VISOKA ŠKOLA ELEKTROTEHNIKE I RAČUNARSTVA

PRIMENA ZAŠTITNIH MEHANIZAMA NA SISTEMU ZA PRIJAVLJIVANJE

Beograd, septembar 2020.

Predmetni saradnik:
Veselin Ilić

Studenti:
Bojan Stojković RIN-44/20
Milovan Srejić RIN-70/20

SADRŽAJ

1.UVOD

U okviru ovog rada predstavićemo naše rešenje zaštite sistema za prijavljivanje od poznatih napada koji predstavljaju pretnju u savremenim *web* aplikacijama. Napadi od kojih se branimo su *brute-force attack* (napad grubom silom) i *SQL Injection* propust. Kao pokazni primer kreirana je aplikacija unutar koje postoji mogućnost simulacije napada na sistem koji ima obezbeđene sigurnosne mehanizme kao i na onaj koji je ranjiv. Sistem zaštite je realizovan na nivou aplikacije, programiranjem u *ASP.NET Core* tehnologiji. Sigurnosne mehanizme, algoritme po kojima rade, sami smo implementirali koristeći funkcionalnosti koje nam pružaju biblioteke programskog jezika *C#*. Radi lakše sinhronizacije i saradnje koristili smo deljeni repozitorijum na *Github*-u. Link do *Github* repozitorijuma: https://github.com/milovan1234/Security_Project.

2.OPIS SIGURNOSNOG PROPUSTA

U ovom poglavlju opisaćemo osnove i načine funkcionisanja *SQL Injection* propusta i *Brute-force Attack*-a.

2.1.SQL INJECTION

SQL Injection predstavlja direktan napad na aplikaciju, odnosno na bazu podataka. Cilj ovog napada je da se izmeni određeni *SQL* upit kako bi se izvršile razne akcije, od dobijanja aletrantivnih podataka, do izmene ili brisanja podataka iz baze podataka. Napad *SQL Injection*-om se uglavnom vrši na skripte za proveru autentifikacije korisnika, odnosno prilikom provere korisničkog imena i lozinke, kako bi se *SQL* upit izmenio i uvek izvršio tako da uvek "dohvata" podatke o nekom korisniku. Osim kod provere autentifikacije, može se iskoristiti i u druge zlonamerne svrhe, a jedan od njih je ubacivanje dodatnog upita koji može uništiti podatke ili neke druge zlonamerne akcije.

O ovom propustu programer mora voditi računa, pre svega misli se na način na koji se dohvataju podaci iz baze podataka. Loša praksa je da se provera autentifikacije prepusti samoj bazi, mnogo je bolje "izvući" podatke iz baze, a onda u samoj aplikaciji proveriti da li su podaci ispravni za traženog korsinika. Problem nastaje kada programer sastavi upit korišćenjem klasične konkatenacije, gde ne postoji nikakav mehanizam provere šta je to korisnik uneo kao podatak. Zlonamerni korisnik unošenjem dodatnih specijalnih karaktera i ključnih reči može izmeniti logiku samog upita.

Najbolja prevencija ovog propusta je korišćenje parametrizovanih upita ili nekih pomoćnih funkcija koje mogu da izvrše filtriranje ili proveru samog upita da li je zlonameran ili nije. Ključna stvar kod parametrizovanih upita je što se ne vrši klasična konkatenacija, već se pri pozivu funkcije mapiraju uneti podaci na odgovarajući način u zavisnosti od tipa podatka koji se očekuje u parametru, gde se onemogućava izvršavanje dodatne logike na tom mestu.

2.2.BRUTE-FORCE

Brute-force napad ili napad uzastopnim pokušavanjem je jednostavna, ali uspešna tehnika rešavanja problema koja se sastoji od sistematskog pronalaženja svih mogućih kandidata za rešenje i isprobavanja svakog od njih. *Brute-force* napad je jednostavan za implementaciju i često se koristi za probijanje lozinki ili raznih enkripcija , a osim toga, on uvek i pronalazi rešenje, ako ono postoji. Međutim, vreme i resursi potrebni za rešavanje problema ovom metodom rastu proporcionalno broju mogućih kandidata za rešenje. Zbog toga se algoritam napada često optimizuje heurističkim dodacima koji bitno umanjuju broj mogućih kandidata. *Brute-force* napadi se obično provode na jedan od sledeća tri načina:

1. Ručnim pokušajima autentifikacije uzastopnim unošenjem korisničkih imena i lozinki.

2. Napadima rečnikom pomoću automatizovanih skripti i programa koje unose hiljade raznih kombinacija korisničkih imena i lozinki iz rečnika.

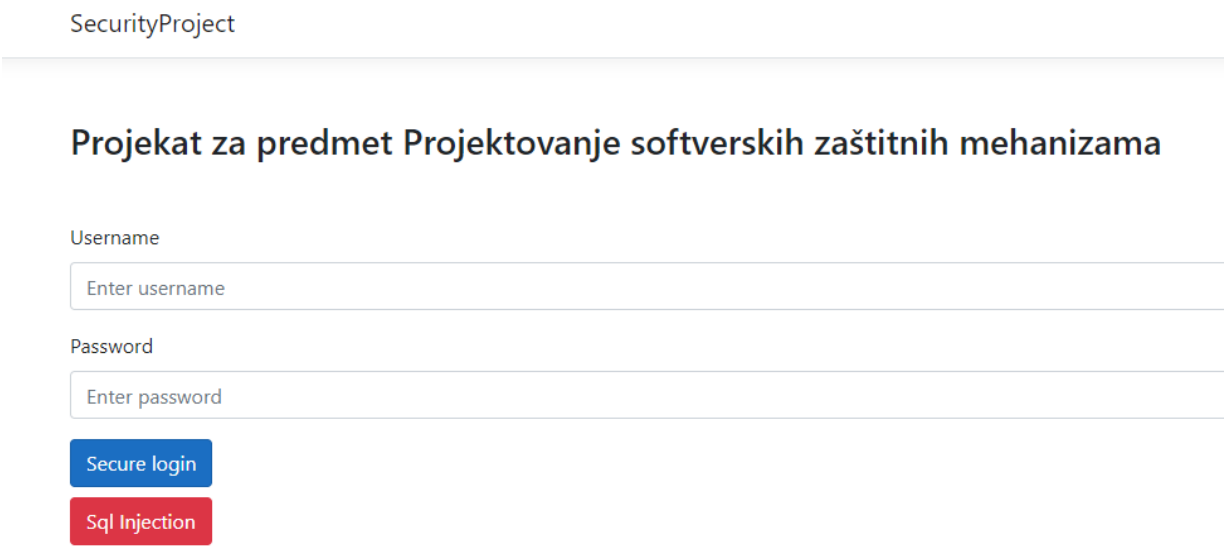
3. Generisanim korisničkim podacima - zlonamerno oblikovanim programom generišu se slučajna korisnička imena i lozinke pomoću kojih se pokušava da pristupi sistemu.

U ovom radu predstavljen je napada *Brute-force* primenom rečnika najčešće korišćenih lozinki. Napad rečnikom (*Dictionary attack*) je tehnika za probijanje šifri ili autentifikacije kod koje se pokušava pronaći tražena lozinka ili tajni ključ uzastopnim isprobavanjem velikog broja reči ili kombinacija reči. Napad rečnikom je varijacija *brute-force* napada kod koje je izbor mogućih kandidata sužen sa skupa svih mogućih kombinacija slova na one kombinacije koje imaju neko značenje na određenom jeziku. Izbor je logičan jer je veća verovatnoća da je lozinka ili ključ, neki skup slova koji ima značenje nego neki niz znakova koji korisniku ništa ne znači i koji mu je zbog toga teško pamtljiv.

3.REALIZACIJA ZAŠTITNIH MEHANIZAMA I NAPADA

Kroz konkretan primer aplikacije biće objašnjeni primenjeni zažtitni mehanizmi za sprečavanje gore pomenutih napada. Aplikacija je zamišljena kao mini *web* aplikacija, tj. poseduje formu za prijavu, kroz koju je moguće simulirati pomenute napade i uporediti uspešnost prijavljivaja kod zaštićenog i ne zaštićenog sistema. Pored ove aplikacije realizovana je još jedna manja aplikacija koja omogućava napad rečnikom. Obe aplikacije su izrađene u *C#* programskom jeziku.

Aplikacija je izrađena po *MVC (Model-View-Controller)* principu. Postoji jedan *View* koji sadrži formu za prijavu kao i opcije za odabir koju vrstu prijave želimo. Možemo odabrati *Secure Login* ili *Login* sa *SQL Injection* propustom. Unutar aplikacije se nalazi kontroler koji vrši obradu pomenutih zahteva. Aplikacija koristi podatke iz *MSSQL Server* baze podataka. Ukoliko je prijavljivanje uspešno servis vraća *status code* 200 (*OK*), u suprotnom vraća *status code* 400 (*Bad Request*).



Slika 1. Izgled aplikacije

3.1. SQL INJECTION

U kontroleru postoje dve metode za obradu zahteva za prijavljivanje. Kod prve metode korišćena je klasična konkatenacija za kreiranje upita kojim se proverava da li postoji tražena kombinacija *username-password* u bazi podataka. Ukoliko bi zlonamerni korisnik uneo za password string nalik sledećem **nesto' or '1' = '1** konkatenacijom na sam upit izmenila bi se logika samog upita i on bi zbog logičkog operatora *OR* (logičko ILI) uvek vraćao informaciju da traženi korisnik postoji.

```
1.  SqlCommand sqlCommand = new SqlCommand("SELECT COUNT(*) FROM Users where Username='
    " + user.Username + "' AND Password='" + user.Password + "'", connection);
2.  int login = (int)sqlCommand.ExecuteScalar();
3.  if (login == 0)
4.  {
5.      return BadRequest();
6.  }
7.  else
8.  {
9.      return Ok();
10. }
```

Izvorni kod metode koja je podložna SQL Injection napadu

Pored ove metode, postoji metoda koja osigurava bezbedno prijavljivanje. U ovoj metodi upotrebljen je parametrizovan *SQL* upit. U programsom jeziku C# parametrizovani upiti se prave tako što se u stringu upita na mesto gde treba postaviti neku vrednost koju korisnik unese postavlja naziv parametra sa prefiksom **@**. Odgovarajućom metodom parametri se zamenjuju sa podacima koje je korisnik uneo. Na ovaj način onemogućuje se izmena logike upita te je ovakav upit otporan na *SQL Injection*.

```
1.  SqlCommand command = new SqlCommand("SELECT COUNT(*) FROM Users WHERE Username=@use
    rname AND Password=@password",connection);
2.  command.Parameters.AddWithValue("@username", user.Username);
3.  command.Parameters.AddWithValue("@password", user.Password);
4.  int login = (int)command.ExecuteScalar();
5.  if(login == 0)
6.  {
7.      command = new SqlCommand("UPDATE USERS set NumberOfInvalidLogin = NumberOfInval
    idLogin + 1 where Username = @username",connection);
8.      command.Parameters.AddWithValue("@username", user.Username);
9.      command.ExecuteNonQuery();
10.     command = new SqlCommand("UPDATE USERS set LastInvalidLogin = @datetime",connec
    tion);
11.     command.Parameters.AddWithValue("@datetime", DateTime.Now);
12.     command.ExecuteNonQuery();
13.     return false;
14. }
15. else
16. {
17.     command = new SqlCommand("UPDATE USERS set NumberOfInvalidLogin = 0 where Usern
    ame = @username",connection);
18.     command.Parameters.AddWithValue("@username", user.Username);
19.     command.ExecuteNonQuery();
20.     return true;
21. }
```

Izvorni kod metode koja nije podložna SQL Injection napadu

Ova metoda je deo SecureLogin metode i kao takva sadrži dodatne naredbe koje služe za odbranu od Brute Force napada koji ćemo opisati u narednom poglavlju.

3.2.BRUTE-FORCE

Kako bi simulirali Brute-force uz pomoć rečnika najčešće korišćenih loziniki, napravili smo aplikaciju koja čita iz datoteke red pod red i napada servis. Potrebno je uneti korisničko ime i pokrenuti napad. U bazi podataka je napravljen korisnik čija je lozinka „*password*“ kako bi imali primer za uspešan napad.

Da bi se odbralinli od Brute-force –a napravili smo dva mehanizma za odbranu. Prvi mehanizam se bazira na tome da se u tabeli, u bazi podataka, za svaki neuspešni pokušaj prijave korisnika beleži koliko je puta zaredom probao da se prijavi. Kako je ovo deo SecureLogin metode, ovaj deo koda je integrisan uz sigurnu SQL proveru za postojanje korisnika. Može se videti na prethodnoj strani. Kada se dostigne maksimalni broj neuspešnih uzastopnih prijava, prijava za tog korisnika postaje onemogućena. U tom trenutku u bazu se upisuje tačano vreme kada je korisnik zadnji put pokušao da se prijavi. Sledeći put kada korisnik pokuša da se prijavi, proveri se da li je razlika između vremena kada se zadnji put bezuspešno prijavio i trenutnog veća od vremena koje je definisano kao trajanje blokade. Ako je prošlo određeno vreme broj neuspešnih prijava u bazi za tog korisnika se resetuje na nulu.

```
1.  SqlCommand command = new SqlCommand("select NumberOfInvalidLogin from Users where U
    sername = @username",connection);
2.  command.Parameters.AddWithValue("@username", user.Username);
3.  int login = (int)command.ExecuteScalar();
4.  if(login == MAX_NUMBER_OF_FAULT_LOGINS)
5.  {
6.      command = new SqlCommand("SELECT LastInvalidLogin FROM Users WHERE Username=@us
    ername",connection);
7.      command.Parameters.AddWithValue("@username", user.Username);
8.      DateTime banTime = (DateTime)command.ExecuteScalar();
9.      if(DateTime.Now - banTime > TIME_TO_WAIT_BETWEEN_FAULT_LOGIN)
10.     {
11.         command = new SqlCommand("UPDATE USERS set NumberOfInvalidLogin = 0 WHERE U
    sername=@username",connection);
12.         command.Parameters.AddWithValue("@username", user.Username);
13.         command.ExecuteNonQuery();
14.         return true;
15.     }
16.     else
17.     {
18.         return false;
19.     }
20. }
21. else
22. {
23.     return true;
24. }
```

Izvorni kod metode koja proverava neispravan broj prijava korisnika

Drugi vid zaštite je osmišljen kao filter na servisnoj metodi koji će se izvršiti pre same servisne metode. Proverava se da li je za kratki vremenski period došlo previše zahteva sa iste *IP* adrese, ukoliko je to slučaj blokiramo tu *IP* adresu na određeno vreme i korisnik/napadač koji šalje ovakve zahteve je blokiran da pristupi prijavi sa bilo kojom kombinacijom *username-password.* Ukoliko je neka od IP adresa blokirana korisniku se vraća odgovor u vidu *status code*-a 403 (*Forbidden*). U nastavku sledi izvorni kod ovog filtera.

```
1.  public class ClientIpCheckActionFilter : ActionFilterAttribute
2.  {
3.      public static Dictionary<string, RequestDetails> requests = new Dictionary<stri
    ng, RequestDetails>();
4.      private TimeSpan SECONDS_BETWEEN_REQUESTS = TimeSpan.FromSeconds(120);
5.      private TimeSpan MINUTES_FOR_BLOCKED = TimeSpan.FromMinutes(1);
6.      private const int MAX_NUMBER_OF_REQUEST = 1;
7.      public override void OnActionExecuting(ActionExecutingContext context)
8.      {
9.          string remoteIpAddress = context.HttpContext.Connection.RemoteIpAddress.ToS
    tring();
10.         RequestDetails requestDetails = requests.FirstOrDefault(x => x.Key.Equals(r
    emoteIpAddress)).Value;
11.         if(requestDetails == null)
12.         {
13.             requestDetails = new RequestDetails()
14.             {
15.                 NumberOfRequests = 0,
16.                 LastRequest = DateTime.Now,
17.                 isBlocked = false
18.             };
19.             requests.Add(remoteIpAddress, requestDetails);
20.         }
21.         else
22.         {
23.             CheckBlockedIpAddresses(remoteIpAddress);
24.             if(requestDetails.isBlocked)
25.             {
26.                 context.Result = new StatusCodeResult(StatusCodes.Status403Forbidde
    n);
27.                 return;
28.             }
29.             requestDetails.LastRequest = DateTime.Now;
30.             if ((DateTime.Now -
     requestDetails.LastRequest) < SECONDS_BETWEEN_REQUESTS)
31.             {
32.
33.                 requestDetails.NumberOfRequests += 1;
34.                 requestDetails.isBlocked = requestDetails.NumberOfRequests > MAX_NU
    MBER_OF_REQUEST;
35.             }
36.         }
37.         base.OnActionExecuting(context);
38.     }
39.
40.     private void CheckBlockedIpAddresses(string key)
41.     {
42.         RequestDetails requestDetails = requests[key];
43.         if (requestDetails.isBlocked)
44.         {
45.             requestDetails.isBlocked = ((DateTime.Now -
     requestDetails.LastRequest) < MINUTES_FOR_BLOCKED);
46.             requestDetails.NumberOfRequests = requestDetails.isBlocked ? requestDet
    ails.NumberOfRequests : 0;
47.         }
48.     }
49. }
```

Izvorni kod filtera IP adresa

3.3.SECURELOGIN METODA

Gore opisani mehanizmi zaštite integrisani su u okviru *SecureLogin* metode na *Login* kontroleru. Pre samog izvršavanja metode poziva se *Filter IP* adresa, ukoliko se ispostavi da *IP* adresa nije blokirana metoda se izvršava. Metoda prvo poziva metodu *CheckNumberOfFaultLogin* koja proverava da li je omogućena prijava za traženog korisnika. Ukoliko ova metoda vrati *true*, proverava se da li tražena kombinacija *username-password* postoji u bazi metodom *SafeSqlCheck* koja koristi parametrizovan *SQL* upit.

```
1.  [ServiceFilter(typeof(ClientIpCheckActionFilter))]
2.  [HttpPost]
3.  public IActionResult SecureLogin([FromBody] User user)
4.  {
5.      bool isLoginNumberOK = CheckNumberOfFaultLogin(user);
6.      if (!isLoginNumberOK)
7.      {
8.          return BadRequest();
9.      }
10.     bool isUserFound = SafeSqlCheck(user);
11.     if (isUserFound)
12.     {
13.         return Ok();
14.     }
15.     else
16.     {
17.         return BadRequest();
18.     }
19. }
```

Izvorni kod SecureLogin metode

4.ZAKLJUČAK

U današnje vreme postoji velika potreba za zaštitom sistema za autentifikacju, koji predstavljaju najčešću metu zlonamernih korisnika. Ovim radom proširli smo naša znanja iz oblasti sigurnosti i pokušali da na naš način osmislimo sigurnosne mehanizme od nekih poznatih napada. Svesni smo da postoje bolja rešenja, ali je rad na praktičnoj realizaciji nekog ovakvog sistema dosta doprineo da shvatimo kako bi neki od već postojećih mehanizama zaštite mogao da funkcioniše.