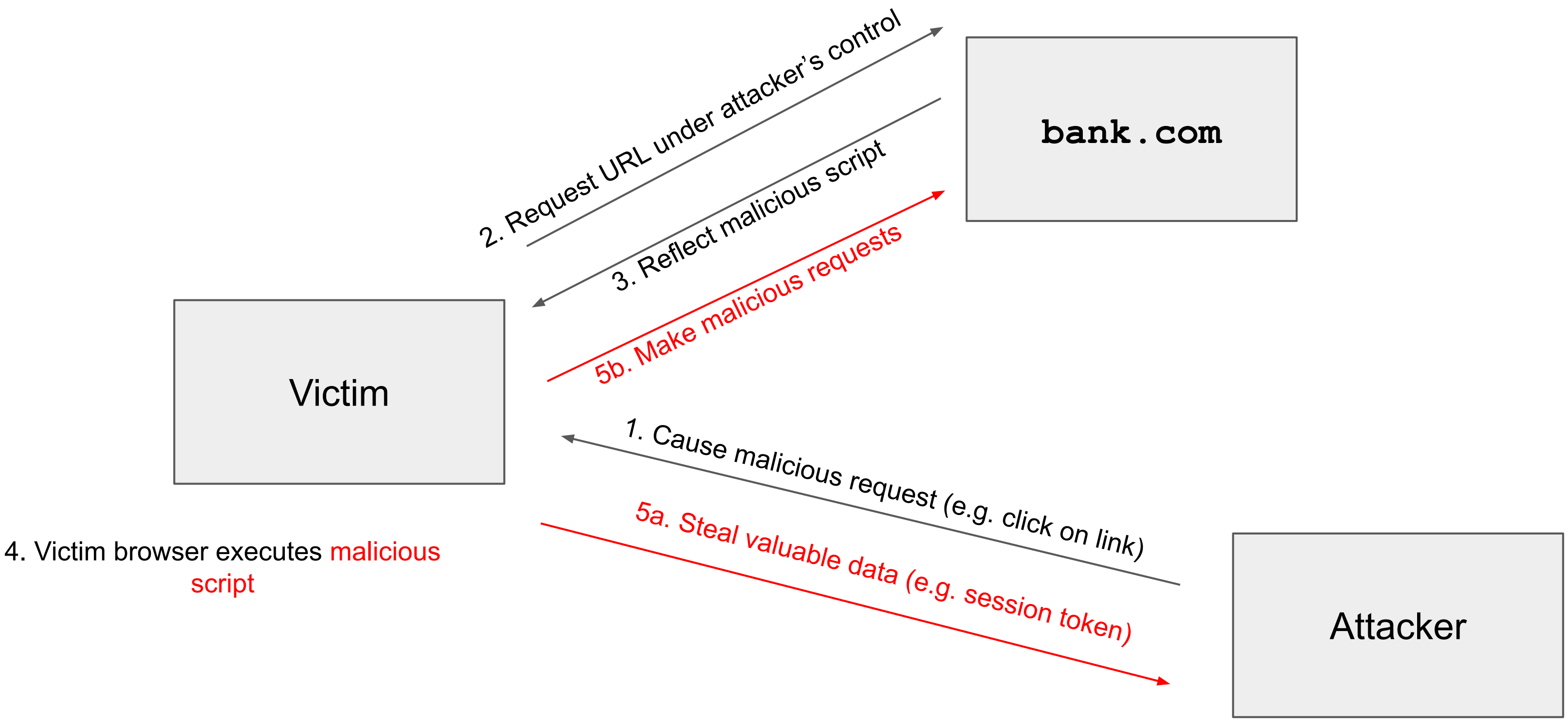
Stored XSS


bank.com
Victim
Attacker
1. Inject malicious script
2. Request content
3. Receive malicious script
4. Victim browser executes malicious script
5a. Steal valuable data (e.g. session token)
5b. Make malicious requests

Reflected XSS

- **Reflected XSS**: The attacker causes the victim to input JavaScript into a request, and the content is **reflected** (copied) in the response from the server
- Classic example: Search
  - If you make a request to `http://google.com/search?q=evanbot`, the response will say "10,000 results for evanbot"
  - If you make a request to `http://google.com/search?q=<script>alert(1)</script>`, the response will say "10,000 results for `<script>alert(1)</script>`"
- Reflected XSS requires the victim to make a request with injected JavaScript

Reflected XSS



bank.com
Victim
Attacker
2. Request URL under attacker's control
3. Reflect malicious script
5b. Make malicious requests
1. Cause malicious request (e.g. click on link)
5a. Steal valuable data (e.g. session token)
4. Victim browser executes malicious script

Reflected XSS: Making a Request

- How do we force the victim to make a request to the legitimate website with injected JavaScript?
  - Trick the victim into visiting the attacker's website, and include an embedded iframe that makes the request
    - Can make the iframe very small (1 pixel x 1 pixel), so the victim doesn't notice it:

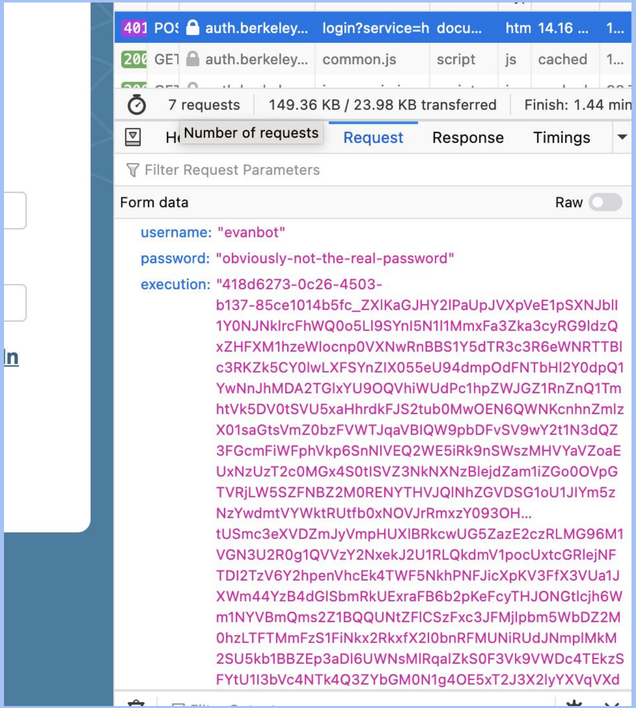
      ```
      <iframe height=1 width=1
      src="http://google.com/search?q=<script>alert(1)</script>">
      ```
  - Trick the victim into clicking a link (e.g. posting on social media, sending a text, etc.)
  - Trick the victim into visiting the attacker's website, which redirects to the reflected XSS link
  - … and many more!

Reflected XSS is not CSRF

- Reflected XSS and CSRF both require the victim to make a request to a lnk
    - XSS: An HTTP response contains maliciously inserted JavaScript, executed on the client side
    - CSRF: A malicious HTTP request is made (containing the user’s cookies), executing an effect on the server side

XSS in the Wild… On CalNet?!

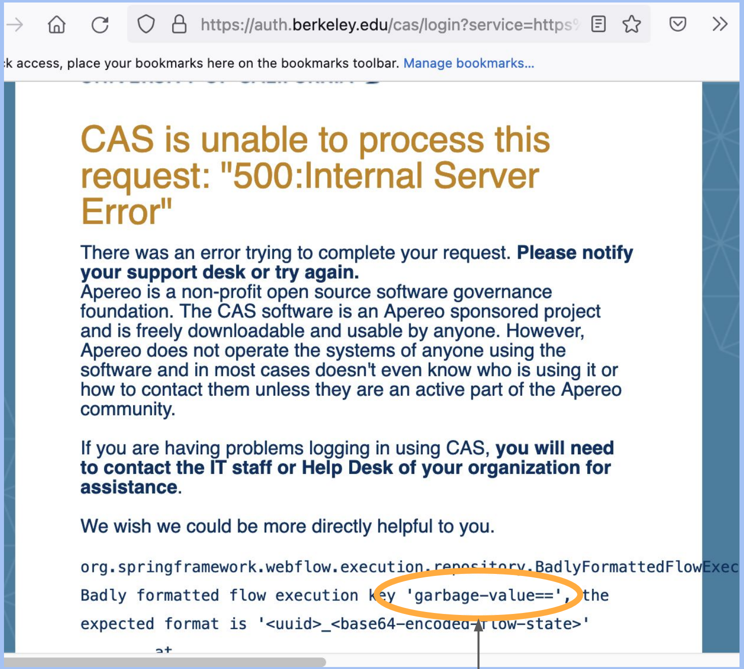
- In 2021, 61C student Rohan Mathur was exploring the CalNet login page
- Fields submitted when logging in:
  - `username`: What user am I logging in as?
  - `password`: What's the user's password?
  - `execution`: Server-side state (like a CSRF token with extra data)



```
401 POS auth.berkeley...  login?service=h  docu...  htm  14.16 ...  1...
200 GET auth.berkeley...  common.js        script   js   cached      1...
7 requests    149.36 KB / 23.98 KB transferred    Finish: 1.44 min
He Number of requests   Request   Response   Timings
Filter Request Parameters
Form data                                                       Raw
  username: "evanbot"
  password: "obviously-not-the-real-password"
  execution: "418d6273-0c26-4503-
             b137-85ce1014b5fc_ZXlKaGJHY2lPaUpJVXpVeE1pSXNJbll
             1Y0NJNklrcFhWQ0o5LI9SYnI5N1l1MmxFa3Zka3cyRG9ldzQ
             xZHFXM1hzeWlocnp0VXNwRnBBS1Y5dTR3c3R6eWNRTTBl
             c3RKZk5CY0lwLXFSYnZIX055eU94dmpOdFNTbHl2Y0dpQ1
             YwNnJhMDA2TGlxYU9OQVhiWUdPc1hpZWJGZ1RnZnQ1Tm
             htVk5DV0tSVU5xaHhrdkFJS2tub0MwOEN6QWNKcnhnZmlz
             X01saGtsVmZ0bzFVWTJqaVBlQW9pbDFvSV9wY2t1N3dQZ
             3FGcmFiWFphVkp6SnNlVEQ2WE5iRk9nSWszMHVYaVZoaE
             UxNzUzT2c0MGx4S0tISVZ3NkNXNzBlejdZam1iZGo0OVpG
             TVRjLW5SZFNBZ2M0RENYTHVJQlNhZGVDSG1oU1JlYm5z
             NzYwdmtVYWktRUtfb0xNOVJrRmxzY093OH...
             tUSmc3eXVDZmJyVmpHUXlBRkcwUG5ZazE2czRLMG96M1
             VGN3U2R0g1QVVzY2NxekJ2U1RLQkdmV1pocUxtcGRlejNF
             TDI2TzV6Y2hpenVhcEk4TWF5NkhPNFJicXpKV3FfX3VUa1J
             XWm44YzB4dGlSbmRkUExraFB6b2pKeFcyTHJONGtlcjh6W
             m1NYVBmQms2Z1BQQUNtZFlCSzFxc3JFMjlpbm5WbDZ2M
             0hzLTFTMmFzS1FiNkx2RkxfX2l0bnRFMUNiRUdJNmplMkM
             2SU5kb1BBZEp3aDl6UWNsMlRqalZkS0F3Vk9VWDc4TEkzS
             FYtU1I3bVc4NTk4Q3ZYbGM0N1g4OE5xT2J3X2lyYXVqVXd
```

XSS in the Wild… On CalNet?!

- The server expects **`execution`** to be in a specific format that contains its server-side state
  - What if you muck with **`execution`**?
- The "corrupted" execution key is placed into the error message in the HTML to aid debugging
  - … and CalNet at the time did not escape the execution key



https://auth.berkeley.edu/cas/login?service=https%

k access, place your bookmarks here on the bookmarks toolbar. Manage bookmarks…

CAS is unable to process this request: "500:Internal Server Error"

There was an error trying to complete your request. **Please notify your support desk or try again.** Apereo is a non-profit open source software governance foundation. The CAS software is an Apereo sponsored project and is freely downloadable and usable by anyone. However, Apereo does not operate the systems of anyone using the software and in most cases doesn't even know who is using it or how to contact them unless they are an active part of the Apereo community.

If you are having problems logging in using CAS, **you will need to contact the IT staff or Help Desk of your organization for assistance**.

We wish we could be more directly helpful to you.

```
org.springframework.webflow.execution.repository.BadlyFormattedFlowExec
Badly formatted flow execution key 'garbage-value==', the
expected format is '<uuid>_<base64-encoded-flow-state>'
        at
```

Garbage execution value in HTML

Constructing an Attack on CalNet

Attack: Force a POST request to CalNet!

```
<html>
  <head>
    <script>
      // When the malicious page finishes loading, automatically submit the form!
      document.addEventListener('DOMContentLoaded', () => {
        document.getElementById('form').submit();
      });
    </script>
  </head>
  <body>
    <!-- Malicious form containing our malicious execution data. -->
    <form id="form" action="https://auth.berkeley.edu/cas/login" method="POST">
      <input name="username" type="text" value="evanbot" />
      <input name="password" type="text" value="obviously-not-the-real-password" />
      <input name="execution" type="text" value="<script>alert('XSS!')</script>" />
    </form>
  </body>
</html>
```

So What Happened?

- CalNet also accepts **`execution`** parameters over URL query parameters
  - A link like **`https://auth.berkeley.edu/cas/login?execution=<script>alert('XSS!')</script>`** would result in the same attack
- It would only fire if you clicked the button to show the error
- … But if clicked, the injected JavaScript could
  - Present a fake login prompt
  - Steal your CalNet password
  - Log you in as the attacker’s account
  - Steal your authentication session token
- Root cause: A >5 year old sample page modified by CalNet
  - CalNet is uses software from Apereo, which comes with sample pages for logging in
  - Apereo fixed that bug many years before, but sample pages don’t get included in bug fixes!

XSS Defenses

- Defense: **HTML sanitization**
  - Idea: Certain characters are special, so create sequences that represent those characters as data, rather than as HTML
- Start with an ampersand (`&`) and end with a semicolon (`;`)
  - Instead of `<`, use **`&lt;`**
  - Instead of `"`, use **`&quot;`**
  - And many more!
    - It is important to escape all dangerous characters (lists of them can be found), or you will still be vulnerable!
- Note: You should always rely on trusted libraries to do this for you!

```
<html>
<body>
Hello &lt;script&gt;alert(1)&lt;/script&gt;!
</body>
</html>
```

XSS Defenses: Escaping

Handler

```
func handleSayHello(w http.ResponseWriter, r *http.Request) {
    name := r.URL.Query()["name"][0]
    fmt.Fprintf(w, "<html><body>Hello %s!</body></html>", html.EscapeString(name))
}
```

URL

```
https://vulnerable.com/hello?name=<script>alert(1)</script>
```

Response

```
<html><body>Hello &lt;script&gt;alert(1)&lt;/script&gt;!</body></html>
```

XSS Defenses: Escaping

- If a programmer has to take an action for every usage…
    - They are *going* to screw up (e.g. CalNet)
    - Recall: Consider human factors!
- Nowadays, escaping is generally achieved through **templating**
    - HTML templates are essentially their own language, where you declare what data goes where
    - The templating engine handles all the escaping internally
    - The HTTP library gets very angry if you don't use templates
    - Recall (again): Consider human factors!

```
<html>
<body>
Hello {{.name}}!
</body>
</html>
```

Example: Golang HTML template

XSS Defenses: CSP

- Defense: **Content Security Policy** (**CSP**)
  - Idea: Instruct the browser to only use resources loaded from specific places
  - Uses additional headers to specify the policy
- Standard approach:
  - Disallow all inline scripts (JavaScript code directly in HTML), which prevents inline XSS
    - Example: Disallow **`<script>alert(1)</script>`**
  - Only allow scripts from specified domains, which prevents XSS from linking to external scripts
    - Example: Disallow **`<script src="https://cs161.org/hack.js">`**
- Also works with other content (e.g. iframes, images, etc.)
- Relies on the browser to enforce security, so more of a mitigation for defense-in-depth

UI Attacks

Textbook Chapter 23

User Interface (UI) Attacks

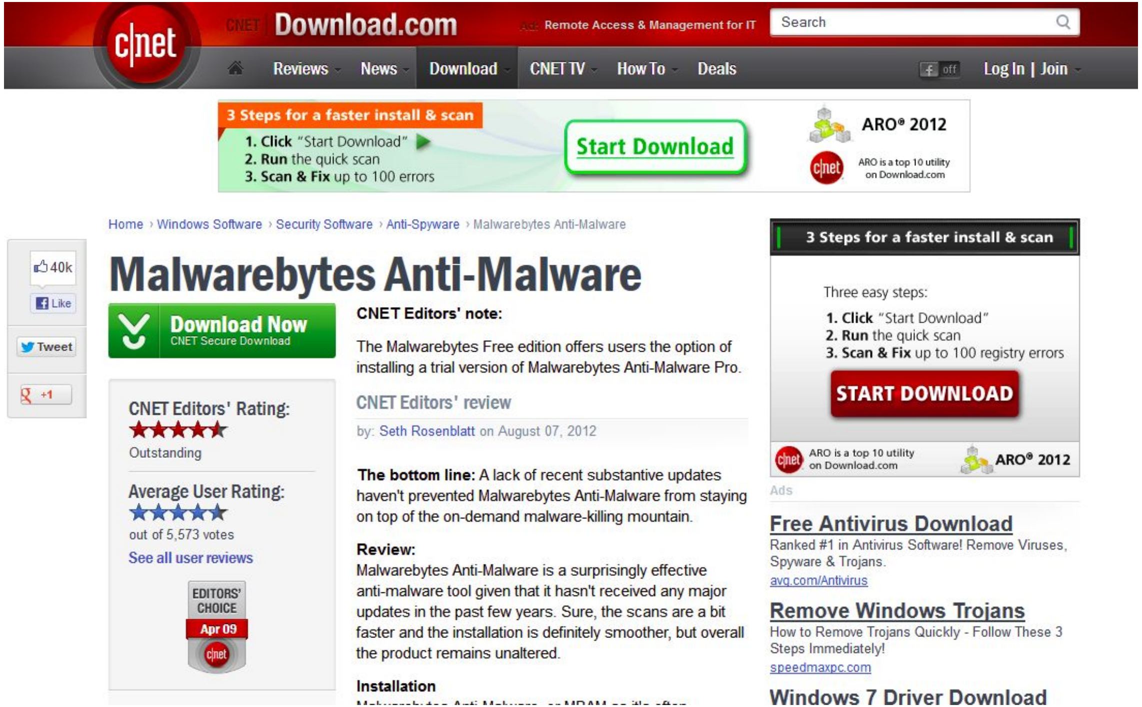
- General theme: The attacker tricks the victim into thinking they are taking an **intended** action, when they are actually taking a **malicious** action
  - Takes advantage of **user interfaces**: The trusted path between the user and the computer
    - Browser disallows the website itself to interact across origins (same-origin policy), but trusts the user to do whatever they want
  - Remember: Consider human factors!
- Two main types of UI attacks
  - Clickjacking: Trick the victim into clicking on something from the attacker
  - Phishing: Trick the victim into sending the attacker personal information

Clickjacking

- **Clickjacking**: Trick the victim into clicking on something from the attacker
- Main vulnerability: the browser trusts the user's clicks
  - When the user clicks on something, the browser assumes the user intended to click there
- Why steal clicks?
  - Download a malicious program
  - Like a Facebook page/YouTube video
  - Delete an online account
- Why steal keystrokes?
  - Steal passwords
  - Steal credit card numbers
  - Steal personal info

Clickjacking: Download buttons

- Which is the real download button?
- What if the user clicks the wrong one?



3 Steps for a faster install & scan

1. Click "Start Download"
2. Run the quick scan
3. Scan & Fix up to 100 errors

Home › Windows Software › Security Software › Anti-Spyware › Malwarebytes Anti-Malware

Malwarebytes Anti-Malware

CNET Editors' Rating:

★★★★★

Outstanding

Average User Rating:

★★★★★

out of 5,573 votes

See all user reviews

CNET Editors' note:

The Malwarebytes Free edition offers users the option of installing a trial version of Malwarebytes Anti-Malware Pro.

CNET Editors' review

by: Seth Rosenblatt on August 07, 2012

The bottom line: A lack of recent substantive updates haven't prevented Malwarebytes Anti-Malware from staying on top of the on-demand malware-killing mountain.

Review:

Malwarebytes Anti-Malware is a surprisingly effective anti-malware tool given that it hasn't received any major updates in the past few years. Sure, the scans are a bit faster and the installation is definitely smoother, but overall the product remains unaltered.

Installation

3 Steps for a faster install & scan

Three easy steps:

1. Click "Start Download"
2. Run the quick scan
3. Scan & Fix up to 100 registry errors

Ads

Free Antivirus Download

Ranked #1 in Antivirus Software! Remove Viruses, Spyware & Trojans.

avg.com/Antivirus

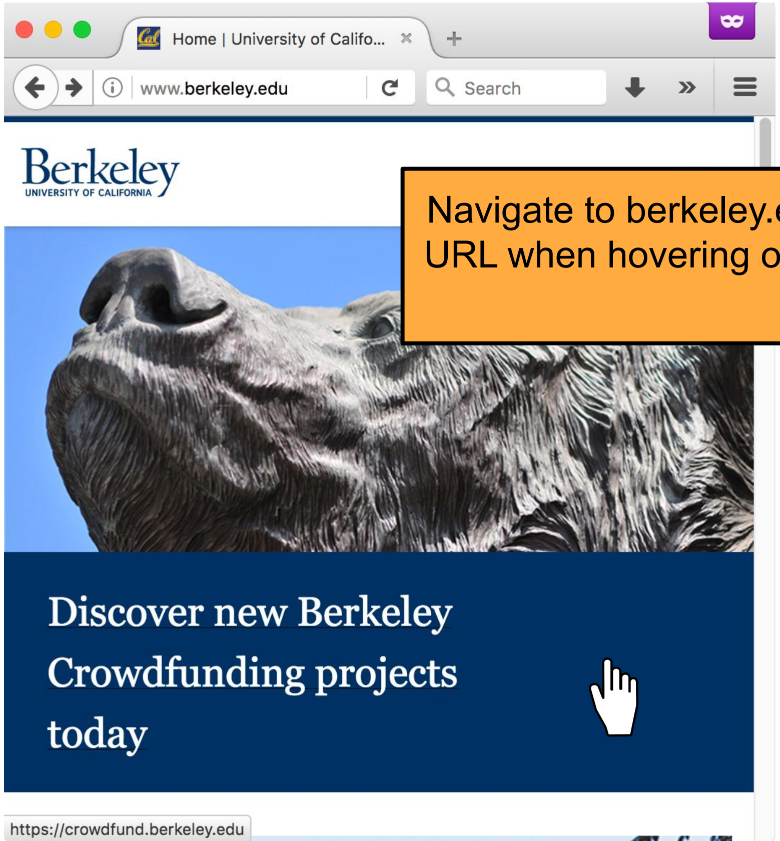
Remove Windows Trojans

How to Remove Trojans Quickly - Follow These 3 Steps Immediately!

speedmaxpc.com

Windows 7 Driver Download

Clickjacking



Discover new Berkeley Crowdfunding projects today

https://crowdfund.berkeley.edu

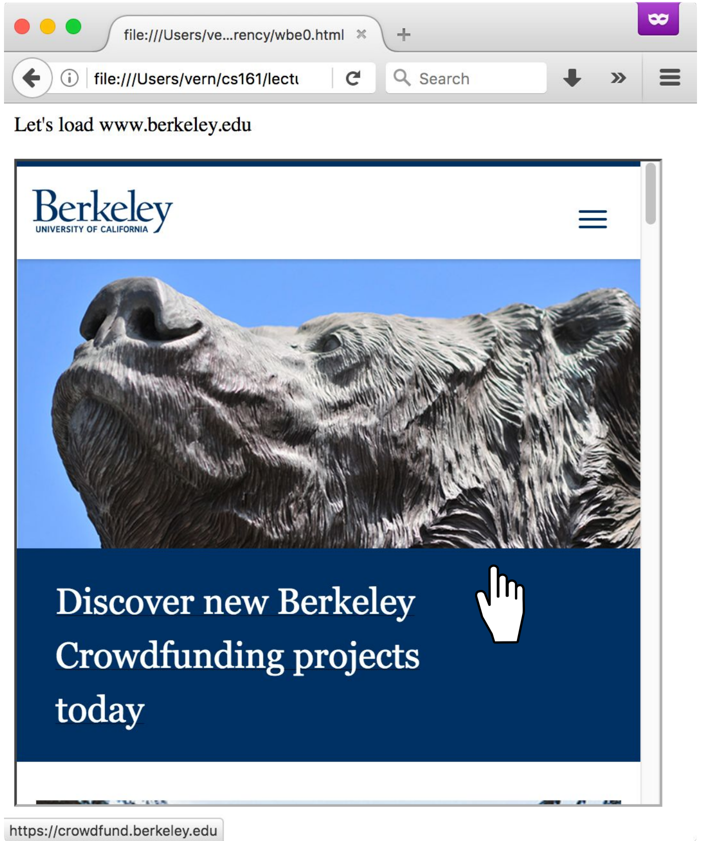
Navigate to berkeley.edu. Notice the URL when hovering over the image.

Clickjacking

Load berkeley.edu in an iframe

We can't generate clicks ourselves because of SOP, but the user can still click…

```
<iframe style="opacity: 1.0"
src="https://www.berkeley.edu/"></iframe>
```



file:///Users/ve...rency/wbe0.html
file:///Users/vern/cs161/lectu
Search
Berkeley
UNIVERSITY OF CALIFORNIA

Let's load www.berkeley.edu

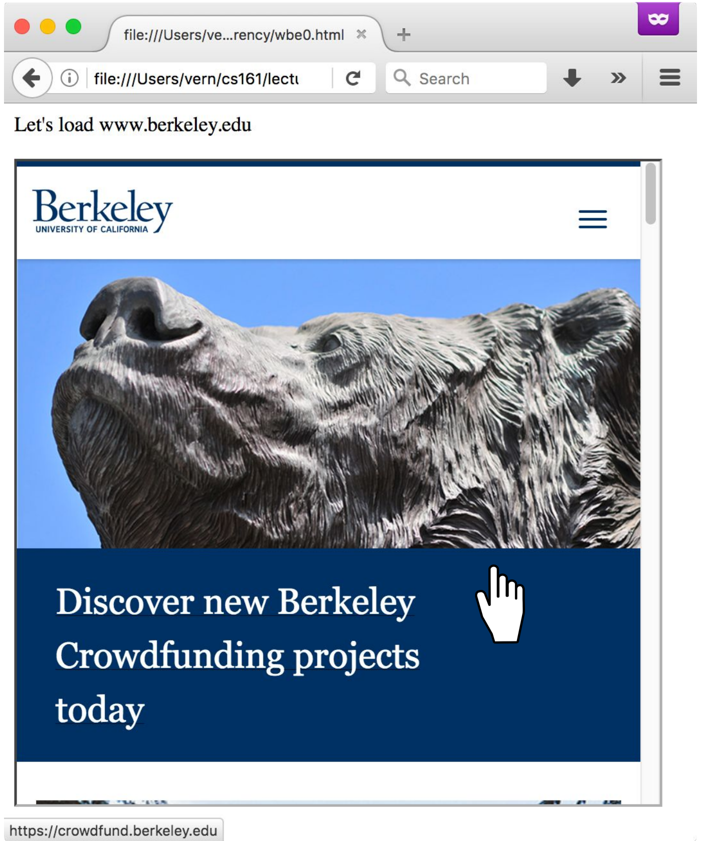
Discover new Berkeley Crowdfunding projects today

https://crowdfund.berkeley.edu

Clickjacking

Place some enticing content underneath

```
<iframe style="opacity: 1.0"
src="https://www.berkeley.edu/"></iframe>
<p style="margin-top: 210pt"><em>You <b>Know</b>
You Want To Click Here!</em></p>
```

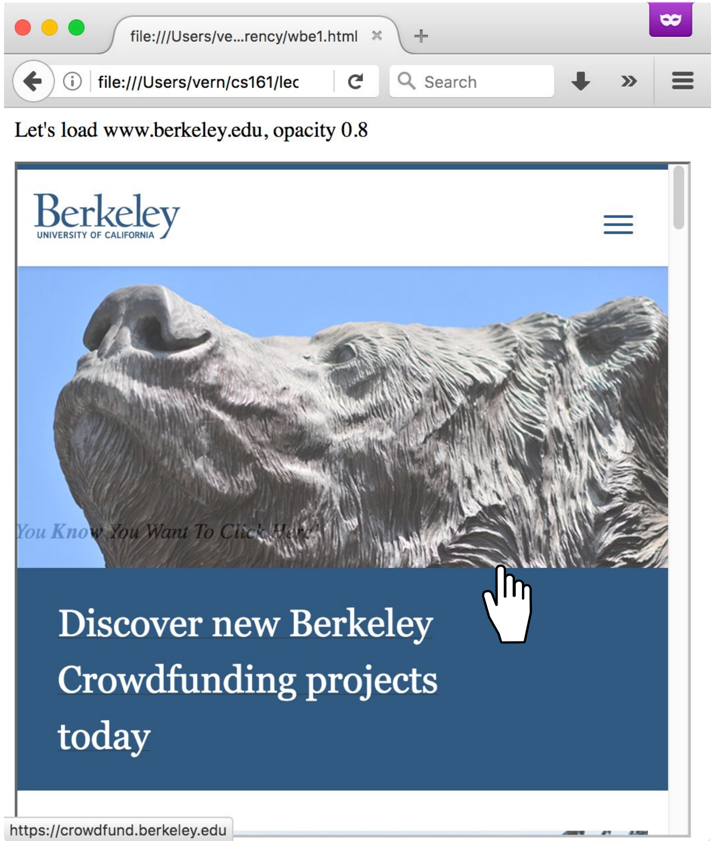

file:///Users/ve...rency/wbe0.html
file:///Users/vern/cs161/lectu
Search
Berkeley
UNIVERSITY OF CALIFORNIA
Discover new Berkeley Crowdfunding projects today
https://crowdfund.berkeley.edu

Let's load www.berkeley.edu

Clickjacking

Make the iframe slightly transparent…

```
<iframe style="opacity: 0.8"
src="https://www.berkeley.edu/"></iframe>
<p style="margin-top: 210pt"><em>You <b>Know</b>
You Want To Click Here!</em></p>
```


file:///Users/ve...rency/wbe1.html
file:///Users/vern/cs161/lec
Search
Berkeley
UNIVERSITY OF CALIFORNIA
https://crowdfund.berkeley.edu

Let's load www.berkeley.edu, opacity 0.8

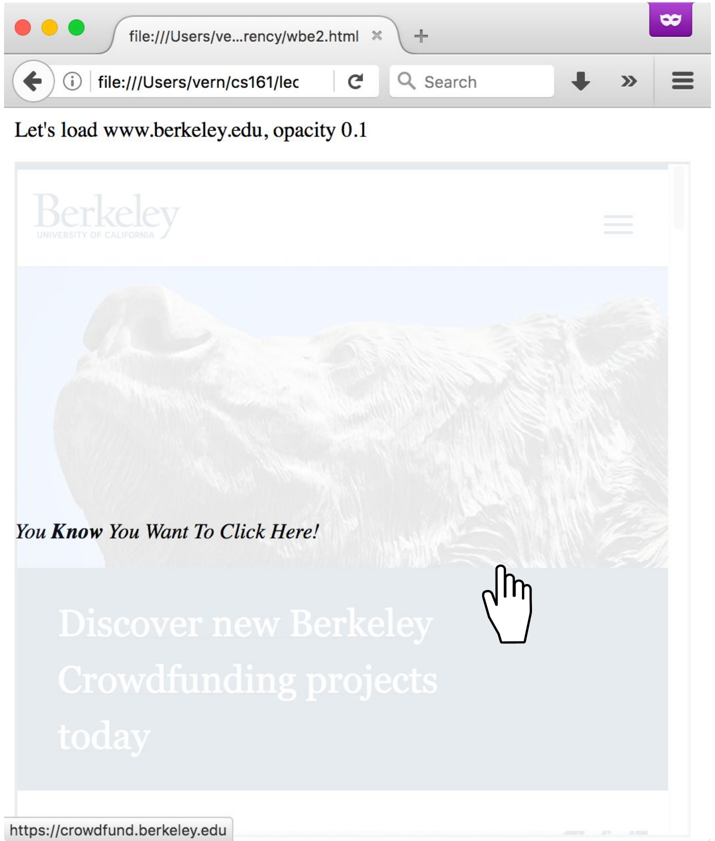
*You **Know** You Want To Click Here!*

Discover new Berkeley Crowdfunding projects today

Clickjacking

Make it *more* transparent

```
<iframe style="opacity: 0.1"
src="https://www.berkeley.edu/"></iframe>
<p style="margin-top: 210pt"><em>You <b>Know</b>
You Want To Click Here!</em></p>
```


file:///Users/ve...rency/wbe2.html
file:///Users/vern/cs161/lec
Search
Berkeley
UNIVERSITY OF CALIFORNIA
Discover new Berkeley Crowdfunding projects today
https://crowdfund.berkeley.edu

Let's load www.berkeley.edu, opacity 0.1

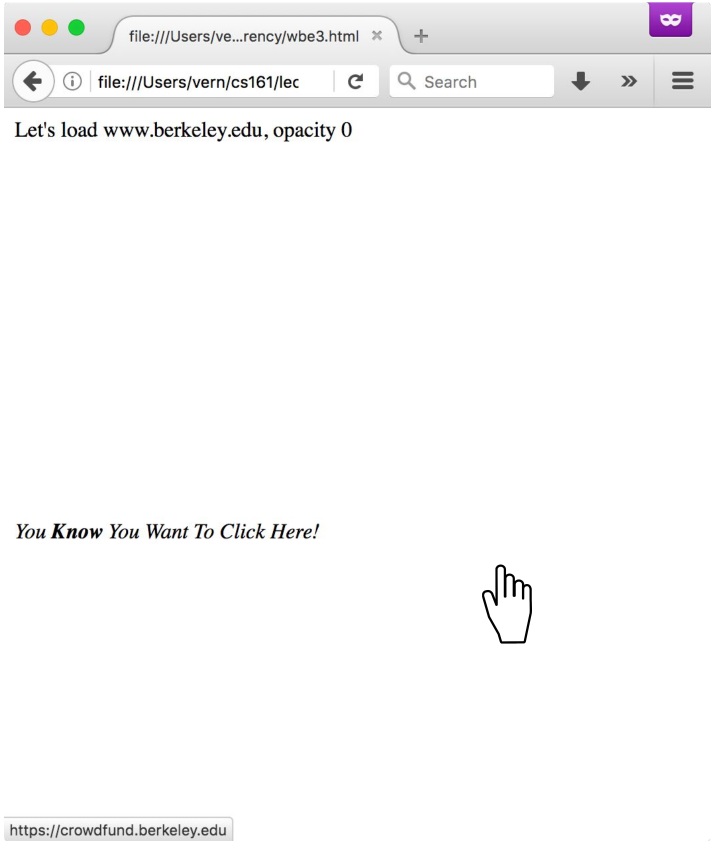
*You **Know** You Want To Click Here!*

Clickjacking

Make it *entirely* transparent

But the user still clicks on the iframe!

```
<iframe style="opacity: 0"
src="https://www.berkeley.edu/"></iframe>
<p style="margin-top: 210pt"><em>You <b>Know</b>
You Want To Click Here!</em></p>
```


file:///Users/ve...rency/wbe3.html
file:///Users/vern/cs161/lec
Search

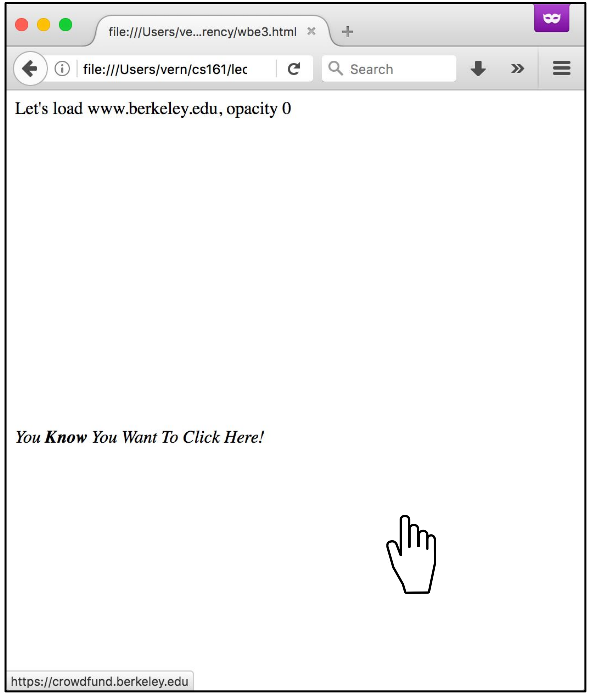
Let's load www.berkeley.edu, opacity 0

*You **Know** You Want To Click Here!*

https://crowdfund.berkeley.edu

Clickjacking: Invisible iframes

- Variant #1: Frame the legitimate site invisibly, *over* visible, enticing content
  - Victim thinks they're clicking on the attacker's enticing website
  - But their click actually happened on the legitimate website!


file:///Users/ve...rency/wbe3.html
file:///Users/vern/cs161/lec
Search

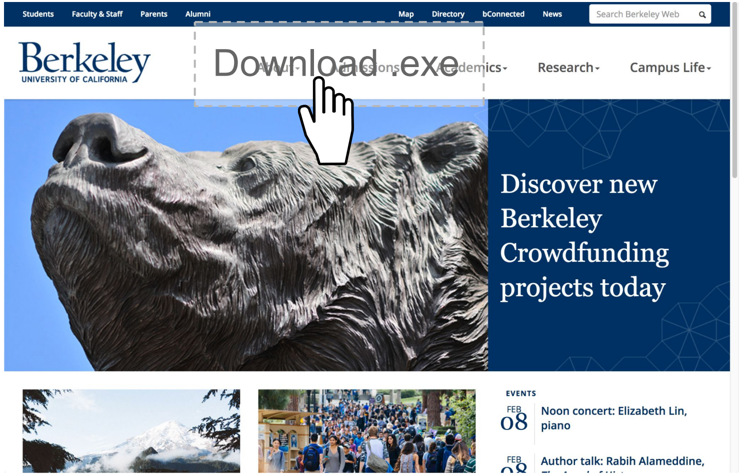
Let's load www.berkeley.edu, opacity 0

*You **Know** You Want To Click Here!*

https://crowdfund.berkeley.edu

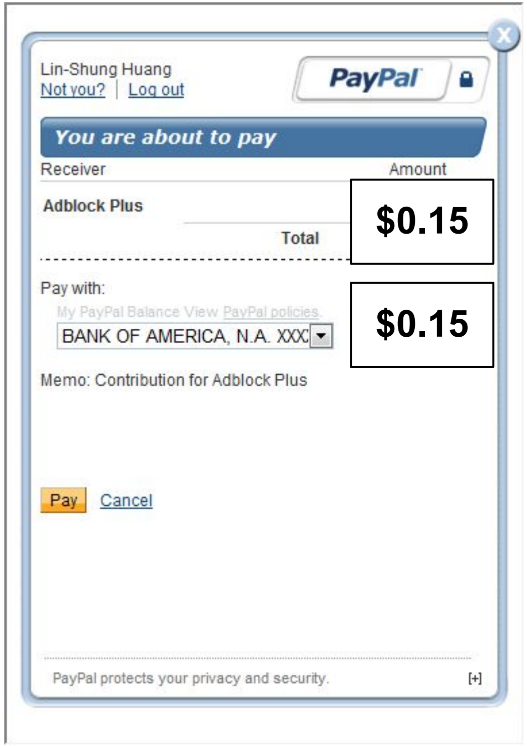
Clickjacking: Invisible iframes

- Variant #2: Frame the legitimate site visibly, *under* invisible malicious content
    - Victim thinks they're clicking on the legitimate site
    - But their click actually happened on the malicious website!



Clickjacking: Invisible iframes

- Variant #3: Frame the legitimate site visibly, *under* malicious content *partially* overlaying the site
  - The attacker can change the appearance of the site without breaking SOP!



Lin-Shung Huang
Not you? | Log out

You are about to pay

Receiver	Amount
Adblock Plus	$0.15
Total	$0.15

Pay with:
My PayPal Balance View PayPal policies.
BANK OF AMERICA, N.A. XXX

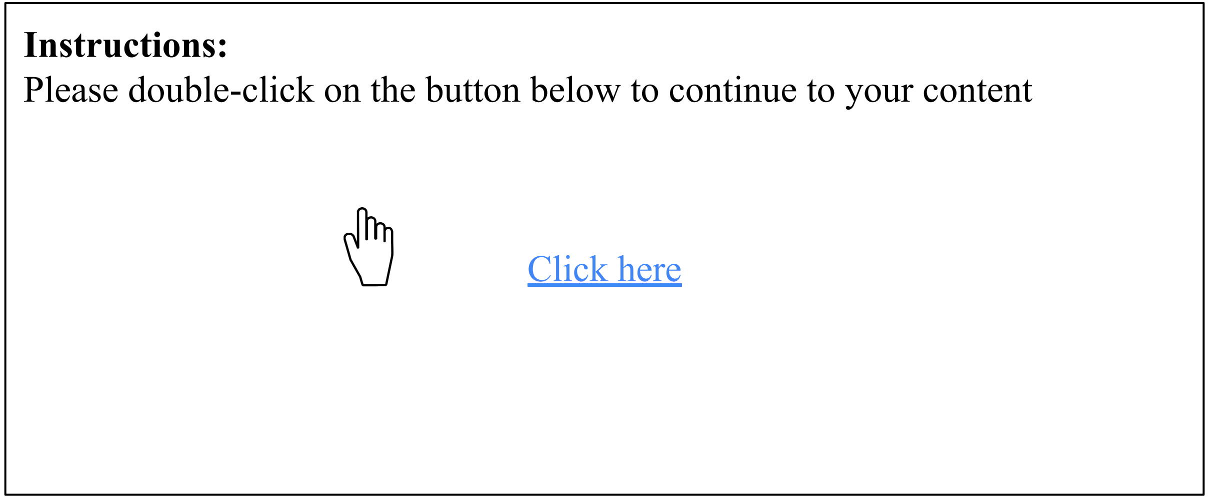
Memo: Contribution for Adblock Plus

PayPal protects your privacy and security. [+]

Clickjacking: Temporal Attack

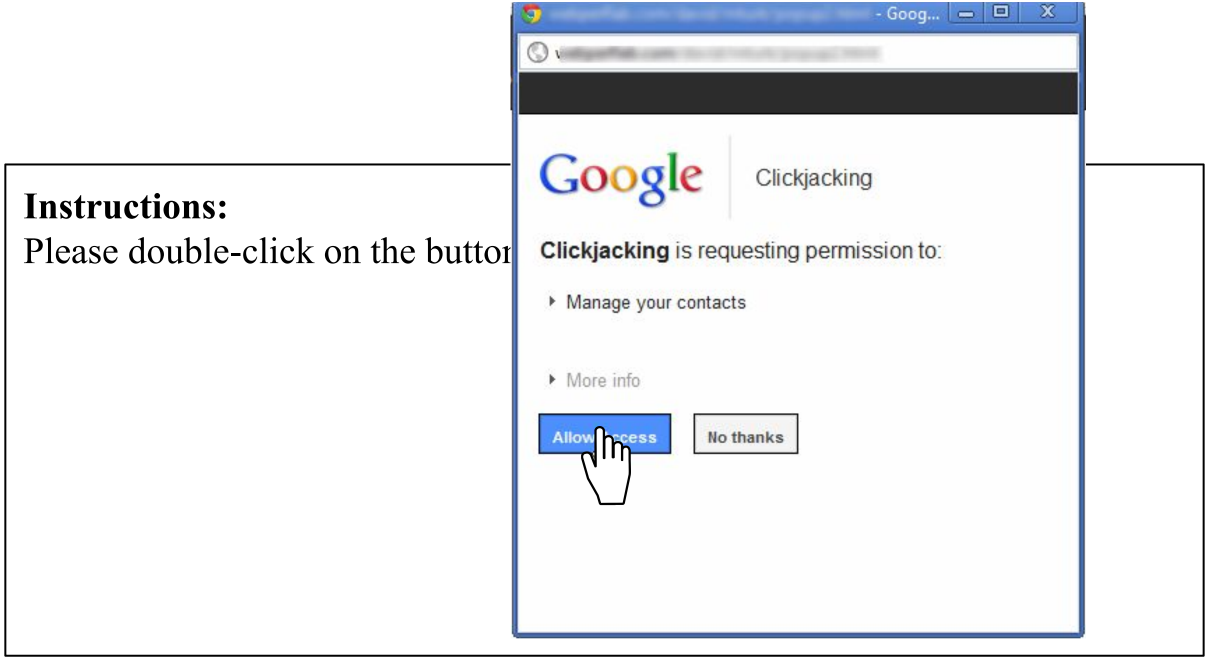
- JavaScript can detect the position of the cursor and change the website right before the user clicks on something
    - The user clicks on the malicious input (embedded iframe, download button, etc.) before they notice that something changed

Clickjacking: Temporal Attack



Instructions:
Please double-click on the button below to continue to your content

Clickjacking: Temporal Attack



Instructions:

Please double-click on the butto

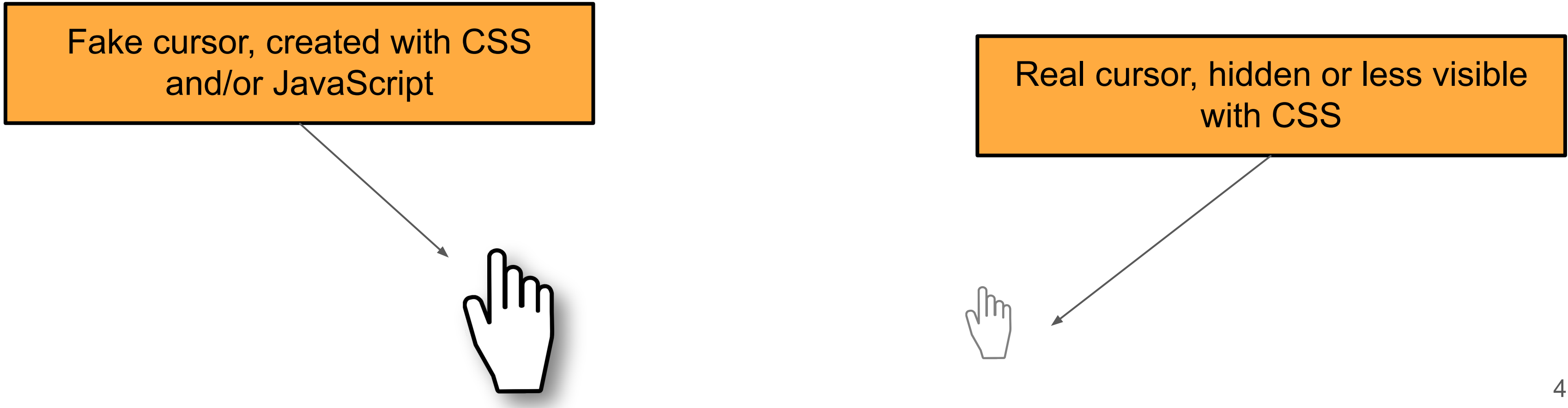
Google | Clickjacking

Clickjacking is requesting permission to:

- Manage your contacts
- More info

Clickjacking: Cursorjacking

- CSS has the ability to style the appearance of the cursor
- JavaScript has the ability to track a cursor’s position
- If we change the appearance a certain way, we can create a fake cursor to trick users into clicking on things!

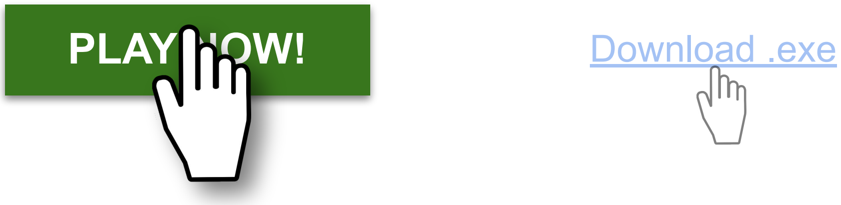


Fake cursor, created with CSS and/or JavaScript

Real cursor, hidden or less visible with CSS

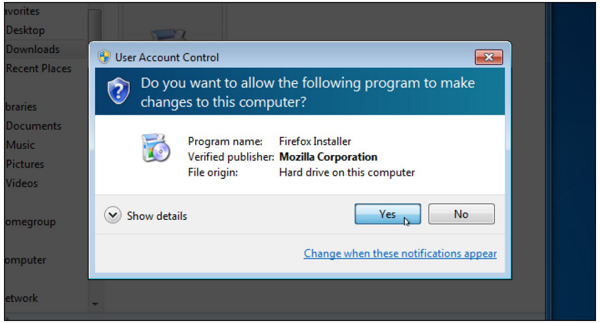
Clickjacking: Cursorjacking

What do you think you’re clicking on?



Clickjacking: Defenses

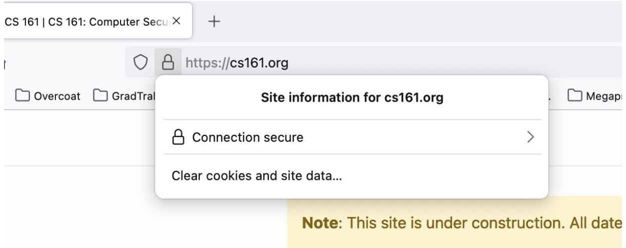
- **Enforce visual integrity**: Ensure clear visual separation between important dialogs and content
  - Notice: Windows User Account Control darkens the entire screen and freezes the desktop
  - Notice: Firefox dialogs “cross the boundary” between the URL bar and content, something that only valid dialogs can do



User Account Control

Do you want to allow the following program to make changes to this computer?

Program name: Firefox Installer
Verified publisher: **Mozilla Corporation**
File origin: Hard drive on this computer



CS 161 | CS 161: Computer Secu

https://cs161.org

Site information for cs161.org

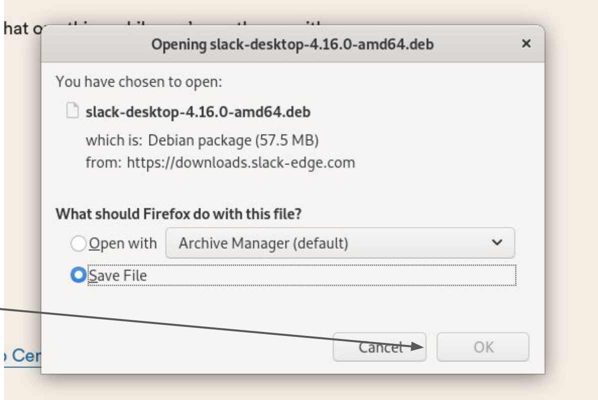
Connection secure

Clear cookies and site data...

Note: This site is under construction. All date

Clickjacking: Defenses

- **Enforce temporal integrity**: Ensure that there is sufficient time for a user to register what they are clicking on
    - Notice: Firefox blocks the “OK” button until 1 second after the dialog has been focused



Opening slack-desktop-4.16.0-amd64.deb

You have chosen to open:

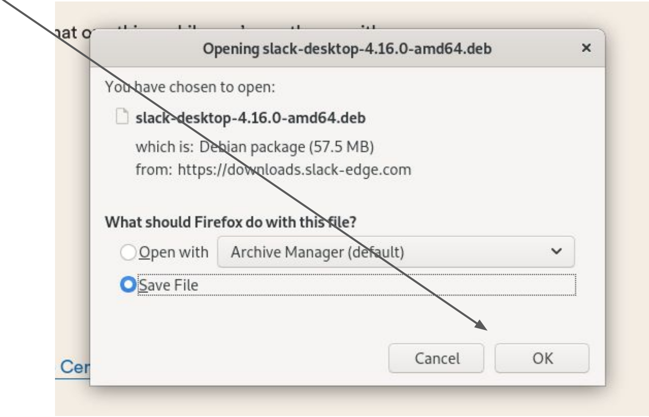
slack-desktop-4.16.0-amd64.deb

which is: Debian package (57.5 MB)
from: https://downloads.slack-edge.com

What should Firefox do with this file?

○ Open with Archive Manager (default)

● Save File



Opening slack-desktop-4.16.0-amd64.deb

You have chosen to open:

slack-desktop-4.16.0-amd64.deb

which is: Debian package (57.5 MB)
from: https://downloads.slack-edge.com

What should Firefox do with this file?

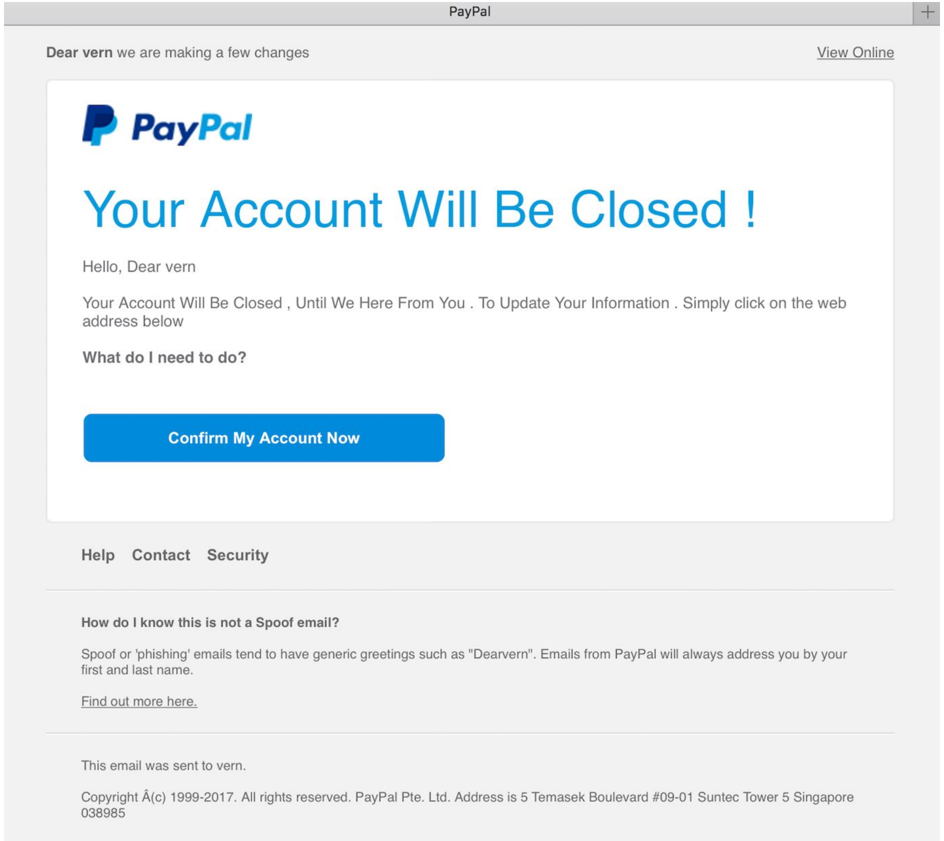
○ Open with Archive Manager (default)

● Save File

Clickjacking: Defenses

- **Require confirmation** from users
  - The browser needs to confirm that the user’s click was intentional
  - Drawbacks: Asking for confirmation annoys users (consider human factors!)
- **Frame-busting**: The legitimate website forbids other websites from embedding it in an iframe
  - Defeats the invisible iframe attacks
  - Can be enforced by Content Security Policy (CSP)
  - Can be enforced by X-Frame-Options (an HTTP header)

Phishing



Dear vern we are making a few changes

View Online

Your Account Will Be Closed !

Hello, Dear vern

Your Account Will Be Closed , Until We Here From You . To Update Your Information . Simply click on the web address below

What do I need to do?

How do I know this is not a Spoof email?

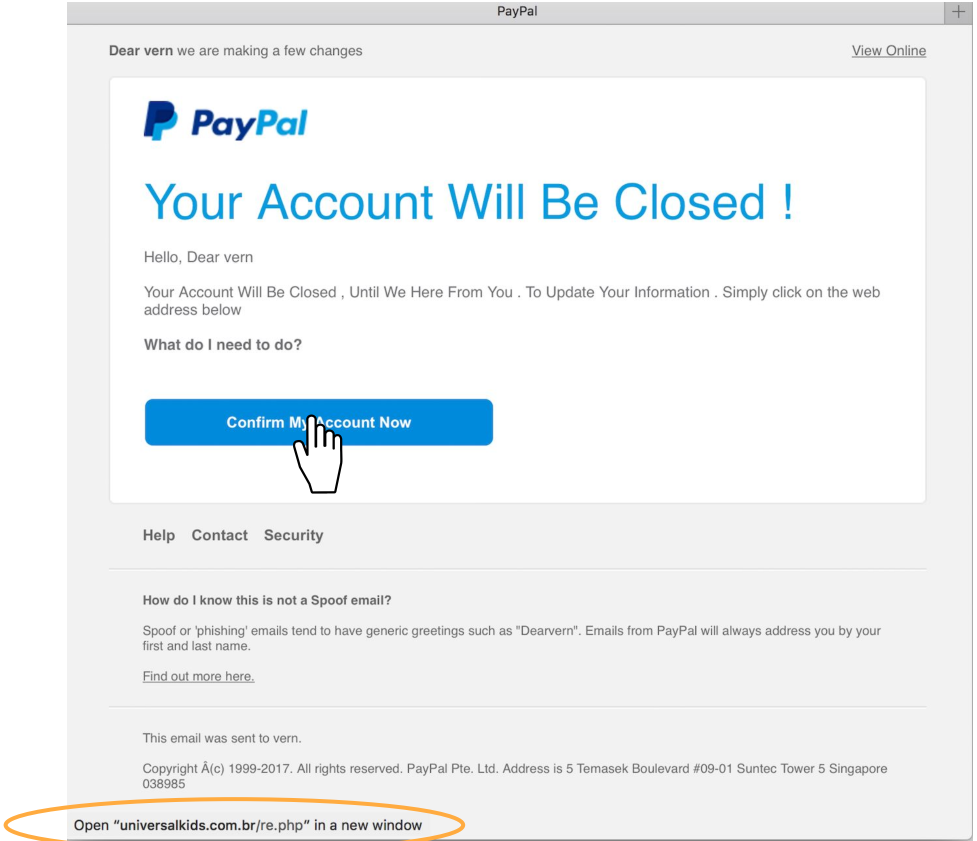
Spoof or 'phishing' emails tend to have generic greetings such as "Dearvern". Emails from PayPal will always address you by your first and last name.

Find out more here.

This email was sent to vern.

Copyright Â(c) 1999-2017. All rights reserved. PayPal Pte. Ltd. Address is 5 Temasek Boulevard #09-01 Suntec Tower 5 Singapore 038985

Phishing



PayPal

Dear vern we are making a few changes

View Online

Your Account Will Be Closed !

Hello, Dear vern

Your Account Will Be Closed , Until We Here From You . To Update Your Information . Simply click on the web address below

What do I need to do?

Help **Contact** **Security**

How do I know this is not a Spoof email?

Spoof or 'phishing' emails tend to have generic greetings such as "Dearvern". Emails from PayPal will always address you by your first and last name.

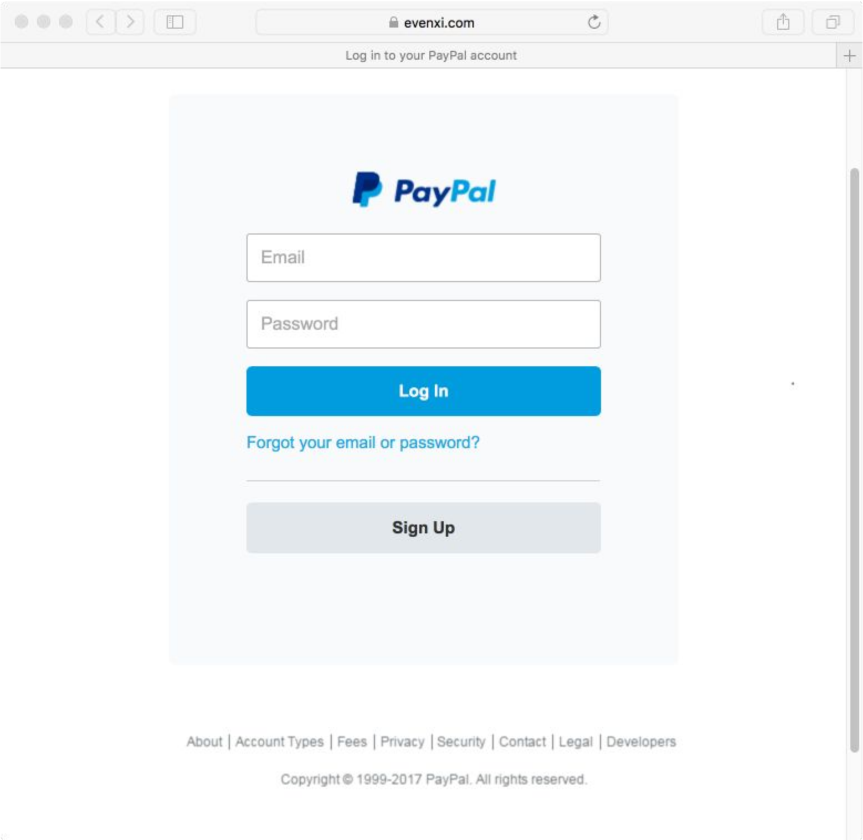
Find out more here.

This email was sent to vern.

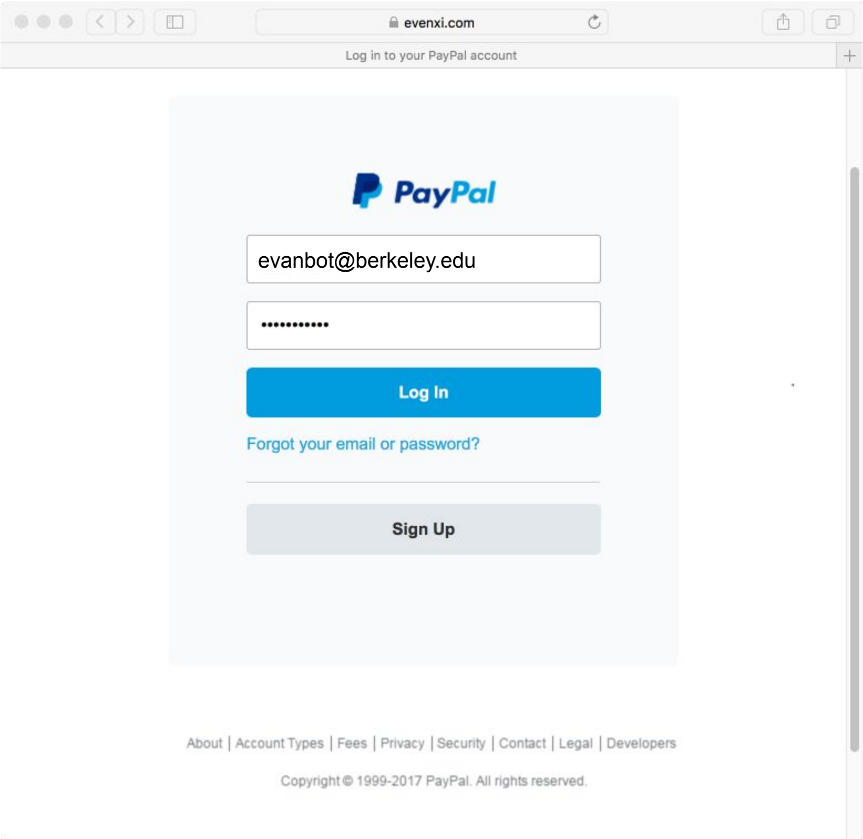
Copyright Â(c) 1999-2017. All rights reserved. PayPal Pte. Ltd. Address is 5 Temasek Boulevard #09-01 Suntec Tower 5 Singapore 038985

Open “**universalkids.com.br**/re.php” in a new window

Phishing


evenxi.com
Log in to your PayPal account
PayPal
Email
Password
Log In
Forgot your email or password?
Sign Up
About | Account Types | Fees | Privacy | Security | Contact | Legal | Developers
Copyright © 1999-2017 PayPal. All rights reserved.

Phishing



evenxi.com

Log in to your PayPal account

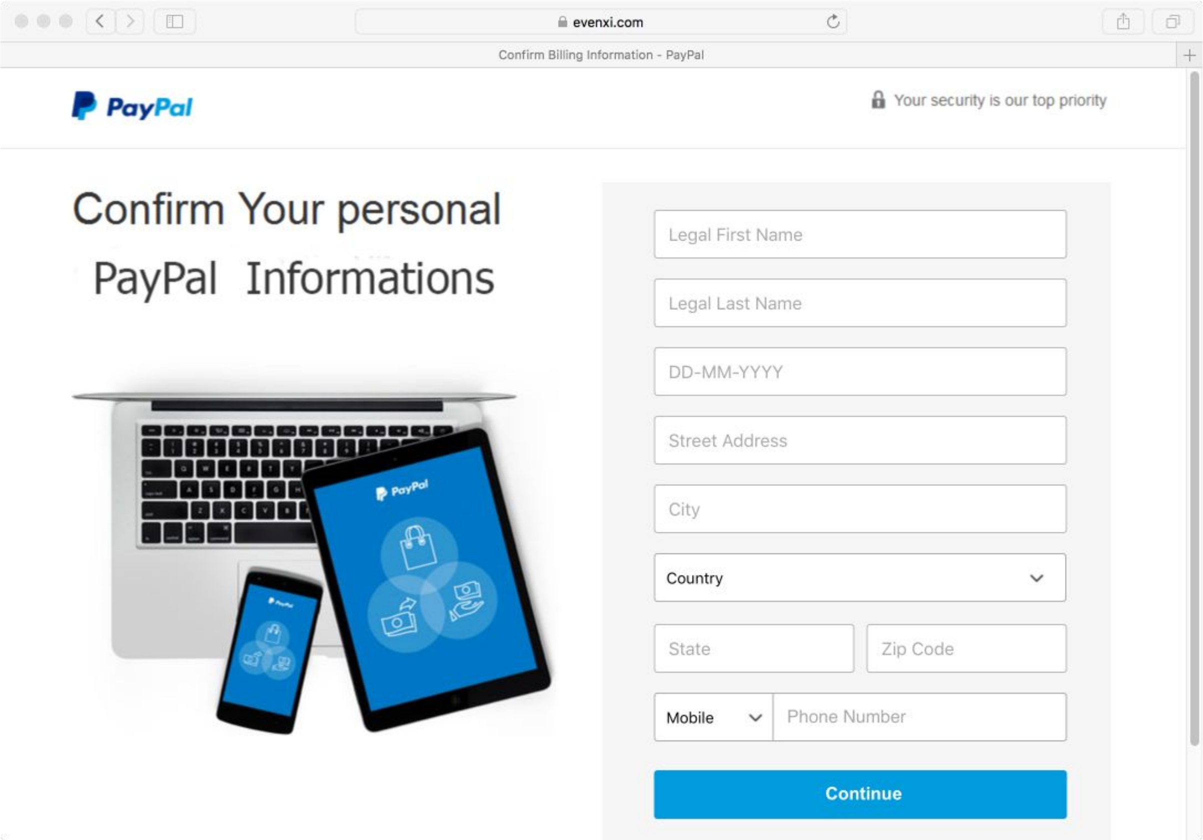
evanbot@berkeley.edu

••••••••••••

Log In
Forgot your email or password?
Sign Up
About | Account Types | Fees | Privacy | Security | Contact | Legal | Developers

Copyright © 1999-2017 PayPal. All rights reserved.

Phishing



evenxi.com

Confirm Billing Information - PayPal

PayPal

Your security is our top priority

Confirm Your personal PayPal Informations

Legal First Name

Legal Last Name

DD-MM-YYYY

Street Address

City

Country

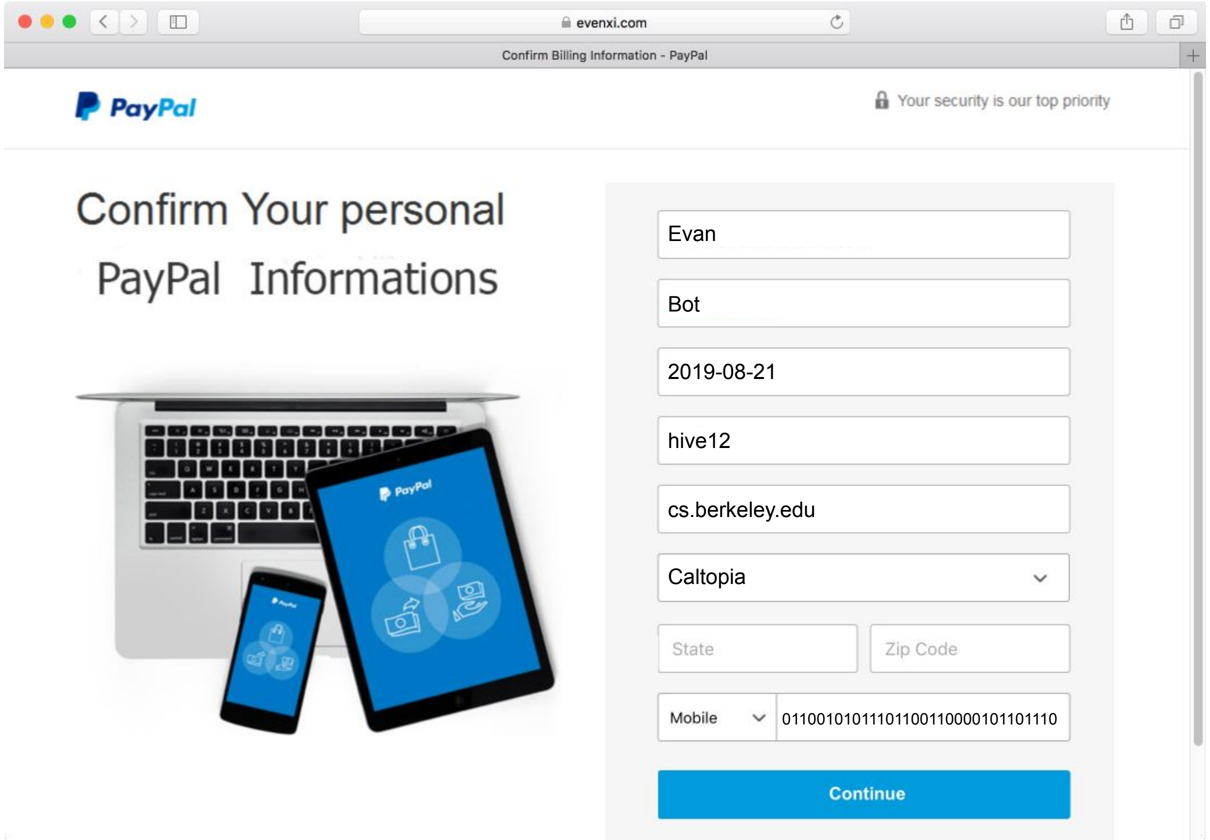
State

Zip Code

Mobile

Phone Number

Phishing


evenxi.com
Confirm Billing Information - PayPal
PayPal
Your security is our top priority
PayPal
State
Zip Code
Mobile
Continue

Confirm Your personal PayPal Informations

Evan

Bot

2019-08-21

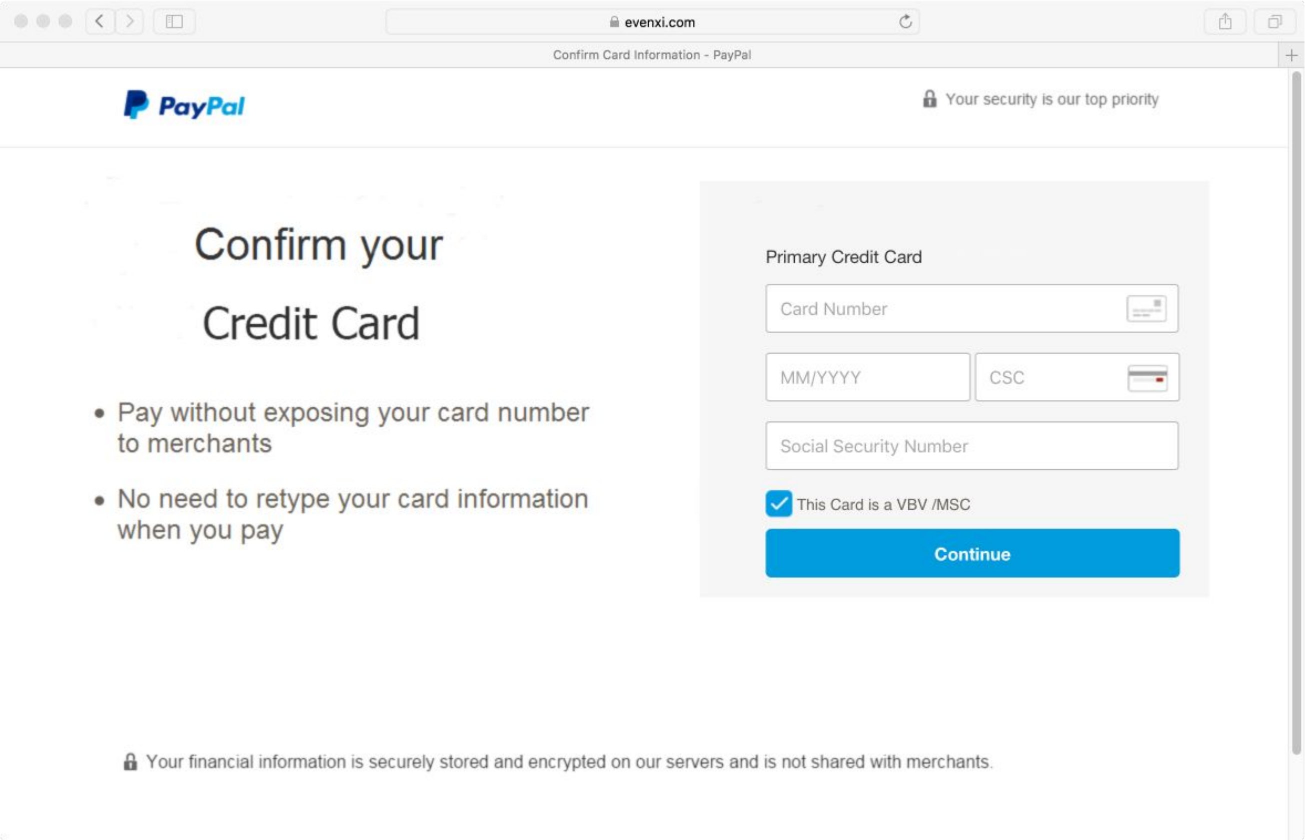
hive12

cs.berkeley.edu

Caltopia

01100101011101100110000101101110

Phishing



Your security is our top priority

Confirm your Credit Card

- Pay without exposing your card number to merchants
- No need to retype your card information when you pay

Primary Credit Card

Card Number

MM/YYYY

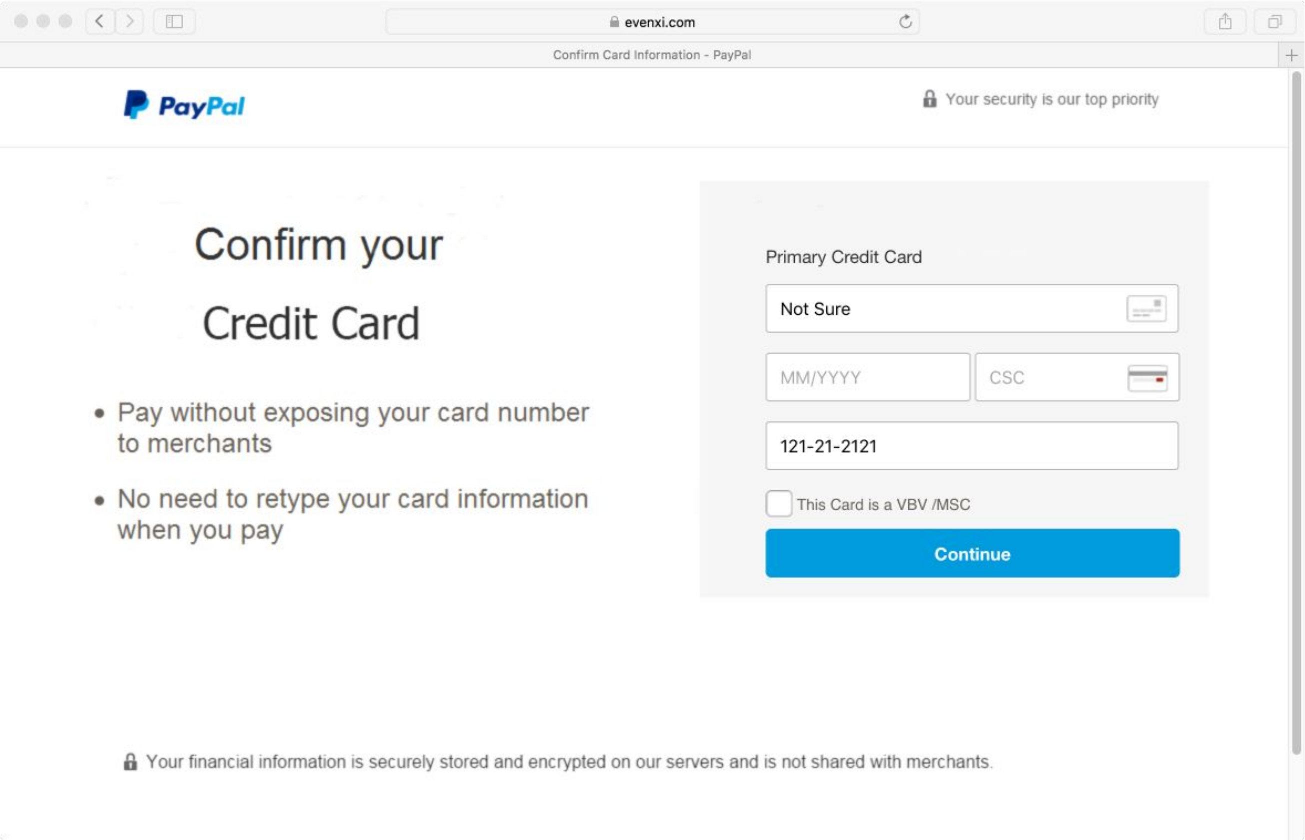
CSC

Social Security Number

☑ This Card is a VBV /MSC

Your financial information is securely stored and encrypted on our servers and is not shared with merchants.

Phishing



Your security is our top priority

Confirm your Credit Card

- Pay without exposing your card number to merchants
- No need to retype your card information when you pay

Primary Credit Card

Not Sure

MM/YYYY

CSC

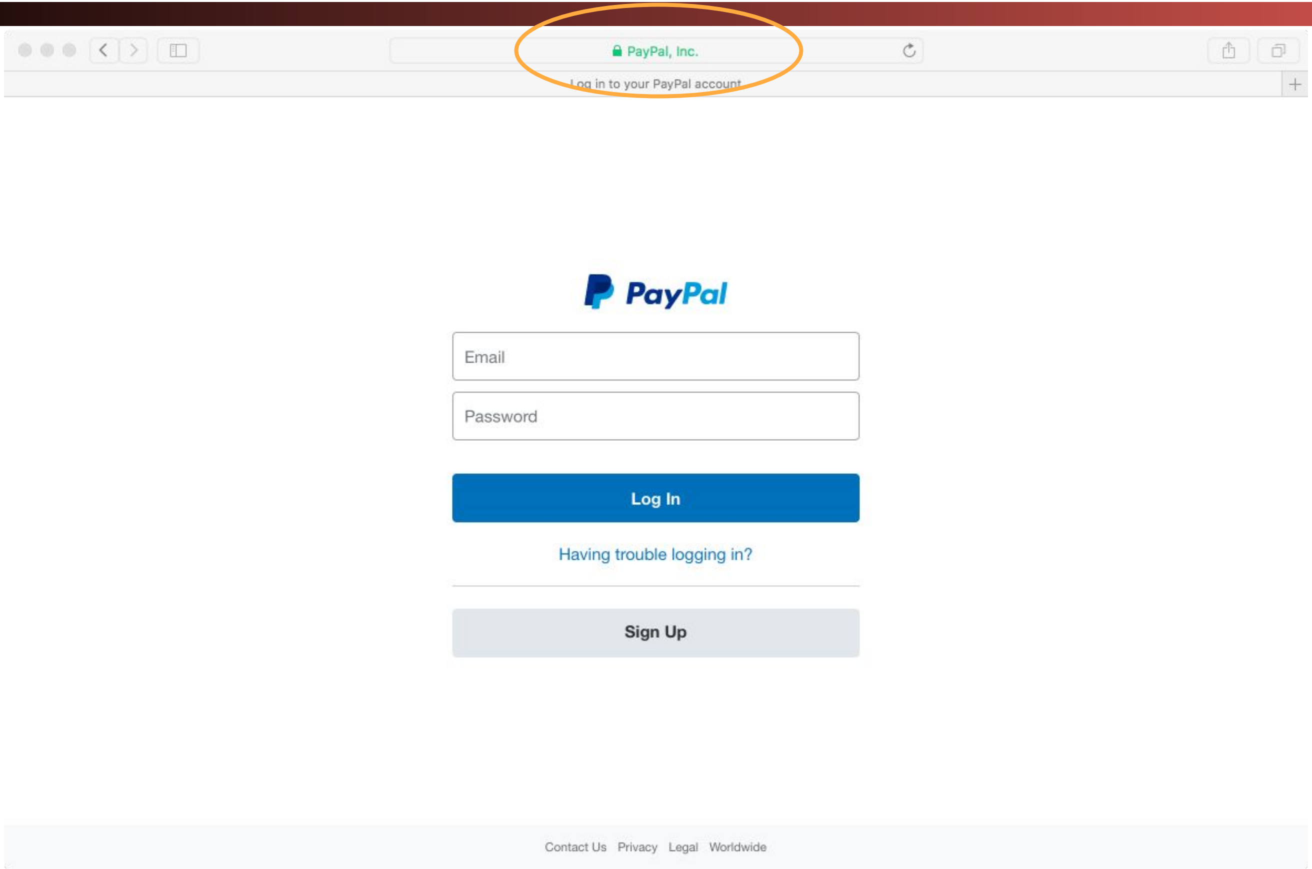
121-21-2121

□ This Card is a VBV /MSC

Continue

Your financial information is securely stored and encrypted on our servers and is not shared with merchants.

Phishing



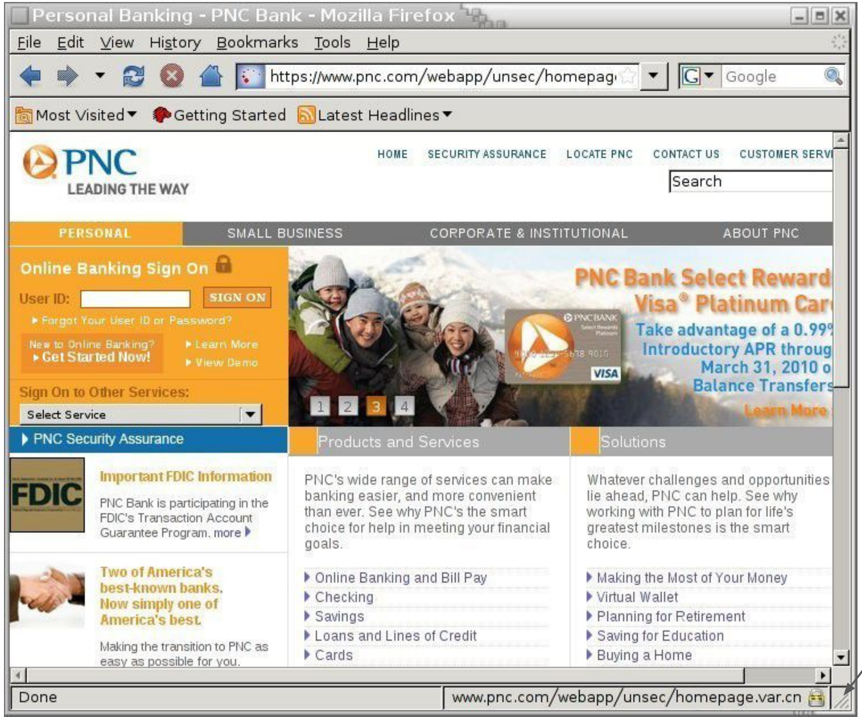
PayPal, Inc.
Log in to your PayPal account
PayPal
Email
Password
Log In
Having trouble logging in?
Sign Up
Contact Us Privacy Legal Worldwide

Phishing

- **Phishing**: Trick the victim into sending the attacker personal information
- Main vulnerability: The user can’t distinguish between a legitimate website and a website *impersonating* the legitimate website

Phishing: Check the URL?

Is this real?



Personal Banking - PNC Bank - Mozilla Firefox

https://www.pnc.com/webapp/unsec/homepag

PNC Bank Select Reward Visa® Platinum Car

Take advantage of a 0.99% Introductory APR throug March 31, 2010 o Balance Transfers

Important FDIC Information

PNC Bank is participating in the FDIC's Transaction Account Guarantee Program. more

Two of America's best-known banks. Now simply one of America's best.

Making the transition to PNC as easy as possible for you.

PNC's wide range of services can make banking easier, and more convenient than ever. See why PNC's the smart choice for help in meeting your financial goals.

Whatever challenges and opportunities lie ahead, PNC can help. See why working with PNC to plan for life's greatest milestones is the smart choice.

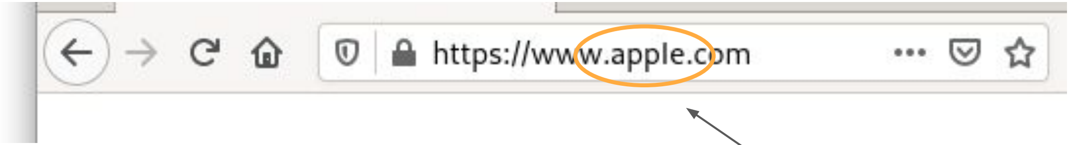
www.pnc.com/webapp/unsec/homepage.var.cn

www.pnc.com/webapp/unsec/homepage.var.cn is actually an entire domain!

The attacker can still register an HTTPS certificate for the perfectly valid domain

Phishing: Check the URL?

Is *this* real?

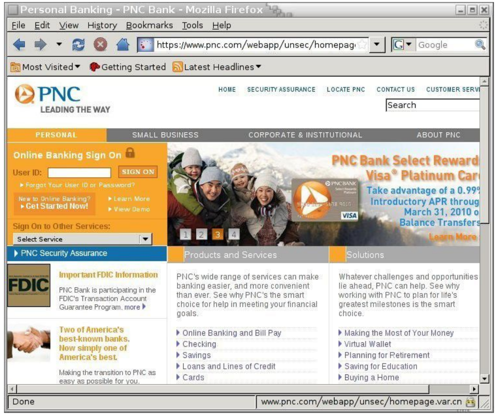


https://www.apple.com

These letters come from the Cyrllic alphabet, not the Latin alphabet! They’re rendered the same but have completely different bytes!

Phishing: Homograph Attacks

- Idea: Check if the URL is correct?
- **Homograph attack**: Creating malicious URLs that look similar (or the same) to legitimate URLs
    - Homograph: Two words that look the same, but have different meanings



https://www.pnc.com/webapp/unsec/homepag

PNC Bank Select Reward Visa® Platinum Car

Take advantage of a 0.99% Introductory APR throug March 31, 2010 o Balance Transfers

Important FDIC Information

PNC Bank is participating in the FDIC's Transaction Account Guarantee Program. more

Two of America's best-known banks. Now simply one of America's best.

Making the transition to PNC as easy as possible for you.

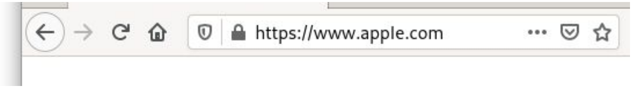
PNC's wide range of services can make banking easier, and more convenient than ever. See why PNC's the smart choice for help in meeting your financial goals.

- Online Banking and Bill Pay
- Checking
- Savings
- Loans and Lines of Credit
- Cards

Whatever challenges and opportunities lie ahead, PNC can help. See why working with PNC to plan for life's greatest milestones is the smart choice.

- Making the Most of Your Money
- Virtual Wallet
- Planning for Retirement
- Saving for Education
- Buying a Home

www.pnc.com/webapp/unsec/homepage.var.cn

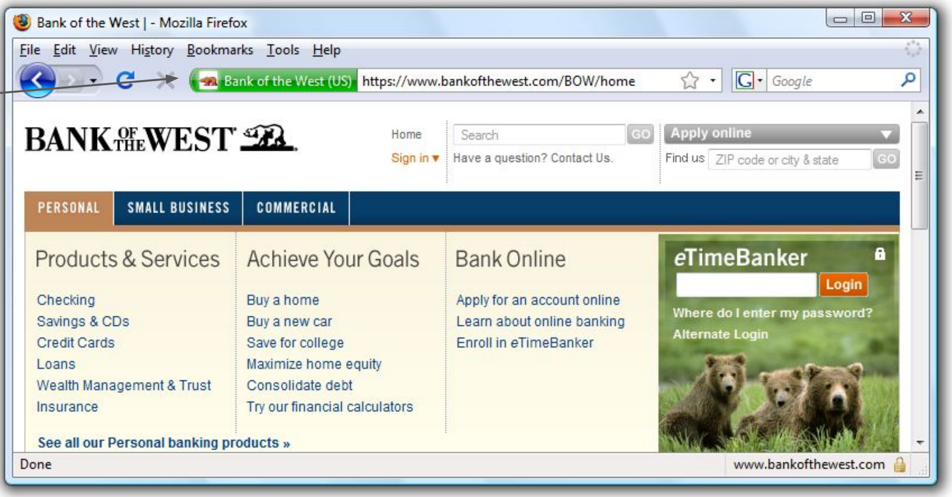


https://www.apple.com

Phishing: Check *Everything*

Is *this* real?

Extended Validation: Certificate authority verified the identity of the site (not just the domain)

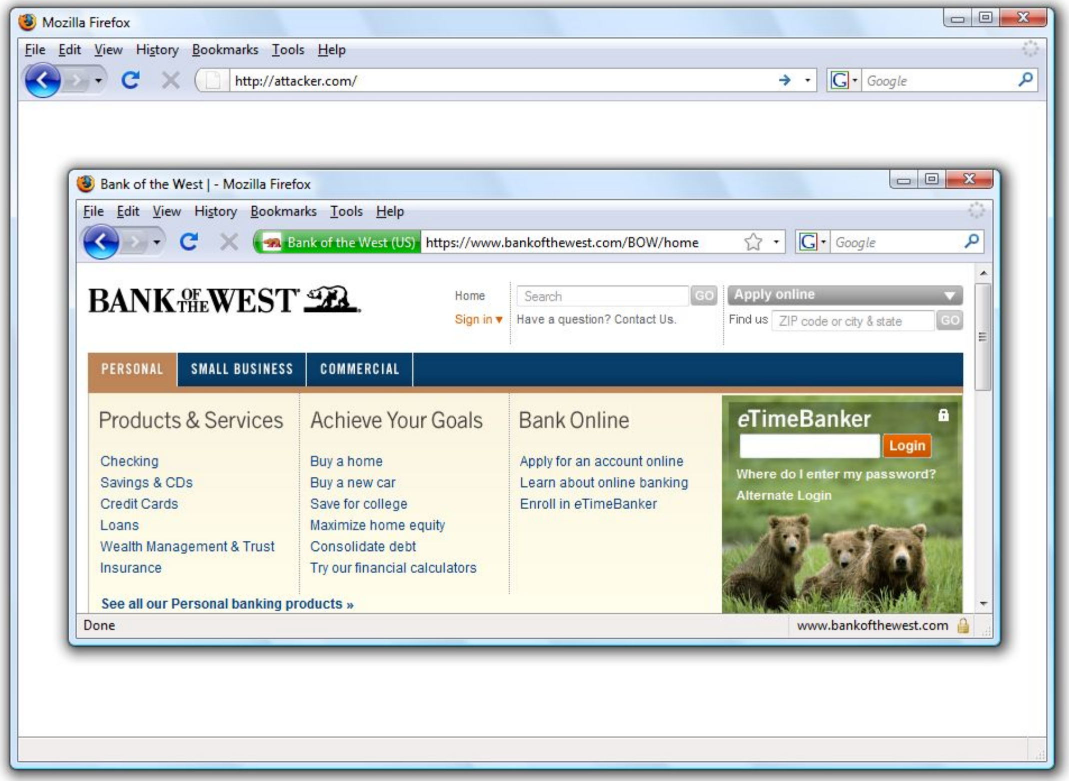

Bank of the West | - Mozilla Firefox

Bank of the West (US) https://www.bankofthewest.com/BOW/home

Done www.bankofthewest.com

Phishing: Check *Everything*

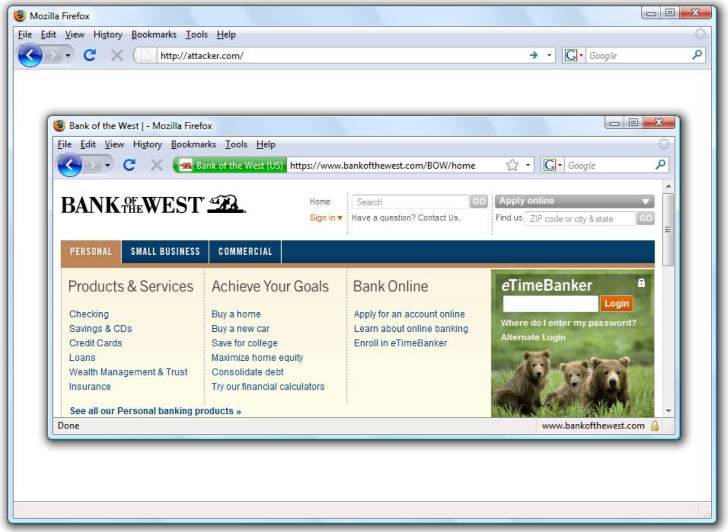
Oops, never mind



Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://attacker.com/
Google
Bank of the West | - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Bank of the West (US) https://www.bankofthewest.com/BOW/home
Google
BANK OF THE WEST
Home
Sign in ▾
Search GO
Have a question? Contact Us.
Apply online
Find us ZIP code or city & state GO
PERSONAL SMALL BUSINESS COMMERCIAL
Products & Services
Checking
Savings & CDs
Credit Cards
Loans
Wealth Management & Trust
Insurance
Achieve Your Goals
Buy a home
Buy a new car
Save for college
Maximize home equity
Consolidate debt
Try our financial calculators
Bank Online
Apply for an account online
Learn about online banking
Enroll in eTimeBanker
eTimeBanker
Login
Where do I enter my password?
Alternate Login
See all our Personal banking products »
Done
www.bankofthewest.com

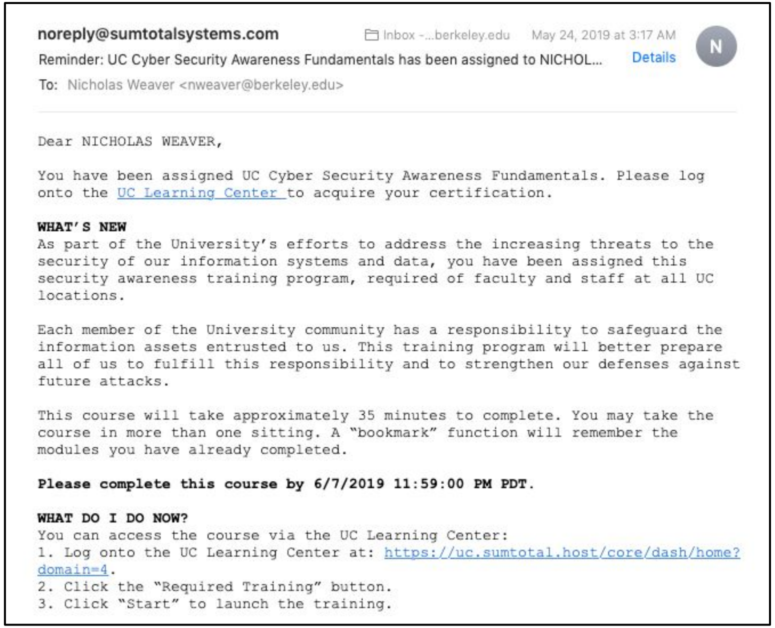
Phishing: Browser-in-browser Attacks

- Idea: Check for a green padlock icon in the browser's address bar, or any other built-in browser security feature
- **Browser-in-browser attack**: The attacker simulates the entire web browser with JavaScript


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://attacker.com/
Google
Bank of the West | - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Bank of the West (US) https://www.bankofthewest.com/BOW/home
Google
BANK OF THE WEST
Home
Sign in
Search GO
Have a question? Contact Us.
Apply online
Find us ZIP code or city & state GO
PERSONAL
SMALL BUSINESS
COMMERCIAL
Products & Services
Checking
Savings & CDs
Credit Cards
Loans
Wealth Management & Trust
Insurance
Achieve Your Goals
Buy a home
Buy a new car
Save for college
Maximize home equity
Consolidate debt
Try our financial calculators
Bank Online
Apply for an account online
Learn about online banking
Enroll in eTimeBanker
eTimeBanker
Login
Where do I enter my password?
Alternate Login
See all our Personal banking products »
Done
www.bankofthewest.com

Phishing: Don’t Blame the Users

- Most users aren’t security experts
- Attacks are uncommon: users don’t always suspect malicious action
- Detecting phishing is hard, even if you’re on the lookout for attacks
  - Legitimate messages often look like phishing attacks!



noreply@sumtotalsystems.com

Reminder: UC Cyber Security Awareness Fundamentals has been assigned to NICHOL...

To: Nicholas Weaver <nweaver@berkeley.edu>

Dear NICHOLAS WEAVER,

You have been assigned UC Cyber Security Awareness Fundamentals. Please log onto the UC Learning Center to acquire your certification.

WHAT’S NEW
As part of the University’s efforts to address the increasing threats to the security of our information systems and data, you have been assigned this security awareness training program, required of faculty and staff at all UC locations.

Each member of the University community has a responsibility to safeguard the information assets entrusted to us. This training program will better prepare all of us to fulfill this responsibility and to strengthen our defenses against future attacks.

This course will take approximately 35 minutes to complete. You may take the course in more than one sitting. A “bookmark” function will remember the modules you have already completed.

Please complete this course by 6/7/2019 11:59:00 PM PDT.

WHAT DO I DO NOW?
You can access the course via the UC Learning Center:
1. Log onto the UC Learning Center at: https://uc.sumtotal.host/core/dash/home?domain=4.
2. Click the “Required Training” button.
3. Click “Start” to launch the training.

Two-Factor Authentication

- Problem: Phishing attacks allow attackers to learn passwords
- Idea: Require more than passwords to log in
- **Two-factor authentication** (**2FA**): The user must prove their identity in two different ways before successfully authenticating
- Three main ways for a user to prove their identity
  - **Something the user knows**: Password, security question (e.g. name of your first pet)
  - **Something the user has**: Their phone, their security key
  - **Something the user is**: Fingerprint, face ID
- Even if the attacker steals the user's password with phishing, they don't have the second factor!

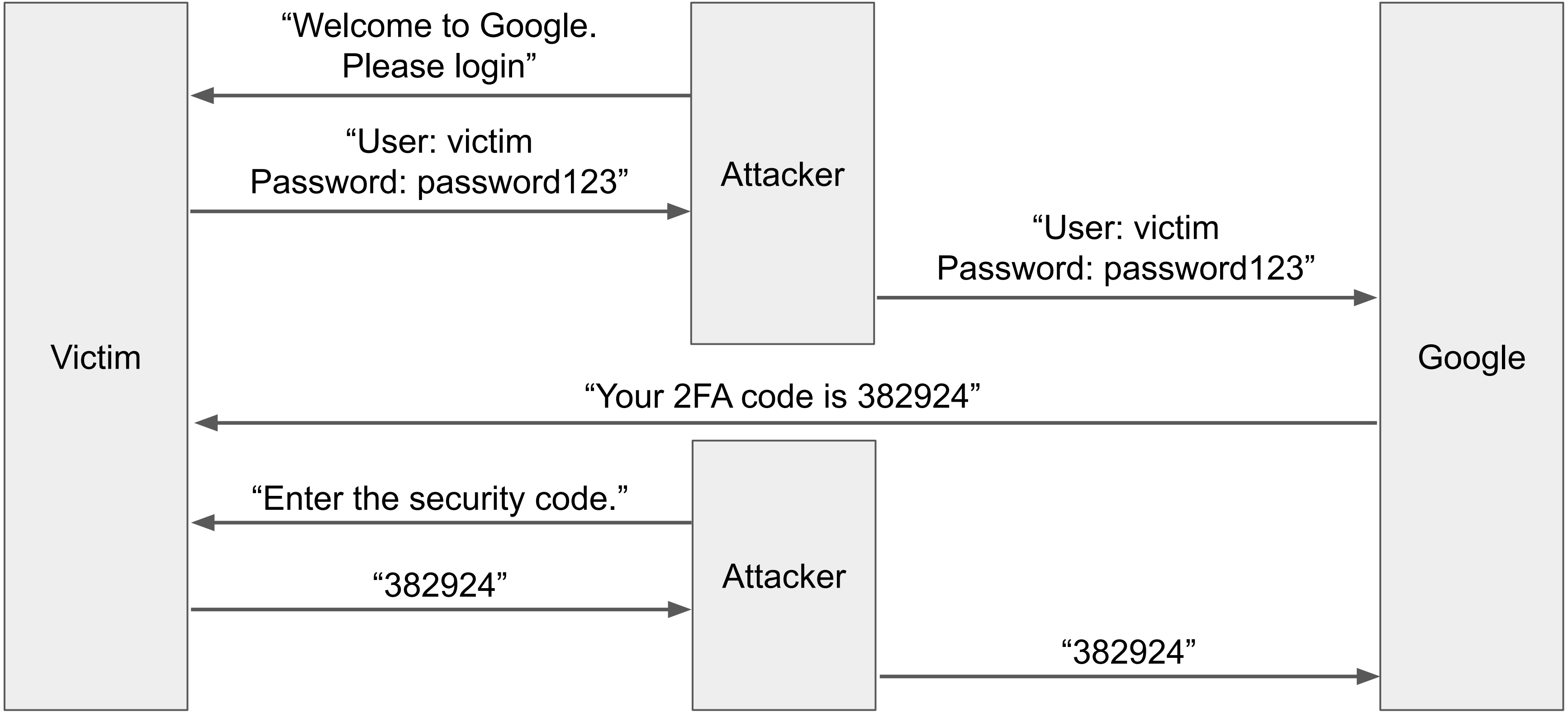
Two-Factor Authentication

- Two-factor authentication also defends against other attacks where a user’s password is compromised
  - Example: An attacker steals the password file and performs a dictionary attack
  - Example: The user reuses passwords on two different websites. The attacker compromises one website and tries the same password on the second website
  - With 2FA, the password alone is no longer enough for the attacker to log in!

Subverting 2FA: Relay Attacks

- **Relay attacks** (**transient phishing**): The attacker steals both factors in a phishing attack
- Example
  - Two-factor authentication scheme
    - First factor: The user's password (something the user knows)
    - Second factor: A code sent to the user's phone (something the user owns)
  - Attack
    - The phishing website asks the user to input their password (first factor)
    - The attacker immediately tries to log in to the actual website as the user
    - The actual website sends a code to the user
    - The phishing website asks the user to enter the code (second factor)
    - The attacker enters the code to log in as the user

Subverting 2FA: Relay Attacks



Victim
Attacker
Google
“Welcome to Google. Please login”
“User: victim Password: password123”
“User: victim Password: password123”
“Your 2FA code is 382924”
Attacker
“Enter the security code.”
“382924”
“382924”

Subverting 2FA: Social Engineering

- Some 2FA schemes text a one-time code to a phone number
  - Attackers can call your phone provider (e.g. Verizon) and tell them to activate the attacker’s SIM card, so they receive your texts!
  - 2FA via SMS is not great but *better than nothing*
- Some 2FA schemes can be bypassed with customer support
  - Attackers can call customer support and ask them to deactivate 2FA!
  - Companies should validate identity if you ask to do this (but not all do)

2FA Example: Authentication Tokens

- **Authentication token**: A device that generates secure second-factor codes
  - Something the user owns
  - Examples: RSA SecurID and Google Authenticator
- Usage
  - The token and the server share a common secret key *k*
  - When the user wants to log in, the token generates a code HMAC(*k*, time)
    - The time is often truncated to the nearest 30 seconds for usability
    - The code is often truncated to 6 digits for usability
  - The user submits the code to the website
  - The website uses its secret key to verify the HMAC
- Drawback: Vulnerable to relay attacks
- Drawback: Vulnerable to online brute-force attacks
  - Possible fix: Add a timeout

2FA Example: Security Keys

- **Security key**: A second factor designed to defend against phishing
  - Something the user owns
- Usage
  - When the user signs up for a website, the security key generates a new public/private key pair and gives the public key to the website
  - When the user wants to log in, the server sends a nonce to the security key
  - The security key signs the nonce, website name (from the browser), and key ID, and gives the signature to the server
- Security keys prevent phishing
  - In a phishing attack, the security key generates a signature with the attacker’s website name, not the legitimate website name
    - Impervious to relay attacks!

Summary: XSS

- Websites use untrusted content as control data
  - `<html><body>Hello EvanBot!</body></html>`
  - `<html><body>Hello <script>alert(1)</script>!</body></html>`
- Stored XSS
  - The attacker’s JavaScript is stored on the legitimate server and sent to browsers
  - Classic example: Make a post on a social media site (e.g. Facebook) with JavaScript
- Reflected XSS
  - The attacker causes the victim to input JavaScript into a request, and the content it’s reflected (copied) in the response from the server
  - Classic example: Create a link for a search engine (e.g. Google) query with JavaScript
  - Requires the victim to click on the link with JavaScript

Summary: XSS Defenses

- Defense: HTML sanitization
  - Replace control characters with data sequences
    - `<` becomes `&lt;`
    - `"` becomes `&quot;`
  - Use a trusted library to sanitize inputs for you
- Defense: Content Security Policy (CSP)
  - Instruct the browser to only use resources loaded from specific places
  - Limits JavaScript: only scripts from trusted sources are run in the browser
  - Enforced by the browser

Summary: Clickjacking

- Clickjacking: Trick the victim into clicking on something from the attacker
- Main vulnerability: the browser trusts the user’s clicks
  - When the user clicks on something, the browser assumes the user intended to click there
- Examples
  - Fake download buttons
  - Show the user one frame, when they’re actually clicking on another invisible frame
  - Temporal attack: Change the cursor just before the user clicks
  - Cursorjacking: Create a fake mouse cursor with JavaScript
- Defenses
  - Enforce visual integrity: Focus the user’s vision on the relevant part of the screen
  - Enforce temporal integrity: Give the user time to understand what they’re clicking on
  - Ask the user for confirmation
  - Frame-busting: The legitimate website forbids other websites from embedding it in an iframe

Summary: Phishing

- Phishing: Trick the victim into sending the attacker personal information
    - A malicious website impersonates a legitimate website to trick the user
- Don’t blame the users
    - Detecting phishing is hard, especially if you aren’t a security expert
    - Check the URL? Still vulnerable to homograph attacks (malicious URLs that look legitimate)
    - Check the entire browser? Still vulnerable to browser-in-browser attacks
- Defense: Two-Factor Authentication (2FA)
    - User must prove their identity two different ways (something you know, something you own, something you are)
    - Defends against attacks where an attacker has only stolen one factor (e.g. the password)
    - Vulnerable to relay attacks: The attacker phishes the victim into giving up both factors
    - Vulnerable to social engineering attacks: Trick humans to subvert 2FA
    - Example: Authentication tokens for generating secure two-factor codes
    - Example: Security keys to prevent phishing