SECURITY CODE REVIEW AND ANALYSIS

1. PRIPREMA

PREPORUČENO ZA ČITANJE:

1. O procesu pregledanja koda *(code review)* u Mozili *(Mozilla)*: https://blog.humphd.org/vocamus-1569/?p=1569
2. OWASP vodič za pregledanje koda: https://www.owasp.org/index.php/OWASP_Code_Review_Guide_Table_of_Contents
3. Alati za statičku analizu: https://github.com/mre/awesome-static-analysis
4. Lekcije nastale pravljenjem alata za statičku analizu u Guglu (*Google*): https://cacm.acm.org/magazines/2018/4/226371-lessons-from-building-static-analysis-tools-at-google/fulltext

2. UVOD

Cilj ove vežbe jeste pravljenje uvoda u postupak pregleda izvornog koda kako bi se pronašle bezbednosne ranjivosti u veb aplikacijama. Aplikacija koju ćemo pregledati je jednostavna PHP aplikacija, koja dozvoljava korisnicima da postave (*upload*) i preuzmu (*download*) datoteke (poput jednostavne Dropbox aplikacije).

Vlše je razloga za pregledanje koda:

- Pregledanje koda može biti brže od "*penetration testing*". Neke probleme je vrlo lako uočiti prilikom pregledanja koda, kao što je loše implementirana enkripcija. Za druge probleme, kao što je XSS, je ipak potrebno dosta više vremena;
- Usklađivanje sa standardima može zahtevati da se pregled koda izvrši (na primer, kao deo PCI DSS 6.3.2);
- "*Penetration testing*" može biti naporan, te je prikladno raditi nešto drugo;
- Neke probleme je izuzetno teško otkriti "*black-box*" testiranjem;
- Korisno je za detekciju namerno uvedenih bezbednosnih problema (*backdoors*);
- Pisanje skripte za iskorištenje konkretnog problema.

Postoji više načina kako se kod može pregledati. Neki od načina su:

- Poređenje stringova (npr. pomoću Grep alata) kako bi se pronašli problemi;
- Praćenje korisničkog unosa;
- Nasumično čitanje izvornog koda;
- Čitanje koda u celosti;
- Proveravanje različitih funkcionalnosti, jednu po jednu.

Prilikom pregledanja koda, potrebno je osmotriti sve:

- Čudno ponašanje;
- Nedostajuće provere;

- Kompleksnost;
- Implementirane bezbednosne provere;
- Razlike između dve funkcije/metode/klase);
- Poređenja i uslovi (if/else);
- Regularni izrazi/poređenje stringova;
- …

3. PRIMER

Aplikacija koju ćemo analizirati je jednostavna, i sadrži desetak bezbednosnih problema.

Korisnik može samo da:

- Se registruje/Prijavi/Odjavi;
- Postavi i preuzme datoteku.

Kod možete preuzeti:

- Koristeći git komandu: git clone https://github.com/stojkovm/cr.git
- Kao ZIP datoteku: https://github.com/stojkovm/cr/archive/master.zip

PROBLEMI

KREDENCIJALI/TAJNE U SAMOM KODU (*HARDCODED*)

Poprilično je jednostavno uočiti kredencijale ili tajne ostavljene u kodu. Na primer, sledeće datoteke sadrže osetljive informacije:

- `classes/db.php`:

```
$lnk = mysql_connect("127.0.0.1", "root", "root");
```

- `classes/jwt.php`:

```
return hash("sha256","donth4ckmebr0".$data);
```

Savremena veb aplikacija bi trebalo da skladišti svoje tajne van samog izvornog koda. Već prilikom razvoja bi tajne i kredencijali trebalo da budu injektovani u aplikaciju (kroz promenljive okruženja *(environment variables)* ili konfiguracione datoteke). Time se ograničava pristup produkcionim tajnama/kredencijalima.

CURENJE INFORMACIJA *(INFORMATION LEAK)*

Kako bi ste pronašli ovaj problem, potrebno je razmišljati o načinu na koji će aplikacija biti postavljena (*deployed*). Korenski direktorijum ove aplikacije je glavni direktorijum izvornog koda. To znači da će i sledeći direktorijumi i datoteke biti javno dostupni:

- `.git`: može biti iskorišten da se izgradi celovit izvorni kod same aplikacije
- `deploy.sql`: sadrži lozinku za bazu podataka u izvornom obliku (plaintext) i hešovanu lozinku `admin` naloga.

Ove datoteke ne bi trebalo da budu u korenskom direktorijumu veb aplikacije.

SECURITY FLAGS ON COOKIES (RAZUMEĆETE ZAŠTO NE PREVODIM OVO, KOLAČIĆI 😁)

U kodu aplikacije je iskorišteno `setcookie` kako bi se postavili *cookies* za autentikaciju. Međutim, ne postave se `secure` i `httpOnly`, niti `sameSite` naznake *(flags)* u sledećim datotekama:

- `login.php`:

```
setcookie("auth", User::createcookie($_POST['username'], $_POST['password']));
```

- `register.php`:

```
setcookie("auth", User::createcookie($_POST['username'], $_POST['password']));
```

SLAB MEHANIZAM ZA HEŠOVANJE LOZINKI

Ako pogledate `register` funkciju u `classes/user.php`:

```
public static function register($user, $password) {
    $sql = "INSERT INTO  users (login,password) values (\"";
    $sql.= mysql_real_escape_string($user);
    $sql.= "\", md5(\"";
    $sql.= mysql_real_escape_string($password);
    $sql.= "\"))";
    $result = mysql_query($sql);
```

videćete da aplikacija koristi `md5` kako bi hešovala lozinke. Takođe, hešvoanje ne uključuje nikakav dodatak *(salt, seed)*. I, konačno, samo hešovanje izvršava baza podataka (umesto da to radi sama aplikacija). To znači da će lozinke u izvornom obliku biti poslate bazi podataka, te da će verovatno završiti u logovima baze podataka. Aplikacija bi trebalo da se osloni na neke od ozbiljnijih algoritama za hešovanje lozinki, ka što su scrypt, bcrypt, PBKDF2 i Argon.

CROSS-SITE SCRIPTING

Aplikacija koristi `h` funkciju kako bi očistila (*escape*) podatke koje unosi korisnik. Ova funkcija predstavlja samo alijas za `htmlentities`, koja je definisana u `classes/phpfix.php`:

```
function h() {
    return call_user_func_array("htmlentities", func_get_args());
  }
```

Možemo primetiti da se funkcija h NE koristi na sledećim mestima:

- index.php:

```
<span class="text text-danger"><b><?php echo $error; ?></b></span>
```

- login.php:

```
<span class="text text-danger"><b><?php echo $error; ?></b></span>
```

- register.php:

```
<span class="text text-danger"><b><?php echo $error; ?></b></span>
```

Međutim, $error nije pod korisničkom kontrolom. Zbog toga su to samo slabosti, a ne i ranjivosti. Therefore, those are only weaknesses, not vulnerabilities. Ipak bi ih trebalo otkloniti kako bi aplikacija bila otpornija na budućnost *(future proof).*

XSS je zaista moguće iskoristiti u datoteci index.php:

```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post"
enctype="multipart/form-data">
```

Kako $_SERVER[‘PHP_SELF’] korisnici mogu da kontrolišu, ovde je zaista moguće izazvati XSS (koristeći putanju poput /index.php/"><script...).

Postoji još jedno mesto u classes/user.php gde se može izazvati, recimo registracijom korišćenjem istog korisničkog imena više puta (zbog poziva mysql_error).

NEDOSTAJUĆA CSRF ZAŠTITA

Ni jedna forma nije zaštićena CSRF tokenom. Ovo je primer situacije u kojoj u kodu nešto nedostaje, zbog čega to ne možemo ni da vidimo.

IZLISTAVANJE DIREKTORIJUMA

Kada korisnik zatraži listu svojih datoteka, sledeći kod se izvršava: (u `classes/user.php`):

```
public static function getfiles($user) {
    $base = "files/".$user;
    if (!file_exists($base)) {
      mkdir($base);
    }
    return array_diff(scandir($base), array('..', '.'));
  }
```

Ovaj kod dobavlja listu datoteka (scandir) u files/[USERNAME] i uklanja roditeljski (..) i trenutni (.) direktorijum iz liste koristeći array_diff.

Ništa u ovom kodu (niti prilikom registracije) ne sprečava napadača da iskoristi sledeća korisnička imena: .., ../.., ../../.. Ovim se omogućava izlistavanje bilo kog direktorijuma na serveru gde je aplikacija.

KRIPTOGRAFSKI PROBLEM

Kriptografski problem se nalazi u datoteci `classes/jwt.php`, u funkciji za potpisivanje:

```
public static function signature($data) {
    return hash("sha256","donth4ckmebr0".$data);
  }
```

Možemo videti da aplikacija koristi heš funkciju umesto hmac funkcije. Time se kreira više problema:

- Funkcija je vrlo verovatno ranjiva na *Length Extension* napade;
- To nije po RFC standardu za JWT tokene. Gubi se kompatibilnost sa drugim aplikacijama.

ZAOBILAŽENJE POTPISA

Propust koji omogućava zaobilaženje potpisa se nalazi u `classes/jwt.php`:

```
public static function verify($auth) {
    list($h64,$d64,$sign) = explode(".",$auth);
    if (!empty($sign) and (JWT::signature($h64.".".$d64) != $sign)) {
      die("Invalid Signature");
    }
```

Možemo videti da aplikacija dobavlja tri elementa `$auth` promenljive: `$h64`, `$d64` i `$sign`. Zatim proverava potpis `$sign` koristeći: `JWT::signature($h64.".".$d64) != $sign`. Međutim, proverava samo ako je potpis tu: `if (!empty($sign) and...` . Napadač može da kreira malicioznu `$auth` promenljivu koja ne sadrži potpist. Ovim se zaobilazi mehanizam provere potpisa.

ZAOBILAŽENJE AUTORIZACIJE

Vrlo je jednostavno zaobići autorizaciju. Kada se datoteka postavi, sačuva se u sledećem direktorijumu: `/files/[USERNAME]/[FILENAME]`:

```
public static function addfile($user) {
    $file = "files/".$user."/".basename($_FILES["file"]["name"]);
    if (!preg_match("/\.pdf/", $file)) {
      return  "Only PDF are allowed";
    } elseif (!move_uploaded_file($_FILES["file"]["tmp_name"], $file)) {
      return "Sorry, there was an error uploading your file.";
    }
```

Trivijalno je pogoditi `[USERNAME]` i `[FILENAME]` kako bi se dobio pristup datotekama drugih korisnika. Potrebno je ili dodati mehanizam koji proverava da li je pristup dozvoljen, ili načiniti linkove takvim da ih je teško pogađati.

REMOTE CODE EXECUTION

Mogućnost izazivanja RCE je u `addfile` funkciji:

```
public static function addfile($user) {
    $file = "files/".$user."/".basename($_FILES["file"]["name"]);
    if (!preg_match("/\.pdf/", $file)) {
      return  "Only PDF are allowed";
    } elseif (!move_uploaded_file($_FILES["file"]["tmp_name"], $file)) {
      return "Sorry, there was an error uploading your file.";
    }
```

Već znamo da se datoteke skladište u korenskom direktorijumu. Takođe, možemo videti da su korisnici limitirani tako da mogu da postave samo PDF datoteke. Međutim, iskorištenom regularnom izrazu nedostaje znak `$` kako bi se prepoznao i kraj stringa. Napadač može postaviti datoteku `exec.pdf.php`. Kako je ekstenzija datoteke `.php` i ona se nalazi u korenskom direktorijumu, napadač može izazvati njeno izvršavanje pristupom.

ZADATAK:

1. Svaki student bi trebalo da odabere jedan projekat implementiran svom prethodnom akademskom ili profesionalnom okruženju i da odradi ručni pregled koda. Takođe bi nad kodom projekta trebalo da pokrene neki od alata za statičku analizu preporučenih tačkom 3.
2. Svaki student bi trebalo da napiše izveštaj (od maksimalno 3 strane) koji uključuje:
    a. Opis projekta na visokom nivou;
    b. Listu članova razvojnog tima;
    c. Opis pronađenih defekata;

d. Sažetak preporuka kojima se kod poboljšava;
e. Vreme koje je proveo svaki od članova tima *(reviewer)* pregledajući kod i broj defekata koji je identifikovao/la, radi.