SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Diplomski rad br. 1823

Automatizirana provjera ranjivosti računalnih sustava

Goran Ličina

Zagreb, studeni 2009.

Sažetak

U ovom diplomskom radu navedene su suvremene sigurnosne prijetnje, kao i tehnike koje napadači koriste prilikom ugrožavanja sigurnosti računalnih sustava. Opisan je i slobodno raspoloživi programski alat za provjeru ranjivosti računalnih sustava – OpenVAS i njegove mogućnosti. Sigurnosni testovi za OpenVAS izrađeni su u programskom jeziku NASL. U praktičnom dijelu rada u NASL-u su ostvarene programske skripte za detekciju operacijskog sustava i drugih servisa na udaljenim računalima.

Abstract

This diploma thesis introduces latest security threats and techniques that malicious attackers use in endangering computer systems security. It also describes open-source tool used for vulnerability scanning - OpenVAS and his capabilities. Security tests for OpenVAS are developed in Nessus Attack Scripting Language. In practical part of this thesis, NASL was used for development of operating system detection script and several other service detection scripts.

Zahvaljujem se svojoj obitelji i djevojci Luciji na podršci koju mi svakodnevno pružaju, svim prijateljima i kolegama na pomoći prilikom izrade ovog rada te mentoru doc.dr.sc. Marinu Golubu na stručnom vodstvu.

Sadržaj

1. Uvod

U skladu s razvojem računalnih sustava i tehnologija, svakim danom pojavljuje se sve više sigurnosnih prijetnji. Tehnike koje napadači koriste sve su sofisticiranije i opasnije. Napadači imaju u svojim rukama cijelu zbirku mogućih napada – od klasičnih napada koji iskorištavaju ranjivosti u raznim poslužiteljskim aplikacijama, preko naprednih DoS napada, pa sve do raznih napada na *web* aplikacije. Važan korak u planiranju napada je prikupljanje informacija o ciljnim sustavima, no i ovdje postoji cijeli niz tehnika koje napadačima olakšavaju planiranje napada. Sve ove činjenice otežavaju posao sistemskim administratorima koji pokušavaju maksimalno osigurati svoja računala i mreže. Njihov cilj je pružiti potencijalnim napadačima što manje informacija, a da se istodobno ne gubi na ugodnosti korištenja njihovih sustava, što često nije lagan zadatak.

Jedan od načina povećanja sigurnosti računalnih sustava je redovita automatizirana provjera ranjivosti. Radi se o postupku koji uz pomoć velikog broja sigurnosnih testova pokušava pribaviti informacije i izvesti napade koje bi mogao izvesti zlonamjerni napadač. Time se dobiva realna slika sigurnosti neke računalne mreže i sustava koji se na njoj nalaze. Testovi se obično provode prema poznatim ranjivostima raznih paketa i sustava, no mogu se izvoditi i razni drugi napadi poput napada grubom silom (eng. *brute force).*

U ovom diplomskom radu opisane su suvremene sigurnosne prijetnje računalnim sustavima te tehnike koje napadači koriste. Kao važan dio svakog napada posebno je opisana faza prikupljanja informacija o računalnim sustavima.

Opisan je i slobodno raspoloživ alat za provjeru ranjivosti računalnih sustava – OpenVAS, te programski jezik NASL koji se koristi za pisanje sigurnosnih testova. U praktičnom dijelu ovog diplomskog rada ostvarene su tri programske skripte. Jedna za detekciju operacijskog sustava udaljenog računala, zatim za detekciju servisa *rsync* te *web* aplikacije WebAPP.

2. Metode prikupljanja informacija o računalima

2.1 Metodologija prikupljanja informacija

Prikupljanje informacija dijeli se u sedam logičnih koraka:

- Otkrivanje inicijalnih informacija
- Otkrivanje mrežnih adresa
- Utvrđivanje aktivnih računala
- Otkrivanje otvorenih mrežnih pristupa (eng. *port*)
- Detekcija operacijskih sustava
- Otkrivanje aktivnih servisa
- Crtanje mrežnog dijagrama

2.2 Inicijalno prikupljanje informacija

Inicijalno prikupljanje informacija je dio pripremne faze napada na neki sustav, a uključuje sakupljanje podataka vezanih za okruženje i arhitekturu ciljnog sustava. Obično se koristi u svrhu pronalaženja najlakšeg načina za upad u sustav. Prvi korak je utvrđivanje ciljnog sustava, aplikacija na sustavu i fizičke lokacije cilja. Kada su te informacije poznate, specifične informacije vezane uz ciljnu organizaciju se istražuju koristeći takozvane "nenametljive" metode. Na primjer, *web* stranica organizacije može sadržavati osobne direktorije i biografije zaposlenika koje napadač može iskoristiti za socijalni inženjering. Napadač također može koristiti tražilice poput Google-a ili Yahoo-a za dobavljanje podataka o zaposlenicima.

Google tražilica često se koristi pri dobavljanju informacija te se taj način prikupljanja informacija često naziva "Google hakiranje". Neke od sljedećih ključnih riječi mogu se koristi kod pretrage Google tražilicom:

- *site* - pretražuje određenu *web* stranicu ili domenu
- *filetype* – traži specifične vrste datoteka,
- *intitle* – pretražuje izraz unutar naslova dokumenta,
- *inurl* – pretražuje samo unutar URL-a (eng. *Uniform Resource Locator*) dokumenta.

Ovim putem mogu biti otkrivene i druge informacije koje uključuju identifikaciju korištenih Internet tehnologija i operacijskih sustava, aktivnih IP adresa, *e-mail* adresa, brojeva telefona, kao i korporativnih sigurnosnih politika.

2.3 Enumeracija DNS protokolom

Enumeracija DNS protokolom je proces lociranja svih DNS poslužitelja i odgovarajućih zapisa za organizaciju nad kojom se izvodi napad. Organizacija može imati vanjske i unutarnje DNS poslužitelje koji mogu otkriti informacije kao što su korisnička imena, imena računala u mreži, te IP adrese potencijalnih ciljnih sustava.

Jedan od najčešće korištenih alata za provođenje DNS enumeracije je *nslookup*. Ovaj alat koristi se za slanje DNS upita poslužiteljima i dostupan je na Unix, Linux i Windows platformama. Primjer pretrage ovim alatom prikazan je na slici 2.1.

```
$ nslookup google.com
Server:         192.168.88.2
Address:        192.168.88.2#53

Non-authoritative answer:
Name:   google.com
Address: 74.125.67.100
Name:   google.com
Address: 74.125.45.100
Name:   google.com
Address: 74.125.53.100
```

Slika 2.1. Korištenje programa nslookup

Osim alata *nslookup*, još jedan alat koristan za prikupljanje podatak je *whois*. Informacije pronađene *whois* pretragom mogu se iskoristiti za detaljniju DNS pretragu. Time mogu biti otkrivene IP adrese poslužitelja kao i drugih krajnjih računala u mreži organizacije koja je cilj pretrage. Whois pretraga izvodi se istoimenim alatom, koji je danas rasprostranjen u mnogim operacijskim sustavima, kao i specijaliziranim alatima za pretragu. Ovaj alat pomaže u prikupljanju informacija o vlasniku određene domene spajajući se na razne internetske baze podataka. Whois pretragama mogu se pronaći informacije o lokaciji vlasnika, imenima odgovornih osoba i administratora, te imenima DNS poslužitelja zaduženih za domenu.

Još jedan od popularnih izvora informacija svakako su Internet registri. Postoje četiri Internet registra od kojih je svaki zadužen za jedan dio svijeta i to kako slijedi:

- ARIN (eng. *American Registry for Internet Numbers*) – za Sjevernu Ameriku i dijelove Afrike
- RIPE NCC (eng. *Réseaux IP Européens Network Coordination Centre*) – za Europu, Bliski Istok i dijelove Centralne Azije
- LACNIC (eng. *Latin American and Caribbean Internet Addresses Registry*) – za Južnu i Centralnu Ameriku te Karibe
- APNIC (eng. *Asia Pacific Network Information Centre*) – Za Aziju i otoke na Pacifiku

U Internet registrima nalaze se podaci o rasponima mrežnih adresa organizacija, što je važan podatak za izvršavanje napada.

2.4 Pretraživanje

Nakon što je početna faza pasivnog i aktivnog istraživanja završena, pristupa se fazi pretraživanja. Tijekom pretraživanja napadač nastavlja prikupljati informacije vezane uz ciljnu mrežu i individualna računala koja su dio mreže. Tako se prikupljaju informacije kao što su IP adrese, operacijski sustavi, servisi i instalirane aplikacije, koje mogu pomoći napadaču da utvrdi koju vrstu napada može upotrijebiti za upad u

sustav. Pretraživanje je proces utvrđivanja je li neko računalo spojeno na mrežu i je li dostupno. Postoje tri vrste pretraživanja i to :

- Pretraživanje mrežnih pristupa – utvrđuje otvorene mrežne pristupe i servise,
- Pretraživanje mreže – otkriva aktivne IP adrese,
- Pretraživanje ranjivosti – otkriva prisutnost poznatih ranjivosti.

Pretraživanje mrežnih pristupa je proces otkrivanja otvorenih i dostupnih TCP/IP pristupa na sustavu. Alati za pretraživanje mrežnih pristupa omogućuju napadaču da otkrije servise dostupne na ciljnom sustavu. Svaki servis ili aplikacija povezana je s dobro poznatim brojem pristupa. Na primjer, ako alat za pretraživanje otkrije da je otvoren mrežni pristup 80, to indicira da je na sustavu pokrenut *web* poslužitelj.

Pretraživanje mreže je procedura identifikacije aktivnih računala na mreži u svrhu napada ili sigurnosne provjere. Računala se identificiraju pomoću njihove jedinstvene IP adrese.

Pretraživanje ranjivosti je proces proaktivnog identificiranja ranjivosti računalnih sustava na mreži. Općenito alati za pretraživanje ranjivosti prvo utvrđuju operacijski sustav koji je pokrenut na računalu, njegovu inačicu te nadogradnje koju su instalirane. Nakon toga pretražuju bazu poznatih ranjivosti za taj operacijski sustav što napadač, u kasnijoj fazi napada, može iskoristiti za dobivanje pristupa računalu.

Sustavi za detekciju upada u mrežu te sigurnosni profesionalci s prikladnim alatima vrlo lako mogu otkriti sve vrste pretraživanja, jer za uspješno pretraživanje je potrebna intenzivna interakcija s računalima u mreži.

2.4.1 Metodologija pretraživanja

Metodologija pretraživanja osigurava da napadač prikupi sve važne informacije potrebne za daljnji napad, a sastoji se od sljedećih koraka:

- Provjera aktivnih sustava
- Provjera otvorenih mrežnih pristupa
- Identifikacija servisa
- Utvrđivanje operacijskog sustava
- Pronalaženje ranjivosti
- Crtanje mrežnog dijagrama ranjivih sustava

2.4.2 Tehnika Ping Sweep

Metodologija pretraživanja započinje provjerom sustava koji su aktivni na mreži, tj. sustava koji odgovaraju na zahtjeve koji su im poslani. Najjednostavniji, iako ne i najpouzdaniji, način utvrđivanja "živih" sustava na mreži jest "pinganje", odnosno slanje ICMP paketa cijelom rasponu IP adresa koji pripada mreži. Sva računala koja odgovore na zahtjev smatraju se aktivnima, tj. "živima".

ICMP pretraživanje jest proces slanja ICMP zahtjeva svim računalima na mreži u svrhu utvrđivanja koja računala su aktivna i odgovaraju na zahtjeve. Prednost ICMP pretraživanja je ta što se ono može izvoditi paralelno, tj. sva računala mogu biti

pretražena istovremeno, a postupak se obavlja relativno brzo na cijeloj mreži. Većina programskih alata namijenjenih pretraživanju podržava ovaj oblik kao opciju. Jedan od problema ove metode su osobni i mrežni vatrozidi koji mogu blokirati odgovore na ovakve zahtjeve.

Gotovo svaki sustav za detekciju i prevenciju mrežnih upada detektirati će ovu metodu pretraživanja i o njoj obavijestiti administratora. Većina vatrozidova i *proxy* poslužitelja blokirati će ICMP odgovore, pa napadač ne može precizno utvrditi je li neko računalo aktivno koristeći samo ovu metodu. Tada je potrebno pristupiti nekim drugim metodama pretraživanja.

2.4.3 Pretraživanje mrežnih pristupa i identifikacija servisa

Provjera otvorenih mrežnih pristupa druga je faza u metodologiji pretraživanja. To je proces ispitivanja svakog pojedinog mrežnog pristupa na sustavu kako bi se utvrdilo koji su otvoreni i prihvaćaju zahtjeve za povezivanje. Pretraživanje mrežnih pristupa obično otkriva više vrijednih informacija o računalima i njihovim ranjivostima nego ICMP pretraživanje.

Identifikacija servisa treći je korak u metodologiji pretraživanja i obično se izvodi istim programskim alatima kao i pretraživanje mrežnih pristupa. Identificiranjem otvorenih mrežnih pristupa (eng. *port*), napadač obično može identificirati i servis pokrenut na tom pristupu.

Mjere za detekciju i sprječavanje pretraživanja mrežnih pristupa mogu biti poduzete od strane administratora mreže. Sljedeće mjere bi trebale biti implementirane na svakoj sigurnoj mreži :

- implementiranje sustava za detekciju i prevenciju upada u mrežu,
- vatrozidovi postavljeni u mreži moraju biti ispitani istim metodama pretraživanja koje koriste i napadači kako bi se utvrdila njihova dobra konfiguracija,
- vatrozid bi trebao detektirati pokušaje pretraživanja, i to tzv. ispitivanjem stanja. Treba ispitivati cijele pakete koji mu stižu, a ne samo TCP zaglavlja pri odlučivanju da li propustiti paket,
- mrežni sustavi za detekciju upada trebali bi prepoznavati metode utvrđivanja operacijskih sustava koje koriste alati za pretraživanje,
- samo potrebni mrežni pristupi trebaju biti otvoreni i odgovarati na zahtjeve, ostali trebaju biti filtrirani ili blokirani.

2.4.4 Vrste TCP pretraživanja

SYN ili *Stealth* pretraživanje još se naziva i polu-otvorenim pretraživanjem. Kod njega program počinje otvarati TCP vezu s računalom, ali ne izvršava treći korak trostrukog rukovanja. Napadač šalje SYN paket ciljnom sustavu; ukoliko je SYN/ACK okvir vraćen, tada se pretpostavlja da će ciljni sustav otvoriti vezu i da sluša na mrežnom pristupu kojem je paket poslan. Ukoliko je vraćen okvir s postavljenom RST zastavicom tada se pretpostavlja da je pristup zatvoren i ne prima zahtjeve. Prednost SYN pretraživanja je što ga većina sustava za detekciju upada u mrežu ne interpretira kao napad.

XMAS pretraživanje šalje pakete s postavljenim FIN, URG i PSH zastavicama. Ako nema odgovora tada se mrežni pristup smatra otvorenim. Ako je mrežni pristup zatvoren tada ciljni sustav odgovara s paketom u kojem su postavljene RST i ACK zastavice. XMAS pretraživanje je učinkovito samo protiv sustava koji slijede RFC 793 implementaciju TCP/IP stoga i time nije učinkovito protiv raznih inačica operacijskih sustava Windows.

FIN pretraživanje je slično kao i XMAS pretraživanje, osim što se kod njega šalju paketi s postavljenom FIN zastavicom. Odgovori na zahtjeve i ograničenja su isti kao i kod XMAS pretraživanja.

NULL pretraživanje je analogno FIN i XMAS pretraživanjima, osim što šalje pakete bez postavljenih zastavica.

IDLE pretraživanje koristi lažne IP adrese za slanje SYN paketa ciljnom sustavu. Ovisno o odgovoru, mrežni pristup može biti okarakteriziran kao otvoren ili zatvoren. IDLE pretraživanje utvrđuje odgovore nadgledanjem broja sekvence u TCP zaglavljima.

2.4.5 Utvrđivanje operacijskih sustava

Utvrđivanje operacijskih sustava četvrti je korak u metodologiji pretraživanja i obično se definira kao uzimanje "otiska" TCP/IP stoga operacijskog sustava. Proces utvrđivanja operacijskog sustava omogućuje napadaču da lakše utvrdi ranjivosti sustava prisutnih na mreži. Taj proces najčešće se svodi na otvaranje TCP veze prema ciljnom sustavu i promatranje dobivenih odgovora. Mnogi *web* i FTP (eng. *File Transfer Protocol*) poslužitelji, kao i poslužitelji elektroničke pošte, na otvaranje *telnet* sjednice odgovaraju imenom i inačicom pokrenute aplikacije. To napadačima može pomoći kod otkrivanja tipa operacijskog sustava koji je pokrenut, jer primjerice programski paket Microsoft Exchange Server može biti pokrenut samo na operacijskim sustavima Windows.

Aktivno utvrđivanje operacijskih sustava najčešće je korištena metoda. Ona uključuje slanje podataka sustavu koji se ispituje i analiziranje dobivenog odgovora, a zasniva se na činjenici da razni operacijski sustavi različito implementiraju TCP/IP stog, te se time odgovori na poslane podatke razlikuju. Odgovori se uspoređuju s podacima u već poznatoj bazi i time se pouzdano utvrđuje tip operacijskog sustava. Problem kod aktivnog utvrđivanja je što se ono lako detektira, jer se u malom vremenskom razmaku otvara veliki broj veza prema ciljnom sustavu.

Pasivno utvrđivanje operacijskih sustava teže se detektira, a uključuje promatranje prometa koji putuje mrežom. Tu se umjesto tehnika pretraživanja koriste tehnike prisluškivanja koje se teže otkrivaju od strane sustava za detekciju upada u mrežu. Iako je u tom smislu sigurnije, pasivno utvrđivanje obično je manje pouzdano od aktivnog i zato se rjeđe koristi.

2.5 Enumeracija

Cilj enumeracije je identifikacija korisničkih ili sistemskih računa, kao i drugih postavki na sustavu u vidu potencijalnog korištenja za dobivanje pristupa sustavu. Nije nužno

pronaći administratorski korisnički račun, jer većina privilegija nekog korisnika mogu biti podignute na višu razinu.

2.5.1 Enumeracija *NetBIOS* protokolom

Mnogi programski alati, dizajnirani prvenstveno za pretraživanje mreža i računala mogu locirati i *NetBios* imena računala. Na Windows 2000 platformi ugrađeni programski alat *net view* može biti korišten za *NetBIOS* enumeraciju i to na način prikazan na slici 2.2.

```
C:\> net view / domain
```

Slika 2.2. Primjer korištenja alata net view

Null sjednica se uspostavlja kada se na sustav prijavljujemo s praznim korisničkim imenom i lozinkom. *Null* sjednice poznata su ranjivost pronađena kod CIFS/SMB (*Common Internet File System/ Service Message Block*) sustava.

Jednom kada napadač uspostavi *null* sjednicu s ciljnim sustavom, lako može izlistati korisnička imena, korisničke grupe, dijeljene resurse, dozvole nad datotekama, servise i druge podatke. SMB i *NetBIOS* standardi na Windows platformi koriste programsko sučelje koje vraća podatke o sustavu na TCP pristupu 139.

Jedna od metoda otvaranja *null* sjednice sa sustavom je korištenje sakrivenog *Inter Process Communication* (IPC$) dijeljenog resursa u Windowsima. IPC je dostupan preko ugrađene naredbe komandne linije *net use* koja se koristi za spajanje na dijeljene resurse na drugim računalima u lokalnoj mreži. Prazni navodnici označavaju da se spajamo s praznim korisničkim imenom i lozinkom. Slika 2.3 prikazuje naredbu za spajanje na računalo s IP adresom 192.21.7.1.

```
C:\> net use \\192.21.7.1 \IPC$ "" /u: ""
```

Slika 2.3. Primjer spajanja na računalo null sjednicom

Jednom kada se spoji, napadač ima kanal preko kojega može koristiti druge programske alate i tehnike.

NetBIOS null sjednica koristi specifične mrežne pristupe na računalu za komunikaciju. To su TCP pristupi 135, 137, 139 i/ili 445. Jedna od protumjera za sprječavanje *null* sjednice bila bi zatvaranje ovih pristupa na računalu koje se želi zaštititi.

2.5.2 Enumeracija SNMP protokolom

SNMP enumeracija je proces korištenja SNMP (*Simple Network Management Protocol*) protokola za enumeraciju korisničkih računa na ciljnom sustavu. SNMP implementacija se sastoji od dvije programske komponente za komunikaciju. To su agent, koji je lociran na uređaju na mreži, i SNMP upravljačka stanica, koja komunicira s agentom.

Gotovo svi mrežni uređaji, poput usmjerivača, preklopnika, pa i računala s Windows operacijskim sustavima, posjeduju SNMP agenta koji se koristi za upravljanje određenim varijablama sustava. Princip djelovanja ovog protokola je vrlo jednostavan: s upravljačke stanice šalju se zahtjevi na koje agenti odgovaraju. Zahtjevi se obično odnose na promjenu ili prikaz odgovarajućih konfiguracijskih postavki sustava na mreži. Time upravljačka stanica može detektirati neke značajne promjene na promatranom sustavu, poput neočekivanog prestanka rada neke usluge.

SNMP protokol koristi dvije lozinke za dohvaćanje i promjenu konfiguracijskih postavki na sustavu s agentom. Prva lozinka je javna i naziva se *read community string*, a služi za prikaz trenutne konfiguracije sustava. Druga lozinka naziva se *read/write community string* i koristi se za mijenjanje postavki sustava. Ova lozinka obično je tajna i koriste ju samo sistemski administratori. Čest sigurnosni propust administratora je ostavljanje podrazumijevanih postavki SNMP lozinki, što napadač može iskoristiti za dobivanje dodatnih informacija o postavkama sustava, kao i za njihovu izmjenu.

Najjednostavnija protumjera za zaštitu od SNMP enumeracije je onemogućavanje korištenja SNMP protokola. Ukoliko to nije opcija, preporuča se promjena inicijalnih lozinki korištenih kod ovog protokola.

3. Sigurnosne prijetnje i napadačke tehnike

3.1 Metode upada u sustav

3.1.1 Tehnike otkrivanja lozinki

Mnogi napadi počinju upravo otkrivanjem lozinke koja je ključna pri upadu u neki sustav. Korisnici, pri kreiranju lozinke, često biraju one koje su lako pamtljive i podložne otkrivanju. To je jedan od razloga zašto većina pokušaja otkrivanja lozinki završava uspješno. One mogu biti otkrivene ručno ili nekim automatiziranim alatom koji koristi napad rječnikom ili grubom silom.

Ručno pogađanje obično je vremenski zahtjevno i neefikasno, pa napadači često pribjegavaju automatiziranju tog procesa. Napadač lako može automatizirati pogađanje lozinke pisanjem jednostavne skripte za ljusku Windows NT/2000 sustava. Skripta je prikazana na slici 3.1 i prikazuje pokušaj uspostavljanja *NetBIOS* sjednice s udaljenim računalom koristeći korisnička imena i lozinke iz datoteke.

```
C:\> FOR /F "token=1, 2*" %i in (credentials.txt)
     do net use \\target\IPC$ %i /u: %j
```

Slika 3.1. Automatizirano pogađanje lozinki

Protokol koji se koristi za pokušaj spajanja na udaljeno računalo je *NetBIOS* (*Network Basic Input/Output System*). On služi za komunikaciju među aplikacijama na različitim sustavima unutar iste lokalne mreže.

Također postoje i mnogi programski alati koji automatiziraju proces otkrivanja lozinke, kao što su L0phtCrack, John The Ripper ili KerbCrack. Napadi koje ovi programi izvode mogu se podijeliti u tri kategorije: napadi rječnikom, napadi grubom silom te hibridni napadi.

Napad rječnikom izvršava se učitavanjem datoteke koja sadrži popis mogućih lozinki, u neki od programskih alata koji tada isprobava lozinke na zadanim korisničkim računima.

Napad grubom silom je najzahtjevniji i obično najduže traje. Izvodi se isprobavanjem svim mogućih kombinacija lozinki koristeći slova, brojke te posebne znakove.

Hibridni napad koristi obje od gore navedenih metoda. Obično se počinje s napadom rječnikom, te se onda pokušava kombinirati riječi iz rječnika, s drugim riječima i brojevima.

Još efikasniji način otkrivanja lozinki je pristup datoteci u koju operacijski sustav zapisuje lozinke. Većina sustava računa sažetak lozinki za pohranu u datoteku. Tijekom procesa prijave na sustav računa se sažetak lozinke koju je unio korisnik, koji se tada uspoređuje sa sažetkom zapisanim u datoteci s lozinkama. Ukoliko napadač može otkriti algoritam računanja sažetka koji je korišten, tada na taj način može pokušati otkriti lozinku.

Operacijski sustav Windows 2000 koristi NTLM (*NT Lan Manager*) algoritam za sigurno prenošenje lozinki preko mreže. U ovisnosti o lozinki sažetak izračunat NTLM algoritmom može lako biti razbijen. Na primjer, ukoliko lozinka glasi 123456abcdef, prvo se sva mala slova u lozinki pretvaraju u velika: 123456ABCDEF. Lozinka se nadopunjava praznim znakovima do duljine od 14 znakova: 123456ABCDEF__. Prije kriptiranja razbija se na dva dijela od po 7 znakova te se svaki dio kriptira zasebno kako je prikazano na slici 3.2.

```
123456A = 6BF11E04AFAB197F
BCDEF__ = F1E9FFDCC75575B15
```

Slika 3.2. Razdvajanje lozinke na dva dijela

Konačni sažetak dobiva se spajanjem rezultata. Rezultat je prikazan na slici 3.3.

```
6BF11E04AFAB197FF1E9FFDCC75575B15
```

Slika 3.3. Konačni sažetak

Za razbijanje prve polovice lozinke programskom alatu L0phtCrack treba 24 sata, dok mu za drugi dio treba tek 60 sekundi i to zbog manjeg broja kombinacija u drugoj polovici lozinke (koriste se samo slova i posebni znakovi). Ukoliko bi lozinka, primjerice, bila kraća od 7 znakova drugi dio sažetka uvijek bi bio isti.

Kod svih operacijskih sustava Windows korisnička imena i sažeci lozinki čuvaju se SAM (*Security Accounts Manager*) datoteci, dok se kod Linux sustava oni čuvaju u tzv. *password shadow* datoteci. Tijekom rada operacijskog sustava SAM datoteka je zaključana pa ju napadač ne može dohvatiti. Jedna od opcija za dohvaćanje je podizanje alternativnog operacijskog sustava, kao što su DOS ili neka od distribucija Linuxa, s optičkog medija. Ukoliko napadač uspije dohvatiti SAM datoteku, tada može pokrenuti neki od prije navedenih napada koristeći već spomenute programske alate.

Prisluškivanje lozinki na mreži još je jedan od čestih načina koje napadači koriste za otkrivanje lozinki. Jedan od primjera je prisluškivanje autentikacijske sjednice SMB (*Server Message Block*) protokola. SMB je protokol aplikacijske razine koji se koristi prvenstveno za korištenje dijeljenih resursa (datoteka, pisača, serijskih pristupa) među računalima u mreži, a za autentikaciju koristi NTLM mehanizam. Da bi dohvatio podatke napadač mora nekako prevariti žrtvu kako bi se spojila na njegovo računalo. Obično se to radi porukom elektroničke pošte koja sadrži poveznicu na lažno, napadačevo, računalo. Poruka može sadržavati običnu HTML oznaku za sliku koja sadrži poveznicu na lažni poslužitelj, kao što se vidi na slici 3.4.

```
<img src=file://lazni_server/null.gif height=1 width=1>
```

Slika 3.4. HTML oznaka s poveznicom na lažni poslužitelj

Računalo žrtve tada se automatski pokušava spojiti na napadačevo računalo s podacima korisničkog računa trenutnog korisnika. Kada napadač dohvati sažetak lozinke, dekripciju je lako izvršiti.

Neke od protumjera za zaštitu lozinki su korištenje snažnih lozinki, vremenski ograničeno trajanje lozinki, te praćenje dnevničkih zapisa na sustavima. Snažne lozinke moraju biti duge barem od 8 do 12 znakova, trebaju koristiti brojeve, slova i posebne znakove. Vremensko trajanje lozinki trebalo bi biti ograničeno na maksimalno 30 dana te bi korisnike nakon tog perioda trebalo prisiliti da ju promijene. Praćenje dnevničkih zapisa jedan je od načina detekcije napada grubom silom te se takva praksa preporuča svakom sistemskom administratoru.

3.1.2 Podizanje razine prava i izvršavanje aplikacija

Podizanje razine prava obično znači dodavanje dozvola i prava trenutno korištenom korisničkom računu kako bi on bio izjednačen s administratorskim računom. Administratorski korisnički računi obično imaju strože sigurnosne politike kod biranja lozinki, pa je administratorske lozinke teže otkriti. Napadači tada obično pribjegavaju drugim metodama kako bi koristeći običan korisnički račun pridobili administratorska prava i ovlasti. Te metode uključuju iskorištavanje neke sigurnosne rupe u jezgri operacijskog sustava, kako bi napadač dodao novi korisnički račun s administratorskim ovlastima ili pak poništio administratorsku lozinku. Mnogi već postojeći programi koji iskorištavaju sigurnosne propuste i podižu razinu prava korisniku lako su dostupni na Internetu.

Jednom kada je napadač spojen na sustav i ima administratorske ovlasti sljedeća stvar koju želi jest izvođenje aplikacije na ciljnom sustavu. Cilj izvođenja aplikacija može biti instalacija programa za praćenje ili drugih zlćudnih programa, jednostavno prikupljanje povjerljivih informacija ili oštećivanje sustava. Kada je napadač u mogućnosti izvoditi naredbe, sustav se smatra kompromitiranim.

3.1.3 Brisanje tragova i sakrivanje podataka

Nakon izvršenog napada napadači obično pokušavaju prikriti tragove svoje prisutnosti na sustavu. Za sakrivanje aktivnosti često se koriste posebnim zlćudnim programima koji se nazivaju *rootkit* programi. Oni zahtijevaju administratorske ovlasti za izvršavanje i sakrivaju procese koje je napadač pokrenuo od drugih korisnika sustava. U sebi najčešće imaju ugrađen i *backdoor* program za lakši pristup sustavu te ih je vrlo teško detektirati.

Još neke od metoda sakrivanja tragova su onemogućivanje praćenja, odnosno zapisivanja u dnevnike tijekom aktivnosti napadača na sustavu te brisanje dnevničkih zapisa.

Napadači često imaju potrebu sakriti neku datoteku na kompromitiranom sustavu. Na NTFS (*NT File System*) sustavima to im omogućuje postojanje alternativnih podatkovnih tokova. Oni omogućuju stvaranje skrivenih datoteka povezanih s normalnim, vidljivim datotekama. Za sakrivanje podataka koristi se i steganografija, tj. sakrivanje podataka u drugim tipovima podataka kao što su slike, tekstualne datoteke i slično.

3.2 Zloćudni programi

Trojanski konji i *backdoor* programi su zloćudni programi pomoću kojih napadači mogu dobiti pristup željenom sustavu. Iako dolaze u puno različitih varijanti svima je zajednička jedna stvar: moraju biti instalirani od strane drugog programa ili korisnik mora biti prevaren da bi ih sam instalirao na svoj sustav.

Virusi i crvi mogu biti jednako destruktivni kao i trojanski konji i *backdoor* programi. Mnogi virusi, u biti, i sadrže trojanske konje i njima mogu zaraziti sustave koje napadaju. Napadači ove programe često koriste kao sredstvo upada u sustav.

3.2.1 Trojanski konji i backdoor programi

Backdoor program instaliran na nekom sustavu omogućuje napadaču kasniji pristup sustavu kroz tzv. stražnja vrata. Tu se obično misli na slušanje programa na nekom mrežnom pristupu računala i čekanje na naredbe napadača koje se onda izvršavaju. Cilj korištenja *backdoor* programa najčešće je brisanje tragova prethodnog napada, ali on se često koristi i za zadržavanje pristupa kompromitiranom računalu čak i nakon što je napad otkriven od strane administratora.

Dodavanje novog servisa uobičajena je tehnika prikrivanja tih zloćudnih programa na računalu. Prije instalacije programa napadač mora istražiti servise koji su na sustavu već pokrenuti i izabrati ime novog servisa tako da ne bude lako uočljivo drugim korisnicima i administratoru ili čak izabrati ime nekog postojećeg servisa koji nije pokrenut. Ova tehnika je učinkovita jer administratori kada otkriju napad na sustavu prvo traže nešto čudno i neuobičajeno bez provjeravanja već poznatih servisa. Tako napadač može ponovno pristupiti sustavu s minimalno uočljivim tragom u dnevnicima. *Backdoor* programi u većini slučajeva napadaču omogućuju pristup sustavu s administratorskim pravima.

RAT (*Remote Administration Trojan*) trojanski konji primjer su *backdoor* programa koji omogućuju daljinsko upravljanje sustavom. Oni korisnicima sustava pružaju korisne funkcije, dok istovremeno preko mrežnog pristupa napadaču omogućuju upravljanje računalom. Jednom kad je takav program pokrenut on se ponaša kao svaki drugi program na sustavu, pokreće druge procese te koristi sistemske servise. Za razliku od tipičnih *backdoor* programa RAT trojanski konji se integriraju duboko u operacijski sustav te se uvijek sastoje od dva modula : klijenta i poslužitelja. Poslužitelj je instaliran na inficiranom sustavu, dok je klijent program na napadačevom računalu pomoću kojeg se kontrolira kompromitirani sustav.

Trojanski konji su programi koji su naizgled bezopasni, čak i korisni. Najčešći način rasprostranjivanja trojanskih konja je s nekim drugim softverskim paketom. Jednom kada su instalirani mogu uzrokovati krađu ili uništavanje podataka, pad ili usporavanje sustava ili biti iskorišteni kao točka za pokretanje DDoS (*Distributed Denial of Service*) napada, odnosno distribuiranog napada s uskraćivanjem usluga. Mnogi trojanski konji koriste se za manipulaciju datotekama i procesima napadnutog računala, udaljeno izvršavanje naredbi, snimanje rada na sustavu itd. Sofisticiraniji trojanski konji podržavaju čak i mogućnost povratnog spajanja na računalo napadača i obavještavanja o uspjehu napada kroz primjerice IRC (*Internet Relay Chat*) kanal. Kako je trojanskim konjima za rasprostranjivanje potreban program domaćin, oni se

često distribuiraju u programima koji se reklamiraju kao besplatni te imaju korisne funkcije poput optimiziranja sustava, uklanjanja zloćudnih programa, itd. Tablica 3.1 prikazuje neke od poznatijih trojanskih konja i mrežne pristupe koje koriste.

Tablica 3.1. Poznati trojanski konji

Trojanski konj	Protokol	Mrežni pristup
BackOrifice	UDP	31337, 31338
Deep Throat	UDP	2140, 3150
NetBus	TCP	12345, 12346
Whack-a-mole	TCP	12361, 12362
NetBus 2	TCP	20034
GirlFriend	TCP	21544
Masters Paradise	TCP	3129, 40421, 40422, 40423, 40426

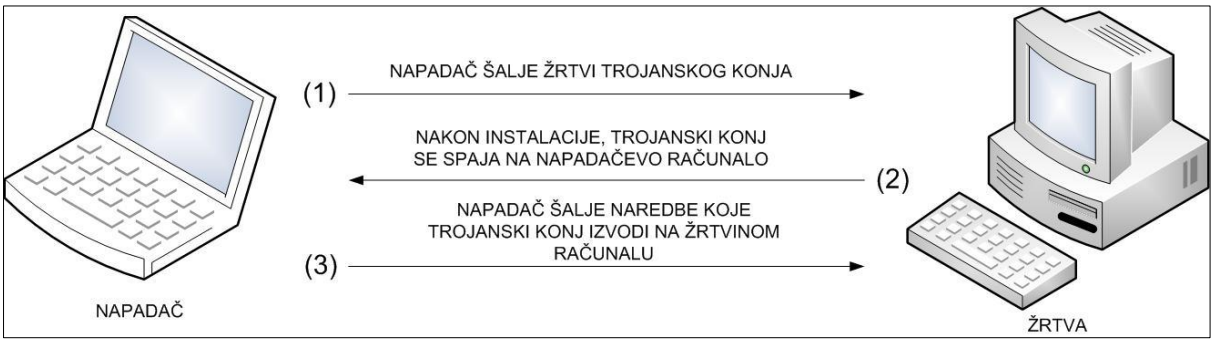
Za primanje naredbi i komunikaciju trojanski konji koriste tzv. sakrivene kanale. Tehnika na koju se skriveni kanali oslanjaju jest tuneliranje, koje dozvoljava prenošenje poruka jednog protokola preko nekog drugog. ICMP (*Internet Control Message Protocol*) tuneliranje jedan je od primjera kod kojeg se ICMP zahtjevi i odgovori koriste za razmjenu podataka između napadača i trojanskog konja na inficiranom sustavu.

S obzirom na vrstu napada koju obavljaju, trojanske konje dijelimo na sljedeće vrste:

- Trojanski konji s udaljenim upravljanjem (RAT – *Remote Administration Trojan*)
- Trojanski konji koji napadaču šalju podatke s inficiranog sustava
- Destruktivni trojanski konji koji brišu i kompromitiraju podatke
- DoS (*Denial of Service*) trojanski konji koji se koriste za izvršavanje napada s uskraćivanjem usluge
- *Proxy* trojanski konji koji tuneliraju mrežni promet ili se koriste za izvršavanje napada s drugog sustava
- FTP trojanski konji koji služe kao FTP poslužitelj na računalu koje su inficirali
- Trojanski konji koji onemogućuju sigurnosne programe (antiviruse, vatrozidove i sl.)

Pomoću trojanskih konja s povratnim spajanjem napadači lako mogu dobiti pristup internoj mreži iz vanjske (npr. s Interneta). Oni djeluju kao obrnuti poslužitelj i to tako

da u određenim vremenskim razmacima pokušavaju pristupiti vanjskom poslužitelju kako bi provjerili je li napadač izdao nove naredbe. Ukoliko je, naredbe se izvršavaju. Za spajanje na vanjski poslužitelj koristi se obični HTTP protokol. To ovu vrstu napada čini jako opasnim, s obzirom da se čini kao da korisnik na inficiranom računalu generira obični HTTP promet.



Slika 3.5. Napad trojanskim konjem s povratnim spajanjem

Wrapperi su softverski paketi koji se koriste za spajanje trojanskih konja s legitimnim datotekama, odnosno programima, kako bi se oni lakše distribuirali. Trojanski konji i legitimni programi spajaju se u jednu izvršnu datoteku i instaliraju pri pokretanju iste.

Računalne igre i drugi softverski paketi s animiranim instalacijama najčešće su korišteni za distribuciju trojanskih konja, upravo zato što kod instalacije korisniku odvraćaju pažnju te tako korisnik ne primjećuje sporiju instalaciju trojanskog konja već samo instalaciju legitimnog programa kojemu je trojanski konj dodan.

Neuobičajeno ponašanje, kao i značajno usporavanje sustava, obično je znak prisutnosti trojanskog konja. Događaji poput pokretanja i izvršavanja aplikacija bez korisnikove interakcije, otvaranja neočekivanih *web* stranica prilikom pregledavanja Interneta, promjene pozadine ili *screensaver*-a te drugih neobičnih akcija najčešći su indikatori prisutnosti ovih zloćudnih programa.

Većina komercijalnih antivirusnih programa imaju mogućnost detekcije i onemogućavanja trojanskih konja. Ovi alati mogu automatski pretražiti tvrdi disk kod pokretanja operacijskog sustava, te time detektirati i onemogućiti rad trojanskih konja prije nego su oni učinili štetu na sustavu. Jednom kada je sustav zaražen puno ga je teže očistiti i vratiti u originalno stanje, iako ovi alati podržavaju i takvu mogućnost, zbog čega ih je važno koristiti.

Ključna aktivnost za sprječavanje trojanskih konja je edukacija korisnika sustava da ne instaliraju neprovjerene programe dohvaćene s Interneta i da ne otvaraju privitke elektroničke pošte poslane od nepoznatih osoba. Mnogi administratori ne dopuštaju instalaciju programa običnim korisnicima upravo iz ovih razloga.

Jedna od dobrih praksi zaštite sustava od napada zloćudnim softverom je provjera integriteta sistemskih datoteka na tvrdom disku. Kod Windows operacijskih sustava to se može učiniti programskim alatom *sigverif*.

3.2.2 Virusi i crvi

Virusima i crvima zajednička osobina je da oboje predstavljaju formu zloćudnog softvera. Virus djeluje tako da inficira drugi izvršni program i koristi ga kao nosioca za širenje. Virusni kod se, dakle, ubacuje u prethodno dobroćudan program i širi se njegovim izvršavanjem. Primjeri programa prenosioca virusa najčešće su makro programi, računalne igre, privitci elektroničke pošte, Visual Basic skripte te animacije.

Crvi su podvrsta virusa, ali ih razlikuje svojstvo da su samoreplicirajući. Crv se automatski širi s jednog sustava na drugi, dok virus treba program nosioc da bi se širio. I virusi i crvi izvršavaju se bez znanja i namjere korisnika.

Virusi se klasificiraju prema tome što inficiraju i kako inficiraju. Virus može inficirati sljedeće dijelove sustava:

- Sistemski dio
- Datoteke
- Makro programe (na primjer makro programe programskog paketa Microsoft Word)
- *Companion* datoteke
- klastere tvrdih diskova
- *Batch* datoteke
- Izvorni kod

Virusi inficiraju sustav kroz interakciju s vanjskim sustavima, a prema tehnici inficiranja razlikujemo sljedeće vrste virusa:

- Polimorfni virusi kriptiraju kod na drugačiji način kod svake infekcije i sposobni su mijenjati formu, zbog čega ih je teško detektirati.
- *Stealth* virusi pokušavaju sakriti normalne karakteristike koje virusi imaju. Na primjer, mogu promijeniti vremensku oznaku nastanka datoteke koju su zarazili kako bi spriječili sustav ili antivirusni program da detektira novu datoteku na računalu.
- Brzoinficirajući i sporoinficirajući virusi izbjegavaju detekciju inficirajući sustav jako sporo ili jako brzo.
- Slabo raspršeni virusi inficiraju mali broj programa i datoteka.
- Kriptirani virusi koriste enkripciju za zaštitu od detekcije.
- Virusi koji inficiraju na više različitih načina.
- Virusi koji se pridodaju praznim dijelovima datoteka.
- Virusi koji su poslani tuneliranjem preko drugog protokola.
- Zamaskirani virusi predstavljaju se kao neki drugi program.
- NTFS i Active Directory virusi koji su specifični za NT podatkovne sustave i Active Directory sustave na Windows platformama.

Postoji puno komercijalnih, kao i slobodno raspoloživih programskih alata za detekciju i uklanjanje virusa, tzv. antivirusni alati. Detekcija virusa najviše se oslanja na tzv. "potpis" virusa. To znači da antivirusne kompanije za svaki novootkriveni virus kreiraju jedinstveni "potpis" kojim se program koji sadrži virus može jednoznačno prepoznati. Međutim, dok virus još nije otkriven i baze antivirusnih programa nisu

osvježene, on lako može proći neopažen jer ga je tada puno teže detektirati. Takvi, još nepoznati virusi, često se nazivaju terminom "*in the wild*" (hrv. u divljini).

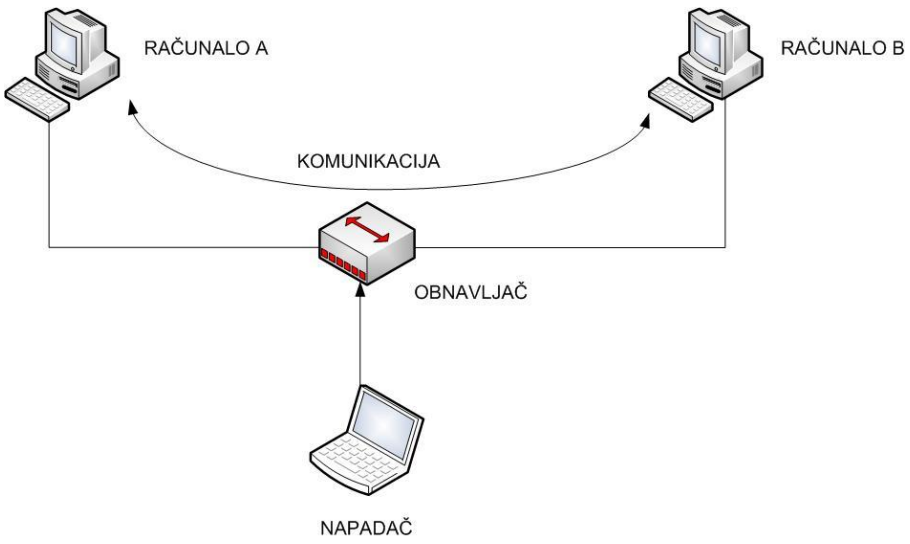
3.3 Prisluškivanje

Sniffer je programski alat za prisluškivanje, odnosno hvatanje mrežnog prometa koji rezultate može prikazivati u konzolnom ili naprednijem grafičkom sučelju. Neki sofisticiraniji *snifferi* imaju mogućnost interpretacije toka paketa kao originalnog podatka koji se prenosi, npr. poruke elektroničke pošte ili običnog dokumenta. Obično se koriste za prisluškivanje prometa između dva sustava. U ovisnosti o načinu upotrebe *sniffera* i sigurnosnih mjera u okolini u kojoj ih koriste, napadači mogu otkriti povjerljive podatke poput korisničkih imena i lozinki. Većina napadačkih alata koriste *sniffere* za dohvaćanje važnih podataka poslanih od strane ciljnog sustava.

Snifferi rade tako da ne hvataju samo promet namijenjen sustavu na kojem su pokrenuti, već sav promet koji do njih dolazi. Obično je posebno važan promet namijenjen ciljnom sustavu. Ovaj način rada poznat je kao promiskuitetni mod te se omogućuje posebnim pogonskim programima za mrežnu karticu. Napadački alati koji koriste prisluškivanje najčešće dolaze s već ugrađenim pogonskim programima potrebnim za rad.

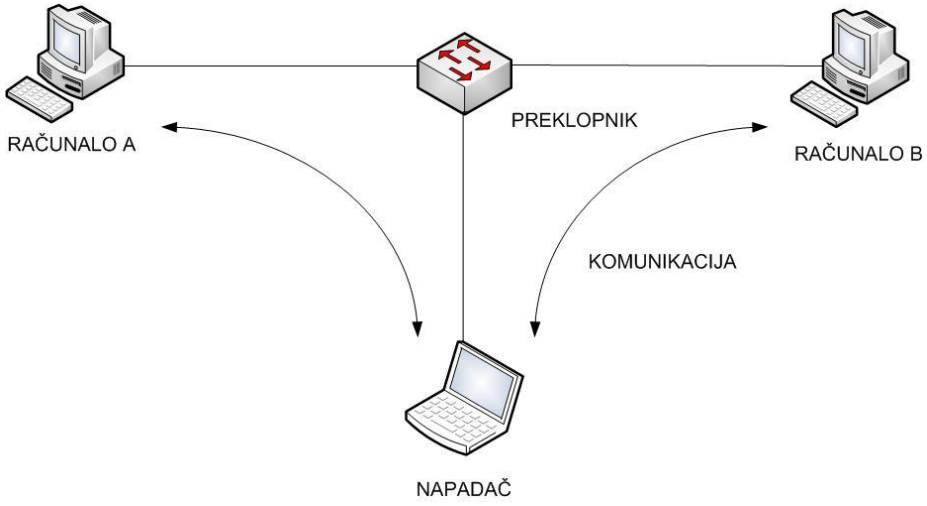
Svi protokoli koji ne koriste enkripciju podataka koje prenose ranjivi su na prisluškivanje. Tu se ubrajaju protokoli kao što su HTTP, POP3, SNMP, te FTP koji se najčešće hvataju *snifferima* i koriste za prikupljanje važnih informacija poput lozinki.

S obzirom na aktivnosti *sniffera* tijekom prisluškivanja razlikujemo dvije osnovne vrste: aktivno i pasivno prisluškivanje. Kod pasivnog prisluškivanja *sniffer* samo sluša i hvata promet na mrežnom priključku koji mu je dodijeljen. Ova vrsta prisluškivanja korisna je kod mreža spojenih preko obnavljača (eng. *hub*) jer su paketi poslani preko obnavljača vidljivi na svim računalima spojenima na mrežu.



Slika 3.6. Pasivno prisluškivanje

Aktivno prisluškivanje koristi se na mrežama koje koriste preklopnike (eng. *switch*) za spajanje računala. Karakteristika tih mreža je da se paketi poslani preklopniku prosljeđuju samo odredišnom računalu čime je sigurnost mreže povećana u odnosu na mreže spojene obnavljačima. Kako bi *sniffer* ipak vidio promet poslan drugim računalima koriste se tehnike kao što je ARP trovanje koja je detaljnije opisana u ovom poglavlju. Kod ovih tehnika promet namijenjen računalima koja se prisluškuju preusmjeruje se na računalo na kojem je pokrenut *sniffer*.



Slika 3.7. Aktivno prisluškivanje

Najučinkovitija zaštita od prisluškivanja je kriptiranje podataka koji putuju mrežom. Iako samo kriptiranje ne sprječava napadače da prisluškuju promet, podaci koji su dohvaćeni nisu im od nikakve koristi. Korištenje VPN (eng. *Virtual Private Network*) tehnologije s enkripcijom najčešća je praksa u korporativnim okruženjima.

3.3.1 ARP trovanje

ARP protokol koristi se za traženje nepoznatih MAC adresa (hardverskih adresa mrežnih kartica) uz pomoć poznatih IP adresa na Ethernet mreži. Kada računalo na Ethernet mreži pomoću TCP/IP protokola želi komunicirati s drugim računalom prvo treba dohvatiti njegovu MAC adresu. Prvo se provjerava priručna ARP memorija sustava koji otvara komunikaciju te se, ukoliko MAC adresa tamo nije pronađena, šalje ARP upit s traženom IP adresom svim računalima u mreži. Računalo kojem je dodijeljena tražena IP adresa odgovara sa svojom MAC adresom čime komunikacija TCP/IP protokolom može započeti.

ARP trovanje je tehnika kojom se nedostaci ARP protokola koriste kako bi se promet prisluškivao i analizirao ili u potpunosti zaustavio. Izvodi se slanjem lažnih ARP poruka računalima i mrežnim uređajima Ethernet mreže. Ove poruke sadrže lažne MAC adrese i time pokušavaju zavarati uređaje na mreži kako bi svoj promet slali krivom, najčešće napadačevom ili nepostojećem računalu (DoS napad). Na ovaj način napadač može, preusmjeravanjem prometa na svoje računalo i prosljeđivanjem računalu kojem je prvotno namijenjen, uspostaviti takozvani MitM (eng. *Man in the Middle*) napad.

Još jedan od načina omogućavanja aktivnog prisluškivanja je tzv. MAC *flooding* napad ili napad preplavljivanjem MAC adresama. Kod ovog napada preklopniku se šalju velike količine lažnog prometa sa slučajno odabranim MAC adresama. Time se, zbog ispunjenosti memorije korištene za čuvanje IP i MAC adresa, neki preklopnici počinju ponašati kao obnavljači, odnosno promet koji primaju prosljeđuju svim spojenim računalima.

Mrežni administratori mogu zaštititi mrežu od napada ARP trovanjem izbjegavanjem korištenja priručne ARP memorije, no to održavanje mreža s velikim brojem računala čini težim. U korporativnim okruženjima sigurnost se može povećati i ograničavanjem samo jedne MAC adrese po mrežnom priključku preklopnika.

3.3.2 DNS lažiranje

DNS lažiranje je tehnika koju napadači koriste kako bi iskoristili nedostatke DNS sustava za prijevaru potencijalnih žrtava. Kod ovog napada cilj napadača je prevariti DNS poslužitelj tako da ga se uvjeri da su informacije koje mu se šalju točne. Jednom kada je primio lažne informacije DNS poslužitelj ih sprema u svoju priručnu memoriju na neko određeno vrijeme. Korisnik koji pošalje zahtjev za razrješenjem domenskog imena od poslužitelja može dobiti krivu informaciju o odgovarajućoj IP adresi i time biti preusmjeren na napadačevo računalo. Za izvođenje ovog napada koristi se propust u DNS sustavu koji dozvoljava da DNS poslužitelj prihvaća informacije bez odgovarajuće provjere izvora čime korisnike sustava izlaže potencijalnom napadu.

Ova vrsta napada najčešće se koristi za preusmjeravanje skupine žrtava, npr. korisnika neke *web* stranice, na lažnu Internet lokaciju u svrhu prikupljanja osjetljivih informacija. Napadač može, primjerice, kompromitirati DNS poslužitelj koji potencijalne žrtve koriste krivim informacijama o IP adresama željene *web* stranice. Zatim na lažni poslužitelj postaviti identične stranice i prikupiti lozinke i korisnička imena kojima su se korisnici pokušali autenticirati. Na lažne stranice napadač može postaviti i zlođudni sadržaj te na taj način iskoristiti povjerenje žrtava i kompromitirati njihova računala.

3.4 DoS napadi

Kod DoS napada (eng. *Denial of Service*) cilj napadača je onesposobljavanje ili značajno usporavanje rada napadnutog sustava opterećivanjem njegovih resursa ili onemogućavanjem pristupa legitimnim korisnicima. Napad se može izvršiti na razini sustava ili na razini cijele mreže.

3.4.1 Vrste DoS napada

Dvije su osnovne kategorije u koje možemo svrstati DoS napade. To su napadi izvršeni s jednog računala te napadi izvršeni s više računala ili tzv. DDoS (eng. *Distributed Denial of Service*) napadi.

Cilj DoS napada nije dobivanje nedozvoljenog pristupa sustavu, već sprječavanje legitimnih korisnika da pristupaju sustavu. Scenariji mogućeg napada su sljedeći:

- Zagušenje mreže prometom i time sprječavanje legitimnog prometa
- Ometanje ili prekid veze između dva računala što rezultira sprječavanjem pristupa računalima
- Sprječavanje pristupa određenom sustavu

Različiti programski alati koji izvršavaju DoS napade koriste razne metode za zagušenje mreže, ali rezultat je uvijek isti: usluga pokrenuta na sustavu ili cijeli sustav je nedostupan korisnicima zbog neuobičajeno velikog broja zahtjeva. DoS napadi smatraju se nesofisticiranim napadima, jer napadač njima ne dobiva pristup sustavu, već upravo sprječava pristup drugim korisnicima i time ometa normalan rad sustava. Zato je ovaj vrsta napada obično zadnja opcija napadačima. Distribuirani DoS napadi (DDoS) mogu imati značajan efekt i biti izuzetno destruktivni s obzirom da se izvršavaju s velikog broja računala istodobno.

3.4.2 DDoS napadi

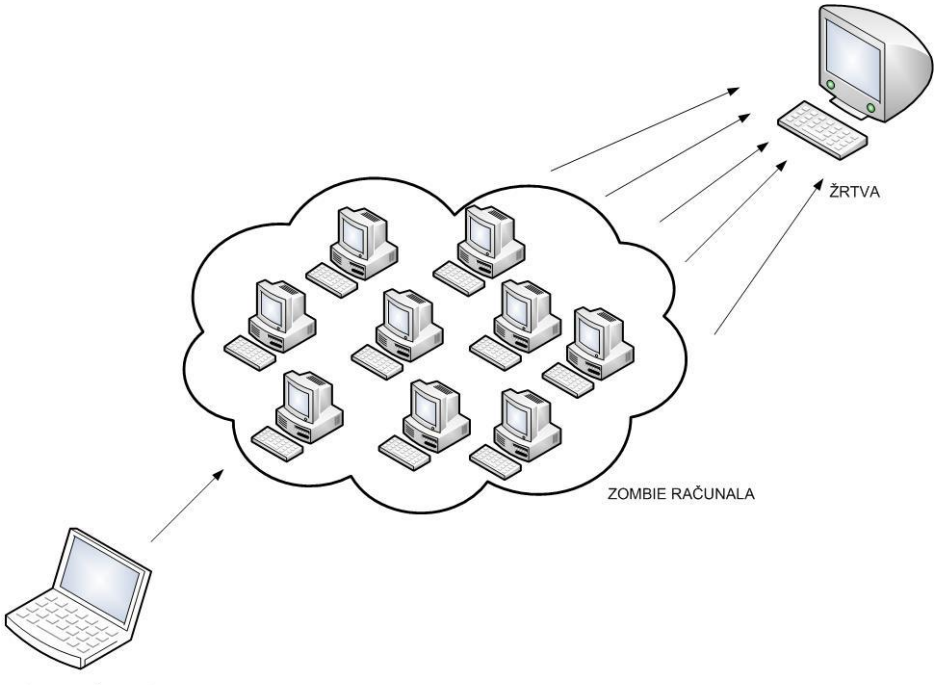
DDoS napadi naprednija su inačica DoS napada kod kojih se napad izvodi s velikog broja najčešće prethodno kompromitiranih računala. Napad se koordinira s jednog glavnog ili tzv. *master* računala. Sudionici ove vrste napada dijele se na:

- *Master* računalo koje koordinira napadom i upravlja svim računalima kojima se napad izvodi.
- *Slave* računala ili zombiji su kompromitirana računala pod upravljanjem *master* računala. Zombiji su najčešće napadnuti nekom drugom vrstom napada koji je prethodio samom DDoS napadu i obično se nazivaju sekundarnim žrtvama.
- Računala žrtve ili primarne žrtve napada predstavljaju računala na koje se izvodi sam DDoS napad.

Sam napad izvodi se u dvije faze:

- U prvoj fazi napadaju se slabo zaštićena računala u raznim mrežama diljem svijeta koja se kasnije koriste za izvođenje DDoS napada.
- U drugoj fazi kompromitirana računala iz prve faze napada koriste se za izvršavanje distribuiranog DoS napada.

Kod DDoS napada teško je detektirati sam izvor jer se napad izvodi s velikog broja različitih IP adresa.



Slika 3.8. DDoS napad

BOT je skraćeni naziv za *web* robota koji predstavlja automatizirani program s inteligentnim ponašanjem. BOT programi najčešće su korišteni od strane *spammera* za automatizirano slanje neželjene pošte (eng. *spam*), ali također mogu biti korišteni i za izvršavanje udaljenih napada.

BOTNET je naziv za skupinu kompromitiranih računala na kojima su pokrenuti BOT programi. Ovakvi sustavi često se koriste za izvršavanje DDoS napada kao i za druge nelegalne radnje kao što su slanje poruka neželjene pošte, internetske prijevare, krađa povjerljivih informacija i sl.

3.4.3 Primjeri DoS napada

- ***Smurf napad***

Kod *smurf* napada šalje se veliki broj ICMP ECHO zahtjeva na *broadcast* adresu mreže s IP adresom žrtve kao adresom izvora. Tada svako računalo koje je primilo paket šalje ICMP ECHO odgovor na adresu žrtve. Time se broj paketa poslanih žrtvi uvećava proporcionalno za broj računala ili sekundarnih žrtava koji su primili zahtjev. Kod višepristupnih mreža koje podržavaju *broadcast* način razašiljanja paketa, stotine računala odgovaraju na svaki poslani ICMP ECHO zahtjev i time se izvršava DoS napad velikih razmjera. IRC (eng. *Internet Relay Chat*) poslužitelji najčešća su meta ove vrste napada.

- ***SYN flooding napad***

SYN *flooding* napad izvršava se slanjem zahtjeva za TCP vezom brže nego što to računalo žrtve može obraditi. Kao izvorna adresa zahtjeva postavlja se neka slučajno izabrana IP adresa. Napadnuto računalo tada odgovara na zahtjev i čeka potvrdu

koja nikad ne stiže. Na taj način tablica mrežnih veza žrtve se brzo popunjava jer čeka na potvrdu svih novootvorenih veza i počinje ignorirati nove zahtjeve. Time napadnuti sustav postaje nedostupan za legitimne korisnike.

3.4.4 Mjere zaštite

Postoji nekoliko mjera za prevenciju DoS napada, a neke od njih su:

- *Network-ingress* filtriranje – većina davatelja Internet usluga danas implementira ovu metodu kontrole mrežnog prometa. Kod ovog filtriranja kontroliraju se IP adrese u zaglavljima okvira i time se sprječava lažiranje. Iako to ne sprječava sam napad, omogućuje se lakše utvrđivanje izvora.
- Ograničavanje prometa – većina današnjih usmjerivača ima mogućnost ograničavanja količine pojedine vrste mrežnog prometa. Ova metoda ponekad se naziva oblikovanje prometa.
- Sustavi za detekciju upada – gotovi svi današnji IDS (eng. *Intrusion Detection System*) sustavi imaju mogućnost detekcije najpopularnijih DoS napada, iako će teško detektirati neke nove varijante napada. Ovi sustavi također su korisni za sprječavanje izvršavanja napada iz mreže koju štite.
- Sustavi za praćenje mrežnog prometa – mogu biti korisni kod detektiranja DoS napada.

3.5 Krađa sjednice

Krađa sjednice događa se kada napadač preuzme kontrolu nad korisničkom sjednicom nakon što se korisnik uspješno autenticirao s poslužiteljem. Napad krađom sjednice obuhvaća identificiranje trenutno aktivne sjednice između korisnika i poslužitelja te njeno preuzimanje. Važan dio ovog napada je predviđanje broja sekvence (SN - eng. *sequence number*) paketa sjednice o čemu će biti riječi u ovom poglavlju.

3.5.1 Koraci napada

U pravilu, tri su osnovna koraka u izvođenju napada krađom sjednice:

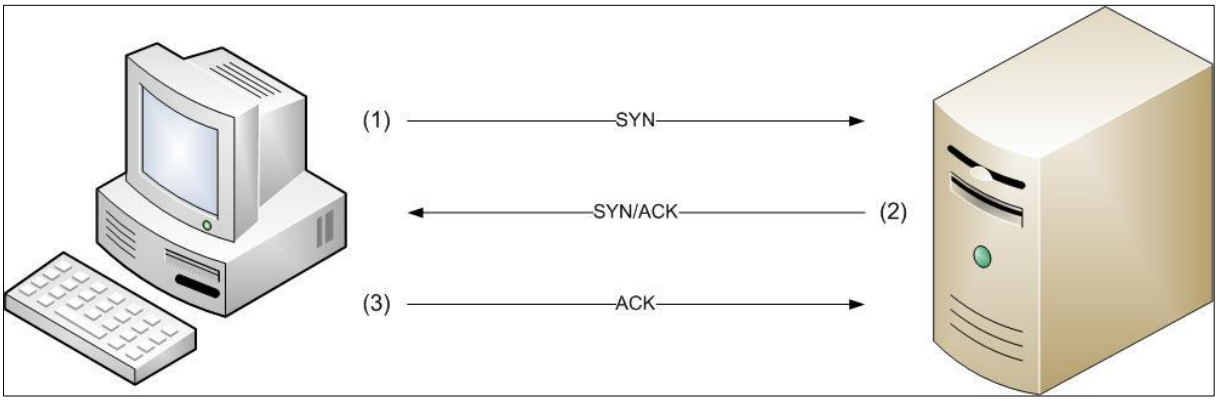
1. Napadač prisluškivanjem identificira otvorenu sjednicu na relaciji korisnik – poslužitelj i predviđa broj sekvence sljedećeg paketa sjednice.
2. Korisniku se šalje TCP paket s postavljenom zastavicom RST ili FIN kako bi se terminirala veza s njegove strane. Alternativno, napadač može izvršiti i DoS napad kako bi onesposobio korisnika za nastavak sjednice.
3. Napadač šalje poslužitelju TCP paket s predviđenim brojem sekvence i poslužitelj prihvaća paket kao ispravan.

Jedna od glavnih odlika TCP protokola je pouzdanost u dostavi paketa. Kako bi to postigao TCP koristi potvrđivanje svakog dostavljenog paketa ACK paketima i brojeve sekvence. Broj sekvence jedinstven je za svaki poslani paket i na njegovo predviđanje se oslanja napadač kod napada krađom sjednice. Kako bi bolje razumjeli krađu sjednice moramo znati kako funkcionira TCP protokol. Otvaranje TCP sjednice

započinje postupkom trostrukog rukovanja (eng *three-way handshake*). On se sastoji od tri sljedeća koraka:

1. Korisnik inicira vezu s poslužiteljem šaljući TCP paket s postavljenom SYN zastavicom i inicijalnim brojem sekvence. Obično je to neki slučajno generirani broj.
2. Poslužitelj prima paket i vraća korisniku TCP paket s postavljenim SYN i ACK zastavicama te inicijalnim brojem sekvence uvećanim za 1.
3. Korisnik potvrđuje primljeni paket TCP paketom s postavljenom zastavicom ACK i uvećanim brojem sekvence.

Ovim koracima TCP veza je otvorena i podaci se počinju razmjenjivati. Ona može biti zatvorena u bilo kojem trenutku od obiju strana slanjem TCP paketa s postavljenom zastavicom RST ili FIN.



Slika 3.9. Trostruko rukovanje

TCP protokol odgovoran je za preslagivanje paketa u originalni redoslijed, ukoliko oni tijekom prijenosa ne stignu redom kojim su poslani što je kod prijenosa Internetom čest slučaj. Svakom paketu tako je dodijeljen jedinstveni broj sekvence koji se koristi kod preslagivanja. Kao što je već objašnjeno prvi paket koji se šalje kod otvaranja TCP veze je tzv. sinkronizacijski paket s postavljenom SYN zastavicom i u njemu se postavlja inicijalni broj sekvence (ISN – eng. *Initial Sequence Number*). To je slučajno generirani broj s više od 4 milijarde kombinacija, ali svejedno ga je statistički moguće predvidjeti. Nakon slanja inicijalnog broja svaki sljedeći broj sekvence samo je uvećan za jedan tijekom otvaranja veze ili za veličinu podataka u paketu tijekom prijenosa.

Kako bi uspješno predvidio sljedeći broj sekvence napadač mora prisluškivati promet između dva sustava koja komuniciraju. Tijekom prisluškivanja mora dohvatiti inicijalni broj sekvence ili neki od sljedećih brojeva i predvidjeti broj koji će sam staviti u sljedeći TCP paket. Ovo u praksi može biti kompliciranije nego što zvuči, jer paketi u mreži putuju velikom brzinom. Ukoliko napadač nije u mogućnosti prisluškivati tijek sjednice pogađanje broja sekvence postaje još težom zadaćom.

Jednom kada je pogodio broj napadač šalje poslužitelju TCP paket s novim brojem sekvence i time preuzima korisničku sjednicu. Naravno, napadačev paket mora stići prije paketa korisnika, što se postiže terminiranjem sjednice s korisnikove strane.

3.5.2 Mjere zaštite

Krađa sjednice jedan je od najopasnijih napada upravo zato što većina sustava koristi TCP/IP protokol za osnovnu komunikaciju i to ih čini ranjivima. Pseudo-slučajno izabiranje inicijalnog broja sekvence uvedeno je kod nekih novijih operacijskih sustava kako bi se otežalo njegovo pogađanje. Međutim ta mjera je uzaludna ukoliko napadač može prisluškivati promet i tako dobiti sve podatke potrebne za napad.

Za sprječavanje krađe sjednica obično se implementira više linija obrane. Najefektivnija mjera zaštite svakako je korištenje enkripcije mrežnog prometa. Najčešće se koristi IPSec protokol, a mogu pomoći i aplikacije poput *Secure Shell* programa te enkripcija SSL protokolom na aplikacijskoj razini. Na ovaj način potencijalni napadači mogu promatrati mrežni promet, ali im to neće pomoći u izvršavanju napada.

Još jedna od potencijalnih mjera zaštite je ograničavanje točaka udaljenog pristupa štićenoj mreži na minimum. Ukoliko mreža mora imati udaljene korisnike, primjerice pristup radnika korporativnoj mreži od kuće ili s terena, preporuča se korištenje VPN tehnologije sa sigurnom enkripcijom.

3.6 Napadi na Internetu

Sustavi poput *web* poslužitelja i aplikacija najčešće su izloženi napadima upravo zato jer su na Internetu lako dostupni 24 sata na dan. Jednom kada je *web* poslužitelj kompromitiran, napadaču su otvorena alternativna vrata za upad u unutarnju mrežu poduzeća. U ovom poglavlju izložene su najčešće ranjivosti *web* poslužitelja i aplikacija, kao i neke mjere za njihovu zaštitu.

3.6.1 Ranjivosti *web* poslužitelja

Web poslužitelji, kao i svi drugi sustavi na Internetu, izloženi su napadima i često su kompromitirani. Sljedeća lista sadrži neke od najčešće iskorištavanih ranjivosti:

- Krivo postavljena konfiguracija programa pokrenutih na poslužitelju
- Greške operacijskih sustava i aplikacija te propusti u programskom kodu
- Nepromijenjena inicijalna konfiguracija programskih paketa na poslužitelju te nedostatak odgovarajućih nadogradnji
- Nepostojanje odgovarajućih sigurnosnih politika i procedura ili nepridržavanje istih

Web poslužitelji obično su smješteni u tzv. DMZ - demilitariziranoj zoni (eng. *DeMilitarized Zone*), odnosno javnom dijelu mreže smještenom između dva mrežna uređaja koji filtriraju promet (obično usmjerivači ili sigurnosne stijene). Kompromitiranje *web* poslužitelja tako je posebno opasno za organizaciju, jer je u unutarnju mrežu lakše upasti iz DMZ zone.

Najvidljiviji oblik napada na *web* poslužitelj jest obezličavanje (eng. *defacement*) *web* stranica koje se nalaze na poslužitelju. Napadači mijenjaju izgled *web* stranicama obično kako bi povećali svoju reputaciju, pa često ostavljaju svoja "hakerska" imena kao dokaz prisutnosti. Međutim, da bi uspjeli izmijeniti *web* stranice napadači prvo

moraju iskoristiti neku ranjivost kako bi dobili pristup poslužitelju. Neki od najčešćih napada kojima napadači dobivaju pristup su sljedeći:

- Dohvaćanje administratorske lozinke nekim od MitM (eng. *Man in the Middle*) napada
- Otkrivanje administratorske lozinke pomoću napada grubom silom
- Trovanje DNS poslužitelja kako bi se korisnici preusmjerili na neki drugi *web* poslužitelj
- Iskorištavanje ranjivosti *web* aplikacija pokrenutih na poslužitelju
- Iskorištavanje grešaka u konfiguraciji *web* poslužitelja
- Napad SQL injekcijom
- Iskorištavanje grešaka u radu ekstenzija *web* poslužitelja

Administratori mogu poduzeti niz mjera za tzv. "očvršćivanje" *web* poslužitelja. Tim mjerama znatno se podiže razina sigurnosti i sprječava niz mogućih napada. Neke od tih mjera su sljedeće:

- Preimenovanje administratorskog računa i korištenje sigurnih lozinki
- Brisanje nekorištenih aplikacija s poslužitelja
- Onemogućiti kretanje direktorijima u konfiguracijskim datotekama poslužitelja
- Na *web* stranicama istaknuti upozorenje o pravnim posljedicama napada
- Nadograđivanje svih programskih paketa na najnovije inačice
- Onemogućavanje daljinskog spajanja na poslužitelj
- Omogućiti praćenje prometa i događaja na poslužitelju u dnevnicima poslužitelja

3.6.2 Ranjivosti *web* aplikacija

Web aplikacije su programi koji se izvršavaju na *web* poslužitelju kako bi pružili veću funkcionalnost korisnicima od samog pregledavanja statičkih *web* stranica. One šalju upite bazama podataka te imaju funkcionalnosti, poput pregleda elektroničke pošte kroz *web* sučelje, pisanja blogova i drugih.

Web aplikacije koriste klijent-poslužitelj arhitekturu gdje se *web* preglednik na korisnikovom računalu koristi kao klijent. Zbog velike rasprostranjenosti *web* preglednika, korištenje *web* aplikacija izuzetno je praktično i popularno.

Cilj napada na *web* aplikacije najčešće je dobivanje povjerljivih informacija. *Web* aplikacije kritične su za sustav, jer se obično spajaju na baze podataka koje sadrže razne ključne informacije poput identiteta, lozinki i brojeva kreditnih kartica. Ranjivosti *web* aplikacija predstavljaju opasnost i za kompromitiranje samih *web* poslužitelja na kojima se izvršavaju.

Postoji puno prijetnji *web* aplikacijama i neke od njih su sljedeće:

- *Cross-site scripting* napad – karakteristika ovog napada je mogućnost da se ulazni parametri uneseni u *web* aplikaciju izvršavaju na poslužitelju. Pravilna obrana obuhvaća odgovarajuću provjeru ulaznih parametara.
- SQL injekcija – podmetanje SQL naredbi kako bi se iz baze podataka dobili, izmijenili ili izbrisali podaci. Protumjere također obuhvaćaju detaljnu provjeru ulaznih podataka.

- Trovanje i podmetanje "cookie" datoteka – kod ovog napada napadači pokušavaju dobiti veći pristup resursima na poslužitelju mijenjanjem ili podmetanjem "cookie" datoteka. Da bi se to spriječilo potrebno je detaljno provjeravati "cookie" datoteke i izbjegavati pohranjivanje lozinki u njih.
- Napad s preljevom spremnika – *web* aplikacijama se kroz *web* forme šalju velike količine podataka kako bi se uzrokovalo prepisivanje spremnika. Ograničavanje unosa podataka korisnicima najčešća je mjera zaštite od ovog napada.
- Krađa autentikacijske sjednice – napadač preuzima sjednicu legitimnog korisnika s poslužiteljem i tako dobiva pristup. Rješenje protiv ove vrste napada je korištenje sigurne enkripcije mrežnog prometa.

3.6.3 SQL injekcija

Napad SQL injekcijom podrazumijeva umetanje zloćudnog SQL (eng. *Structured Query Language*) programskog koda u ulazna polja *web* forme. Baš kao što legitimni korisnici unose podatke u *web* formu i time nadopunjuju upite koji se šalju bazama podataka, napadači mogu unositi i izvršavati SQL naredbe. To je tehnika koja koristi nedovoljnu provjeru ulaznih podataka koji se prosljeđuju bazi i time je kompromitira. Baze mogu sadržavati osjetljive podatke poput lozinki i brojeva kreditnih kartica, pa ranjive *web* aplikacije mogu značajno narušiti sigurnost svojih korisnika.

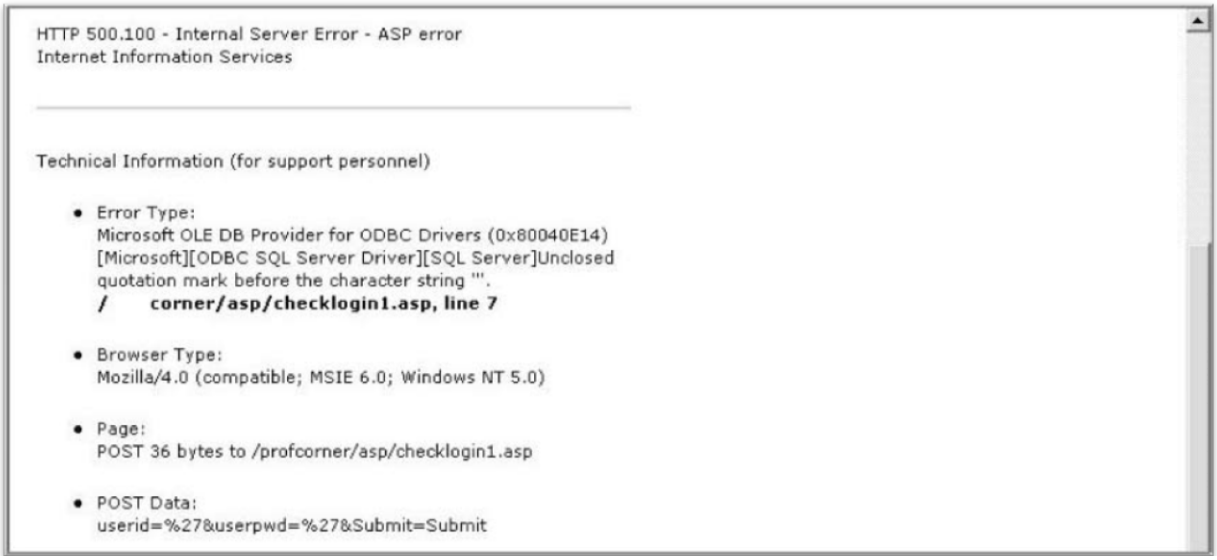
Vrste *web* aplikacija koje se napadaju su one koje korisnikov unos koriste za izradu dinamičkih SQL upita. To napadaču omogućuje podemtanje posebno oblikovane SQL naredbe i time pristup podacima u bazi, pa čak i računalu na kojem je baza pokrenuta.

Prije samog napada napadač mora provjeriti je li baza uopće ranjiva. Koraci za utvrđivanje ranjivosti su sljedeći:

1. Koristeći *web* preglednik pronaći *web* stranicu koja od korisnika traži da unese podatke. Najčešće su to stranice za pretragu sadržaja ili autenticiranje korisnika. Pregledom izvornog koda stanice provjeriti da li *web* forma šalje podatke GET ili POST metodom.
2. Ispitati bazu koristeći jednostruke navodnike kao ulazne podatke. Ukoliko poslužitelj odgovori pogreškom, vjerojatno je ranjiv na SQL injekciju.

Nakon utvrđivanja ranjivosti počinje napad umetanjem SQL koda. Najčešće se koriste sljedeće tehnike:

- zaobilaženje autorizacije (eng. *authorization bypass*)
- korištenje SELECT, UPDATE, DELETE i INSERT naredbi za manipulaciju podacima u bazi
- korištenje pohranjenih procedura iz baze



Slika 3.10. Greška kod ispitivanja ranjivosti aplikacije na SQL injekciju

Dinamički generirane SQL naredbe, bez provjere ulaznih argumenata, najčešća su meta napada SQL injekcijom. Primjer ranjivog koda u funkciji za autentikaciju korisnika napisanoj u programskom jeziku C# prikazan je na slici 3.11.

```
private void cmdLogin_Click(object sender, System.EventArgs e) {
  string strCnx = "server=localhost;database=northwind;uid=sa;pwd=;";
  SqlConnection cnx = new SqlConnection(strCnx);
  cnx.Open();
  //Ovaj dio koda je ranjiv na SQL injekciju
  string strQry = "SELECT Count(*) FROM Users WHERE UserName='" +
  txtUser.Text + "' AND Password='" + txtPassword.Text + "'";
  int intRecs;
  SqlCommand cmd = new SqlCommand(strQry, cnx);
  intRecs = (int) cmd.ExecuteScalar();
  if (intRecs>0) {
    FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false);
  }
  else {
    lblMsg.Text = "Login attempt failed.";
  }
  cnx.Close();
}
```

Slika 3.11. Primjer ranjive funkcije za autentikaciju

Zatamnjeni dio koda predstavlja dinamičko generiranje SQL naredbe koja se izvršava na SQL poslužitelju. Ako korisnik u *web* formu za korisničko ime unese John, a za lozinku `password` tada se na SQL poslužitelju izvršava naredba prikazana na slici 3.12.

```
SELECT Count(*) FROM Users WHERE UserName='John' AND Password='password'
```

Slika 3.12. Dinamički generirana SQL naredba

Međutim, ukoliko napadač za korisničko ime unese " ' OR 1=1" dobivamo drugačiju SQL naredbu, kako je prikazano na slici 3.13.

```
SELECT Count(*) FROM Users WHERE UserName='' OR 1=1 --'AND Password=''
```

Slika 3.13. SQL naredba sa zlonamjernim unosom

Kod SQL programskog jezika sve nakon oznake `--` smatra se komentarom, pa se naredba interpretira kako je prikazano na slici 3.14.

```
SELECT Count(*) FROM Users WHERE UserName='' OR 1=1
```

Slika 3.14. Interpretacija zloćudnog SQL upita na poslužitelju

Naredba se uvijek evaluira kao istina i napadač je time zaobišao sustav autentikacije.

Kod Microsoftovog SQL poslužitelja napadači mogu iskoristiti neke od ugrađenih procedura i time izvršavati naredbe na samom poslužitelju. Oblik naredbe ubačene u aplikaciju tada bi izgledao kao na slici 3.15.

```
blah';exec master..xp_cmdshell "ovdje ide naredba" --
```

Slika 3.15. SQL naredba s umetnutim funkcijama za izvršavanje naredbi

Prva mjera zaštite od SQL injekcije je korištenje korisničkog računa s manjim pravima za spajanje na bazu podataka i korištenje jakih lozinki. Također treba spriječiti prikazivanje detaljnih opisa pogrešaka korisnicima *web* aplikacija, jer tako napadač može lako pronaći ranjivu aplikaciju.

Najvažnija zaštitna mjera je svakako detaljna kontrola izvornog koda aplikacija. Sve moguće ranjivosti i pogreške moraju biti uklonjene, a podaci primljeni od korisnika moraju biti provjereni prije slanja upita bazama podataka.

3.7 Napadi s preljevom spremnika

Napadi s preljevom spremnika iskorištavaju pogreške u aplikacijama i operacijskim sustavima te se, poput napada SQL injekcijom, zasnivaju na nedovoljnoj provjeri korisničkog unosa. Greška se obično događa jer program alocira memorijski blok određene duljine, ali korisnički unos bude veći od zauzete memorije pa se dio memorije prepiše novim vrijednostima. Jednostavan primjer takvog prepisivanja kratki program sa slike 3.16, napisan u programskom jeziku C.

```
#include <stdio.h>
#include <string.h>
int main ( int argc , char **argv) {
  int i;
  char attacker[6]="AAAAAA";
  char target[5]="BBBBB";
  printf("MEMORIJA PRIJE PRELJEVA:\n");
  for (i=0; i<6; i++) printf("attacker[%u] = %c\n", attacker+i,
*(attacker+i));
  for (i=0; i<5; i++) printf("target[%u] =  %c\n", target+i, *(target+i));
  strcpy( attacker,"PPPPPPPP");
  printf("MEMORIJA NAKON PRELJEVA:\n");
  for (i=0; i<6; i++) printf("attacker[%u] = %c\n", attacker+i,
*(attacker+i));
  for (i=0; i<5; i++) printf("target[%u] =  %c\n", target+i, *(target+i));
    return 0;
}
```

Slika 3.16. Primjer prepisivanja memorijskog spremnika u programskom jeziku C

Kao što vidimo u programu su deklarirana dva znakovna niza `target` i *`attacker`* duljine 5 odnosno 11 znakova. U funkciji *`strcpy()`* na mjesto znakovnog niza *`attacker`* kopira se novi znakovni niz, ali duljine veće od rezerviranog memorijskog bloka. Zato se dio novog niza prepisuje u memorijski prostor varijable *`target`*, što možemo vidjeti kod ispisa na slici 3.17.

```
$ gcc buffer2.c
$ ./a.out
MEMORIJA PRIJE PRELJEVA:
attacker[4031362928] = A
attacker[4031362929] = A
attacker[4031362930] = A
attacker[4031362931] = A
attacker[4031362932] = A
attacker[4031362933] = A
target[4031362944] =  B
target[4031362945] =  B
target[4031362946] =  B
target[4031362947] =  B
target[4031362948] =  B
MEMORIJA NAKON PRELJEVA:
attacker[4031362928] = P
attacker[4031362929] = P
attacker[4031362930] = P
attacker[4031362931] = P
attacker[4031362932] = P
attacker[4031362933] = P
target[4031362944] =  P
target[4031362945] =  P
target[4031362946] =
target[4031362947] =  B
target[4031362948] =  B
```

Slika 3.17. Pokretanje primjera s preljevom spremnika

Programski jezik C posebno je ranjiv na napade preljevom spremnika jer posjeduje funkcije koje dozvoljavaju prepisivanje memorije. Neke od tih funkcija su `strcat()`, `strcpy()`, `sprintf()`, `vsprintf()`, `bcopy()`, `gets()` i `scanf()`. Programski jezik Java, primjerice, ne dozvoljava pohranjivanje prevelikog niza i prepisivanje memorije.

3.7.1 Vrste preljeva spremnika

Preljeve spremnika razlikujemo prema vrsti pohrane podataka u memoriji kod koje se dogodilo prepisivanje. Tako postoji:

- preljev stoga i
- preljev gomile.

Osnovna razlika između ovih vrsta pohrane je što se stog koristi za statičku alokaciju memorije, dok se na gomili memorija rezervira dinamički. Stog se koristi kod pozivanja funkcija za pohranu adrese povratka programa, pa postoji opasnost da napadač prepiše povratnu adresu i tako izvrši zloćudni kod. Primjer dinamičke alokacije memorije je pozivanje funkcije *`malloc()`* u programskom jeziku C.

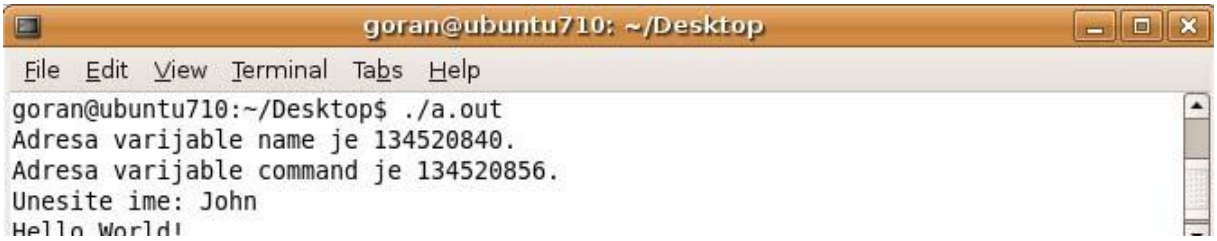
3.7.2 Primjer napada preljevom spremnika

Kod sa slike 3.18, napisan u programskom jeziku C, jednostavno ilustrira ranjivost koja može biti iskorištena za pokretanje proizvoljnog programskog koda.

```
#include <stdio.h>
main() {
  char *name;
  char *command;
  name = (char *) malloc(10);
  command = (char *) malloc(128);
  printf("Adresa varijable name je %u.\n", name);
  printf("Adresa varijable command je %u.\n", command);
  sprintf(command, "echo Hello World!");
  printf("Unesite ime: ");
  gets(name);
  system(command);
}
```

Slika 3.18. Ranjivi program

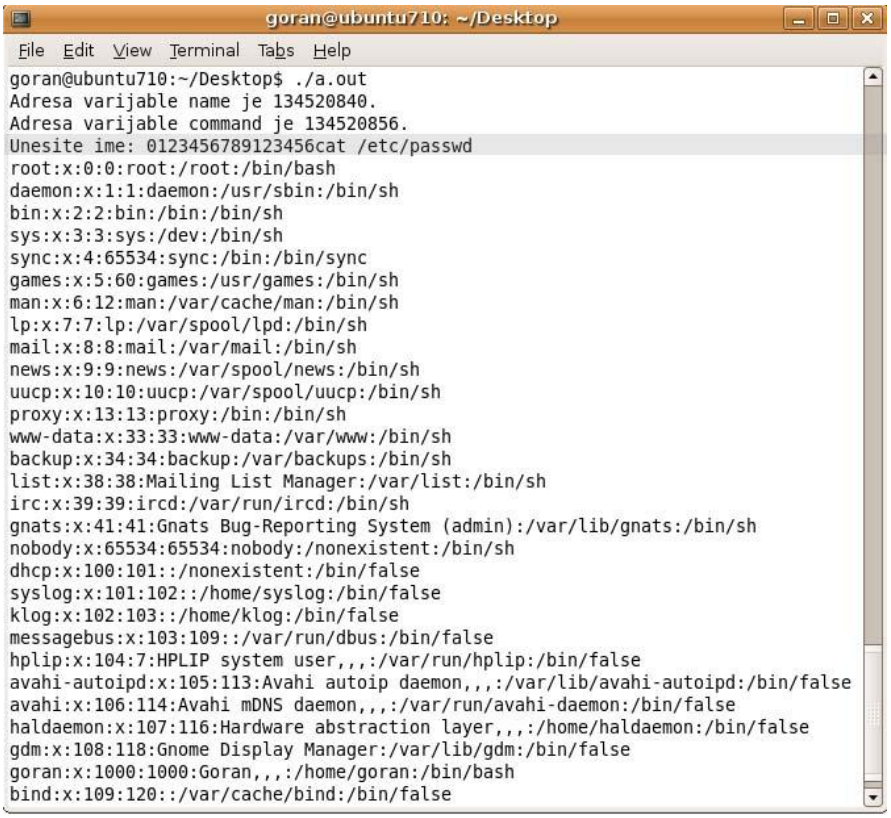
Ovaj program prvo deklarira dvije znakovne varijable i dinamički im dodjeljuje memoriju. Varijabli *`name`* dodijeljeno je 10 memorijskih lokacija, a varijabli *`command`* 128. U slučaju programskog jezika C memorijske lokacije dodijeljene ovim varijablama biti će susjedne u virtualnom memorijskom prostoru programa. Dalje, program učitava ime korisnika u varijablu *`name`* i izvodi naredbu spremljenu u varijabli *`command`*. Normalnim prevođenjem i izvođenjem ne primjećuje se nikakav propust u radu programa što ilustrira slika 3.19.



Slika 3.19. Normalno izvođenje ranjivog programa

Možemo primijetiti da je početna adresa znakovnog niza `name` udaljena 16 memorijskih lokacija od početne adrese niza `command` iako je njegova duljina samo 10 bajtova. Tih 6 bajtova dodala je funkcija `malloc()` kako bi se taj memorijski odsječak mogao normalno koristiti kada bude oslobođen.

Funkcija `gets()` učitava korisnikov unos na zadanu memorijsku lokaciju i nema argumenta koji ograničava duljinu tog unosa. Znajući to, napadač može prepisati memoriju varijable `command` unoseći preveliki znakovni niz za varijablu `name`. Dapače, pažljivo smišljenim znakovnim nizom može izvršiti proizvoljnu naredbu, jer se niz u varijabli `command` izvršava u komandnoj liniji pozivom sistemske funkcije `system()`. Primjerice može otvoriti datoteku s lozinkama naredbom `cat /etc/passwd` na Linux/UNIX sustavu. Primjer ovakvog napada prikazan je na slici 3.20.



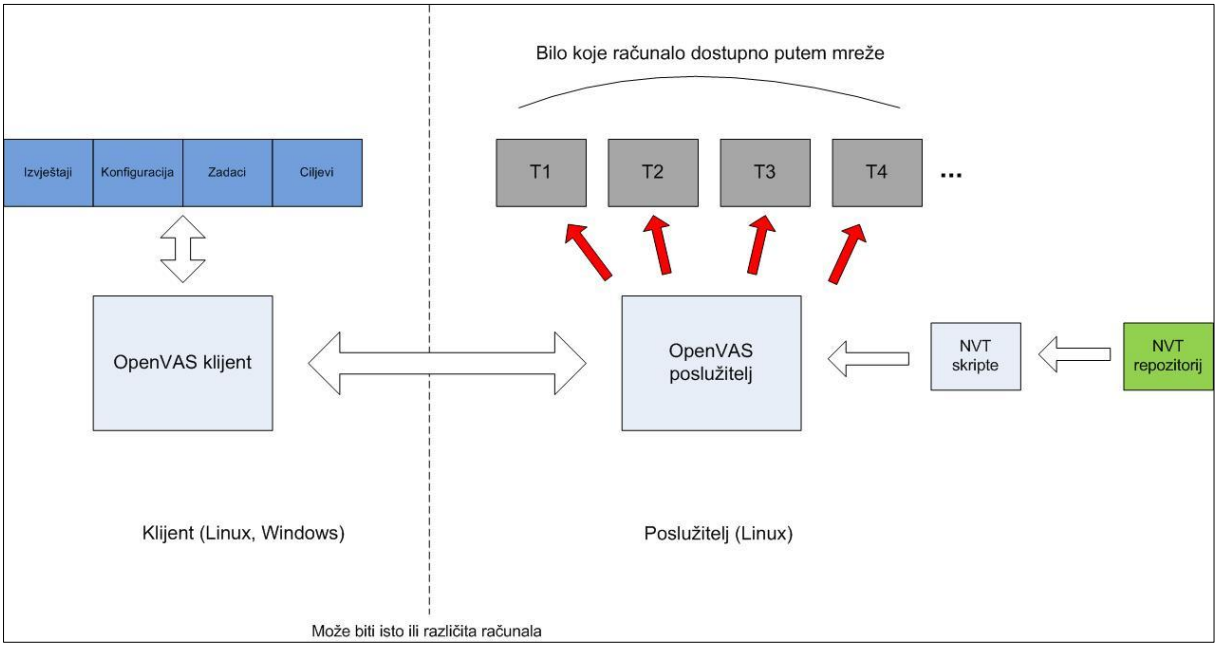
Slika 3.20. Podmetanje posebno oblikovanog znakovnog niza

Zatamnjeni redak na slici 3.20. prikazuje unos koji uzrokuje preljev spremnika i izvršavanje naredbe *cat /etc/passwd* na računalu. Vrijednost prvih 16 unesenih znakova nije važna, jer oni samo popunjavaju 16 memorijskih lokacija namijenjenih varijabli *name*. Također može se vidjeti da je izlistavanje sadržaja datoteke */etc/passwd* uspješno i time su napadaču otkrivene važne informacije za daljnji tijek napada.

4. Programski alat OpenVAS

OpenVAS (eng. *Open Vulnerability Assesment System)* je skup slobodno raspoloživih programskih alata namijenjenih automatskom otkrivanju ranjivosti računalnih sustava i mreža. Jezgru ove aplikacije čini poslužiteljska komponenta koja se koristi za izvršavanje tzv. NVT (eng. *Network Vulnerability Test*) skripti, odnosno testova za detekciju sigurnosnih problema na udaljenim računalima. U sustav je uključeno grafičko korisničko sučelje, kao i niz drugih slobodno raspoloživih alata.

Svoje korijene OpenVAS ima u drugom popularnom alatu za automatiziranu provjeru ranjivosti računalnih sustava – Nessusu. Kada je, 2004. godine, Nessus postao komercijalan alat, nekoliko sigurnosnih stručnjaka na čelu s Timom Brownom iz tvrtke Portcullis Computer Security nastavilo je razvijati posljednju inačicu Nessusa izdanu pod GPL (eng. *General Public License*) licencom. Nedugo zatim, projekt je preimenovan u OpenVAS i dalje se razvijao pod novim imenom.



Slika 4.1. Osnovna arhitektura sustava OpenVAS

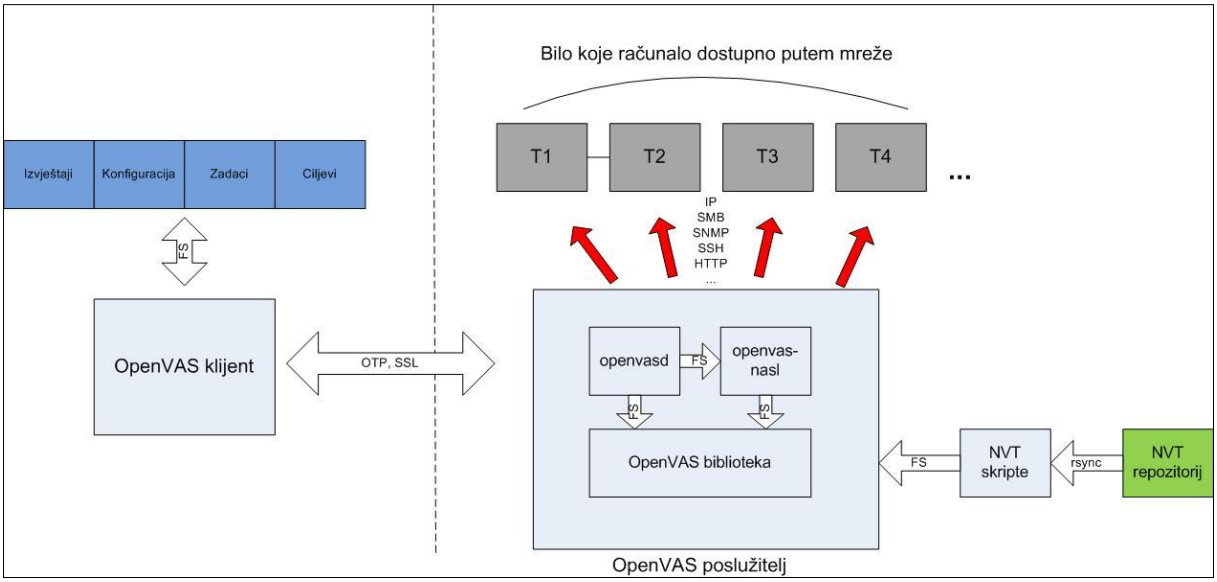
U projekt su uključeni brojni sigurnosni stručnjaci iz svih područja (akademija, komercijalne tvrtke i dr.) i različitih dijelova svijeta. Već u ranim godinama razvoja ovog perspektivnog projekta, oko njega se okupila velika zajednica ljudi koji doprinose njegovu razvoju. Danas, 5 godina nakon njegovog začetka, projekt je poprilično odmakao od Nessusa i priprema svoju treću veću nadogradnju (OpenVAS 3.0).

4.1 Arhitektura

OpenVAS je svoju arhitekturu naslijedio od Nessusa. Radi se o jednostavnoj klijent – poslužitelj arhitekturi kod koje se na poslužiteljskoj strani pokreću sigurnosni testovi,

a na klijentskoj strani je implementirano rukovanje izvještajima i konfiguracijom poslužitelja. Osnovna arhitektura sustava prikazana je na slici 4.1.

Važan element arhitekture je tzv. NVT repozitorij (o čemu će biti riječi kasnije u poglavlju) putem kojega se dobavljaju novi sigurnosni testovi.

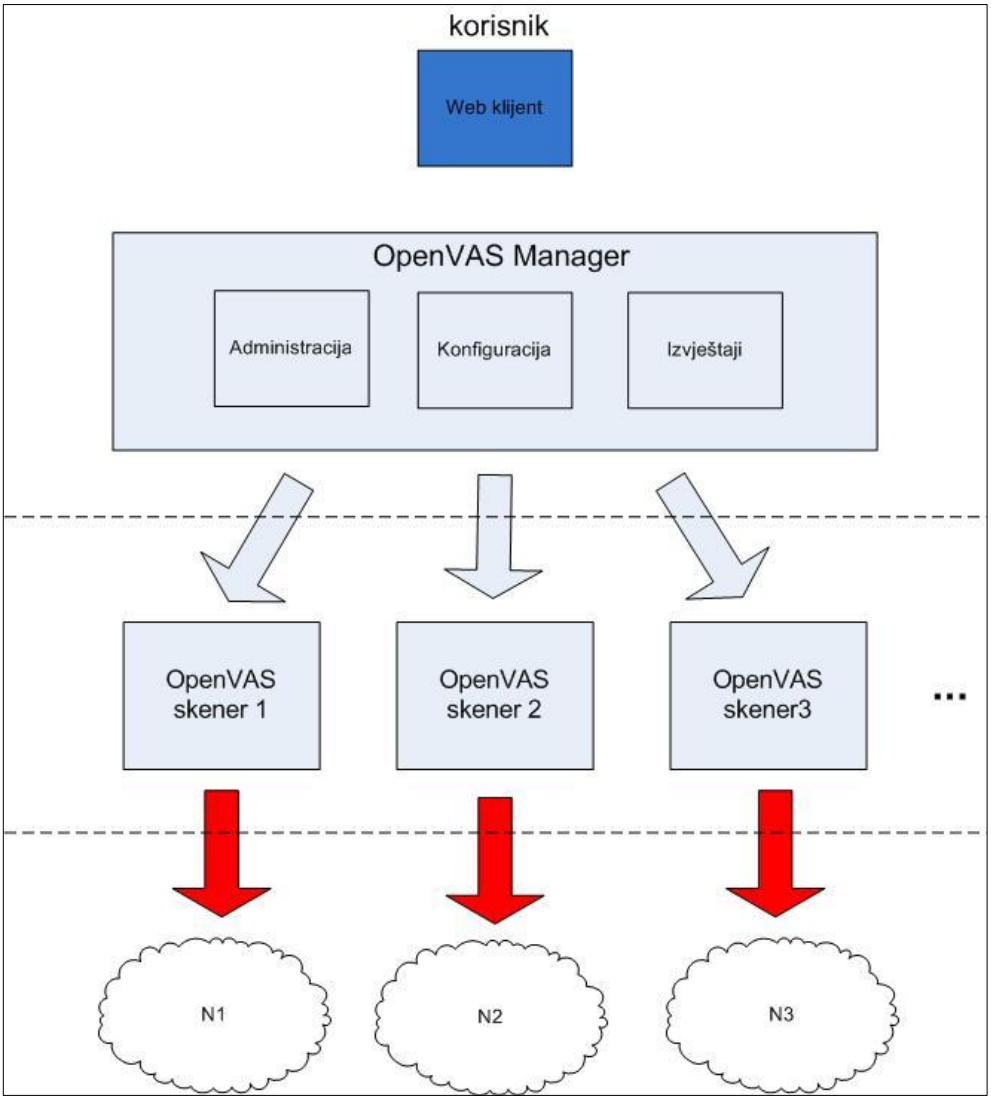


Slika 4.2. Interna arhitektura i protokoli OpenVAS-a

Na slici 4.2 prikazana je interna arhitektura OpenVAS-a i međuovisnost među njegovim modulima. Također, na poveznicama pojedinih modula prikazani su i korišteni protokoli. Tako se može vidjeti da OpenVAS poslužitelj i klijent međusobno komuniciraju OTP (eng. *OpenVAS Transfer Protocol*) protokolom. Riječ je o protokolu koji je posebno dizajniran za OpenVAS, a zasniva se na NTP (eng. *Nessus Transfer Protocol*) protokolu koji se koristi kod Nessus-a. Nove NVT skripte dohvaćaju se iz repozitorija pomoću *rsync* tehnologije. Poveznice koje imaju oznaku FS označavaju komunikaciju putem datotečnog sustava (eng. *File System*).

4.1.1 Budućnost

Arhitektura naslijeđena od Nessus-a trebala bi se u budućnosti promijeniti. Glavna promjena odnosi se na podjelu dosadašnjeg OpenVAS poslužitelja na OpenVAS *scanner* i OpenVAS *manager*. Time su administrativni zadaci odvojeni od izvođenja sigurnosnih testova. OpenVAS *manager* trebao bi upravljati s više *scanner*-a, čime bi bili postavljeni temelji za distribuiranu provjeru ranjivosti računalnih sustava. Izgled buduće arhitekture OpenVAS-a prikazan je na slici 4.3.



Slika 4.3. OpenVAS arhitektura u budućnosti

4.2 Instalacija klijenta i poslužitelja

OpenVAS se sastoji od pet osnovnih modula koje održava OpenVAS zajednica. Moduli su sljedeći:

- **OpenVAS-Server** – ovo je osnovni modul paketa i sadrži mogućnost provjere velikog broja udaljenih računala pri visokoj brzini. Budući da se svi testovi pokreću s računala na kojem je instaliran ovaj modul, pristup svim ispitivanim računalima mora biti omogućen. Sljedeća tri modula nužna su za rad poslužitelja.
- **OpenVAS-Libraries** – sadrži biblioteke funkcija koje poslužitelj koristi u svom radu.
- **OpenVAS-LibNASL** – sadrži funkcije koje poslužitelju omogućuju izvršavanje NVT skripti napisanih u jeziku NASL (eng. *Nessus Attack Scripting Language*)

- **OpenVAS-Plugins** – ovaj modul sadrži osnovni skup NVT skripti te programe nužne za dohvaćanje novih skripti iz repozitorija dostupnih na Internetu.
- **OpenVAS-Client** – program s grafičkim sučeljem koji se koristi za upravljanje poslužiteljem, provođenje provjere i obradu rezultata. Ovaj paket ne mora biti instaliran na poslužiteljskom računalu, jer se njim može upravljati preko računalne mreže.

Svi moduli namijenjeni su za rad na operacijskim sustavima Linux, ali moguće ih je, uz manje modifikacije, pokrenuti i na brojnim drugim platformama.

Binarne datoteke za instalaciju sustava OpenVAS mogu se pronaći u repozitorijima mnogih Linux distribucija iako razina konfiguracije i dostupnost najnovijih inačica paketa ovise isključivo o ažurnosti održavatelja (eng. *maintainer*) određene distribucije. Zato se za najbolje performanse savjetuje kompajliranje svih modula iz izvornog koda. Ovaj postupak će detaljnije biti opisan u nastavku.

Budući da OpenVAS koristi niz drugih paketa važno je prije instalacije osigurati da su svi dostupni na sustavu. U slučaju nedostatka nekog od paketa (eng. *missing dependency*) ispisati će se pogreška i instalacija neće uspjeti. Paketi koji su potrebni za rad OpenVAS-a su:

- libglib2
- libgpgme
- libpcap
- libgnutls
- libcrypt
- bison
- libgtk (samo za OpenVAS klijenta)

Ovi paketi obično su lako dostupni putem sustava za upravljanje paketima (eng. *package management system*) pojedine distribucije.

Nakon instaliranja svih potrebnih paketa potrebno je instalirati OpenVAS module u sljedećem redoslijedu:

- OpenVAS-Libraries
- OpenVAS-LibNASL
- OpenVAS-Server
- OpenVAS-Plugins

Kako modul OpenVAS-Client ne ovisi ni o jednom drugom modulu, on se može instalirati u bilo kojem koraku, ali ne nužno na istom računalu kao OpenVAS poslužitelj.

Kompajliranje modula iz izvornog koda obavlja se naredbama uobičajenim za Linux sustave prikazanim na slici 4.4.

```
$ ./configure --prefix=/opt/openvas
$ make
# sudo make install
```

Slika 4.4. Naredbe za instalaciju OpenVAS modula

Argument `prefix` kod pozivanja konfiguracijske skripte određuje lokaciju na koju će se moduli instalirati. Podrazumijevana lokacija je `/usr/local`.

4.3 Konfiguracija poslužitelja

Nakon procesa instalacije potrebno je obaviti još nekoliko inicijalnih koraka prije korištenja OpenVAS poslužitelja. Ti koraci obuhvaćaju stvaranje poslužiteljskog certifikata i dodavanje korisnika, i detaljnije su opisani u nastavku.

4.3.1 Stvaranje poslužiteljskog certifikata

Iz sigurnosnih razloga komunikacija između OpenVAS klijenta i poslužitelja moguća je jedino putem veze kriptirane SSL (eng. *Secure Sockets Layer*) protokolom. Kako bi se uspješno uspostavila komunikacija poslužitelj mora imati SSL certifikat. Ukoliko računalo na kojem je instaliran poslužitelj već ne posjeduje valjani certifikat potrebno je stvoriti ga.

To se najlakše može postići korištenjem programske skripte `openvas-mkcert` koja dolazi s modulom OpenVAS-Server. Pozivanjem skripte prvo se stvara tzv. CA (eng. *Certificate Authority*) certifikat, odnosno lokalni autoritativni certifikat kojim se kasnije potpisuje poslužiteljski certifikat.

4.3.2 Dodavanje korisnika

Da bi uspješno koristio usluge OpenVAS poslužitelja, svaki klijent mora imati otvoren valjani korisnički račun. OpenVAS instalacija sadrži skriptu `openvas-adduser` koja pojednostavljuje stvaranje novih korisničkih računa na poslužitelju. Korištenjem ove skripte može se odrediti način korisničke autentikacije te njegova prava pristupa.

Restriktivna prava pristupa korisna su kod sprječavanja korisnika pri provjeri proizvoljnih računala i mreža. Ova pravila mogu ograničiti korisnika na provjeru samo određenih mreža, pa čak i spriječiti ga da provjerava bilo koje računalo osim vlastitog.

Valjana sintaksa pravila koja definiraju prava pristupa prikazana je na slici 4.5.

```
accept|deny ip/mask
default accept|deny
```

Slika 4.5. Sintaksa pravila za ograničavanje korisnika

Pod oznakom `mask` misli se na mrežnu masku u CIDR (eng. *Classless Inter-Domain Routing*) formatu. Naredba `default` mora biti posljednja među pravilima jer definira podrazumijevanu politiku za korisnika.

Primjer na slici 4.6 predstavlja set pravila kod kojeg je korisniku dozvoljeno ispitivati samo mreže 192.168.1.0/24, 192.168.3.0/24 i 172.22.0.0/16:

```
accept 192.168.1.0/24
accept 192.168.3.0/24
accept 172.22.0.0/16
default deny
```

Slika 4.6. Primjer pravila za ograničavanje korisnika

Ukoliko korisniku želimo dozvoliti provjeru svih mreža osim mreže 192.168.1.0/24, pravila trebaju izgledati kao na slici 4.7.

```
deny 192.168.1.0/24
default accept
```

Slika 4.7. Primjer korištenja pravila za ograničavanje korisnika

Ključna riječ `client_ip` prilikom izvođenja programa zamjenjuje se s IP adresom korisnika. Tako, ukoliko želimo korisniku dozvoliti samo ispitivanje vlastitog računala, potrebno je postaviti pravila prikazana na slici 4.8.

```
accept client_ip
default deny
```

Slika 4.8. Primjer korištenja ključne riječi client_ip

4.3.3 Napredna konfiguracija

Podrazumijevana konfiguracija OpenVAS poslužitelja može se izmijeniti u konfiguracijskoj datoteci `/etc/openvas/openvasd.conf`. U ovoj datoteci moguće je podesiti konfiguraciju poslužitelja do najmanjih detalja. Format datoteke uobičajen je za konfiguracijske datoteke na Linux/UNIX sustavima.

4.3.4 NVT repozitoriji

Glavni izvor novih sigurnosnih testova su tzv. NVT repozitoriji. OpenVAS ima ugrađenu podršku za dohvaćanje novih testova iz ovih repozitoriju i to u vidu skripte `openvas-nvt-sync`. Ova skripta koristi *rsync* tehnologiju za dohvaćanje novih i ažuriranje postojećih programskih skripti.

Iako je instalacijom modula OpenVAS-Plugins dostupan osnovni set sigurnosnih testova, bez redovitog ažuriranja ovog skupa, provjere ranjivosti izvršene ovim alatom neće biti kvalitetne i temeljite.

Da bi se omogućio normalan rad skripte `openvas-nvt-sync` važno je da su na sustavu dostupni programski alati *rsync* i *md5sums*. Osvježavanje seta sigurnosnih provjera odvija se u sljedećim koracima:

1. Provjera konfiguracije varijabli `NVT_DIR` i `FEED` na početku skripte. Ove varijable moraju biti postavljene na željene vrijednosti: lokaciju sigurnosnih skripti i adresu NVT repozitorija.
2. Pokretanje skripte sljedećom naredbom:

```
# openvas-nvt-sync
```

Slika 4.9. Naredba za sinkronizaciju NVT skripti

3. Ponovno pokretanje OpenVAS poslužitelja:

```
# killall openvasd
# openvasd –D
```

Slika 4.10. Ponovno pokretanje OpenVAS poslužitelja

Osvježavanje sigurnosnih provjera može se automatizirati skriptom sa slike 4.11.

```
#!/bin/sh

temp=`tempfile`
openvas-nvt-sync 2>&1> $temp
if [ $? -ne  0 ]
then
      cat $temp
fi
rm $temp
if [ -f /var/lib/run/openvasd.pid ]
then
      pid=`cat /var/lib/run/openvasd.pid`
      kill -1 $pid 2>/dev/null
```

Slika 4.11. Skripta za automatsko dohvaćanje novih NVT skripti

Ukoliko skriptu za osvježavanje imenujemo `openvas-update`, potrebno je konfigurirati automatsko pokretanje skripte pomoću *crontab* datoteke kako je prikazano na slici 4.12.

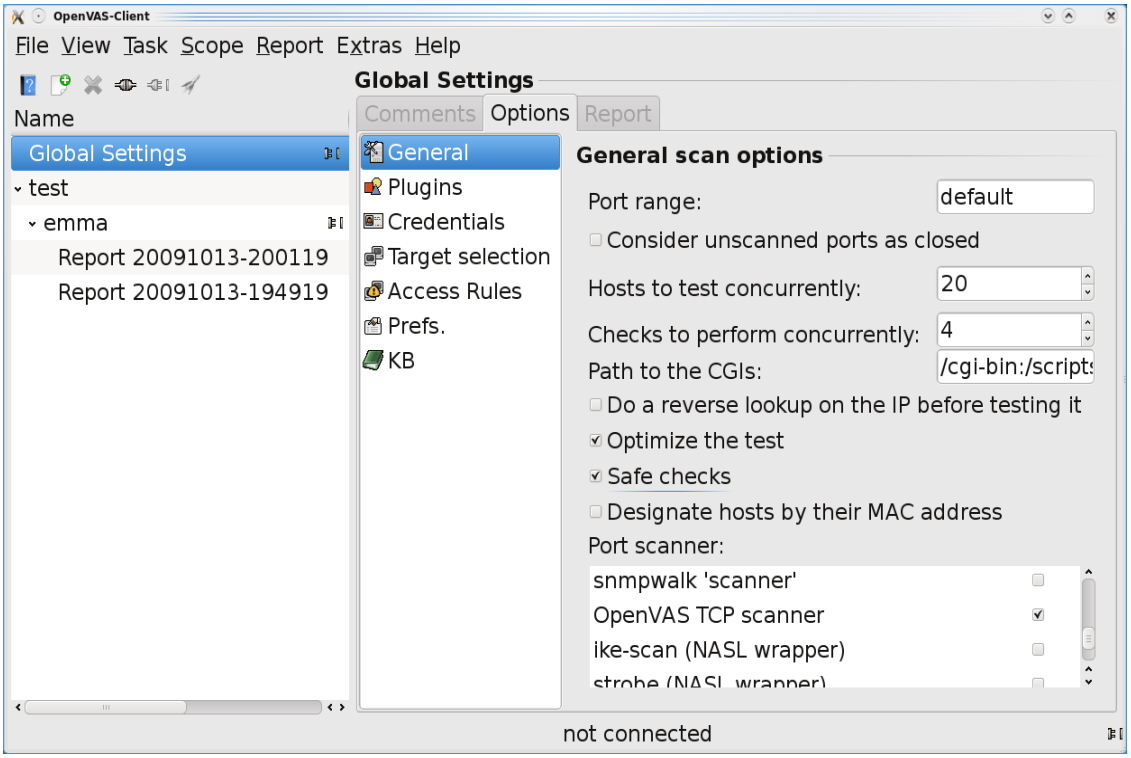
```
25 4 * * *  root  /usr/local/bin/openvas-update
```

Slika 4.12. Redak u cron datoteci

4.4 Klijent

OpenVAS klijent je program namijenjen upravljanju OpenVAS poslužiteljem te provođenju samih sigurnosnih testova. Iako sadrži grafičko sučelje može ga se koristiti i iz komandne linije. Klijent je, poput poslužitelja, prvenstveno namijenjen operacijskim sustavima Linux, no u budućnosti bi trebao biti dostupan i za Windowse.

Glavni prozor aplikacije prikazan je na slici 4.13. S lijeve strane, u obliku stabla, nalaze se lokalno pohranjeni zadaci, ciljevi i izvještaji. S desne strane nalazi se prostor za bilješke, komentare, podešavanje opcija i prikaz sadržaja izvještaja.



Slika 4.13.Glavni prozor OpenVAS klijenta

Prilikom prvog pokretanja klijenta u lijevom polju pojaviti će se samo jedan zapis – "*Global Settings*". Radi se o podrazumijevanim postavkama za provjeru koji dolaze s instalacijskim paketom. Kod ovih postavki nije moguće mijenjati selekciju sigurnosnih skripti ukoliko ne postoji veza na neki OpenVAS poslužitelj.

4.4.1 Zadaci

Zadaci su zamišljeni tako da obuhvaćaju više aktivnosti koje povezuje zajednička tema. Naziv zadatka tako može biti: "Ispitivanje računala na Zavodu" ili "Klijent ABC".

Zadatak može sadržavati razne komentare i bilješke. Tako se ovdje mogu bilježiti podaci o budućim provjerama, bilješke o problemima koji su se javili prilikom prošlih provjera i dr. Politike ispitivanja ne zadaju se u sklopu zadataka, već u sklopu ciljeva, što će biti objašnjeno kasnije. Također, zadatak može sadržavati jedan ili više ciljeva.

4.4.2 Ciljevi

U sklopu OpenVAS sustava cilj se može shvatiti kao dio zadatka. On definira određenu sigurnosnu provjeru. Naziv cilja, primjerice, može biti, "Provjera *web* poslužitelja" ili "Ispitivanje korisničkih računala" i sl. Osim komentara, kod ciljeva je moguće mijenjati opcije provjere i određivati ciljna računala. Pri stvaranju novog cilja kopiraju se opcije postavljene pod stavkom "*Global Settings*".

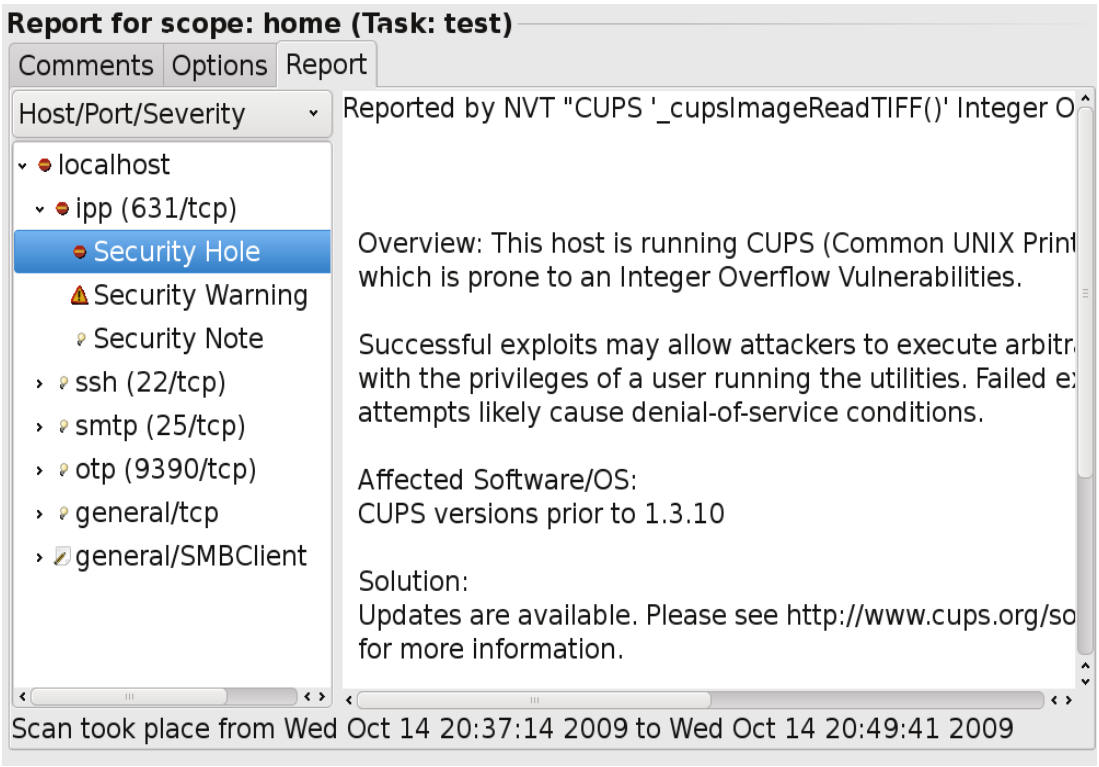
Važno je naglasiti da se veza s poslužiteljem ostvaruje u sklopu pojedinog cilja. Tako je moguće unutar jednog prozora klijenta ostvariti veze s nekoliko poslužitelja. Ciljevi mogu sadržavati više izvještaja podijeljenih prema vremenu provjere. Politike

ispitivanja korištene za određene ciljeve mogu se spremiti u datoteku odabirom opcije "*Scope->Save As*".

4.4.3 Izvještaji

Izvještaji nastaju kao rezultat samih sigurnosnih provjera. Oni sadrže sve informacije koje su sigurnosni testovi prikupili o ciljnim sustavima. Prikupljene informacije vezane su uz odgovarajući sustav predstavljen svojom IP adresom ili DNS imenom, zatim određeni mrežni pristup (eng. *port*) na sustavu te razinu ranjivosti. Svaka pronađena ranjivost okarakterizirana je jednom od sljedećih razina:

- **visoka razina** – ovo su obično ranjivosti koje zlonamjernom napadaču omogućuju postizanje potpune kontrole nad ranjivim sustavom,
- **srednja razina** – označava ranjivosti koje napadaču omogućuju kompromitiranje neke aplikacije i
- **niska razina** – označava ranjivosti koje napadaču omogućuju otkrivanje podataka o ciljnom sustavu.



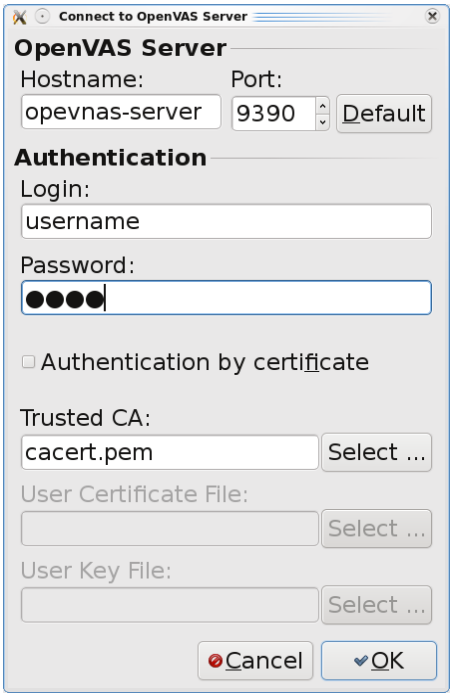
Slika 4.14. Izvještaj u OpenVAS klijentu

Izvještaj je moguće konvertirati u razne formate od kojih su najzanimljiviji HTML, PDF i XML. XML format se najčešće koristi za prijenos u sustave za obradu izvještaja. Podrazumijevani format za čuvanje izvještaja je NBE.

4.4.4 Autentikacija

Za izvršavanje same provjere i dohvaćanje novih sigurnosnih testova OpenVAS klijent mora se spojiti na poslužitelj. Kao što je već spomenuto, jedan klijent može

ostvariti više veza na različite poslužitelje. Svaka veza mora biti vezana uz pojedini cilj. Također, moguće je i spojiti se na poslužitelj sa stavkom "*Global Settings*" kako bi se izmijenila podrazumijevana politika ispitivanja.



Slika 4.15. Prozor za spajanje na poslužitelj

4.4.5 Opcije ispitivanja

Nakon spajanja na poslužitelj, OpenVAS klijent nudi niz opcija za podešavanje same provjere. U nastavku poglavlja navedene su opcije koje alat OpenVAS nudi.

General

U izborniku "*General*" nude se neke osnovne postavke za ispitivanje:

- ***Port range***
  Ova opcija omogućuje korisniku podešavanje raspona mrežnih pristupa (eng. *port*) koji će se provjeravati na svakom ispitivanom računalu. U slučaju odabira opcije "*default*" provjeravaju se mrežni pristupi navedeni u datoteci `/etc/services`.
- ***Consider unscanned ports closed***
  Ukoliko korisnik nije odabrao provjeru nekog mrežnog pristupa OpenVAS smatra taj mrežni pristup otvorenim i svi testovi koji koriste taj pristup će se izvesti. Korisnik može uključiti ovu opciju ukoliko ne želi ispitati pristupe koji nisu provjereni. Time je moguće uštediti vrijeme, ali postoji rizik propuštanja nekih ranjivosti.
- ***Number of hosts to test at the same time***
  Ovom opcijom može se podesiti broj računala koje će OpenVAS poslužitelj paralelno ispitivati.
- ***Number of checks to perform at the same time***

Ovom opcijom se određuje broj sigurnosnih testova koje će OpenVAS poslužitelj pokrenuti nad jednim računalom.

- ***Path to CGIs***
  Sigurnosni testovi koji izvode ispitivanja nad *web* aplikacijama koje koriste CGI sučelja pretražuju neke podrazumijevane direktorije na *web* poslužiteljima. To su obično direktoriji "/*cgi-bin*", "*scripts*" i sl. Ukoliko korisnik želi sam dodati neke direktorije, može to učiniti u kućici pored ove opcije.
- ***Do reverse lookup of IP before testing it***
  Ukoliko je ova opcija uključena, OpenVAS izvodi rekurzivne DNS upite za sva računala koja provjerava.
- ***Optimize the test***
  Ova opcija vezana je uz veze koje programeri stvaraju između sigurnosnih testova. Ako je ova opcija isključena svi sigurnosni testovi biti će pokrenuti. S druge strane, ako je uključena izvršiti će se samo oni testovi koji prema podacima zapisanima u bazi znanja detektiraju da je servis koji oni ispituju pokrenut.
- ***Safe checks***
  Ako je ova opcija uključena izvode se samo oni testovi koji ne mogu ugroziti sigurnost ciljnih sustava. Obično su to testovi koji svoj rad zasnivaju na čitanju pozdravnih poruka (eng. *banner*) servisa. Iako uključivanjem ove opcije korisnici mogu biti sigurni da neće srušiti servise pokrenute na ciljnim sustavima, rezultati ispitivanja biti će manje pouzdani.
- ***Designate hosts by their MAC address***
  U slučaju ispitivanja računala na lokalnoj mreži, moguće je razlikovati ciljna računala prema MAC (eng. *Medium Access Layer*) adresama. Ova opcija posebno je korisna kod ispitivanja DHCP (eng. *Dynamic Host Configuration Protocol*) mreža.
- ***Port Scanner***
  U kućici ispod ove opcije moguće je izabrati između desetak različitih skripti koje pretražuju mrežne pristupe ciljnih računala.

Plugins

Sigurnosni testovi (eng. *plugins*) pohranjeni su na strani OpenVAS poslužitelja. Kako bi se selektirati pojedini testovi za izvođenje potrebno je spojiti se na poslužitelj i dohvatiti listu dostupnih testova.

Testovi su podijeljeni u više familija, koje su kasnije nabrojane u poglavlju 5.6.2. Moguće je izabrati pojedine testove ili cijele familije testova. Budući da su testovi logično podijeljeni prema vrsti ispitivanja koje provode ili vrsti operacijskog sustava kojem su namijenjeni, preporučljivo je odabrati samo one familije koje su nužne za ispitivanje.

Credentials

Neki sigurnosni testovi, kao što su lokalne provjere, zahtijevaju korisničke podatke za prijavu na ciljni sustav. Radi se o lokalnim sigurnosnim provjerama o kojima će više riječi biti kasnije u poglavlju. U izborniku "*Credentials*" korisnici mogu unijeti odgovarajuća korisnička imena i lozinke ukoliko žele koristiti ovu vrstu provjere.

Target Selection

Pod izbornikom "*Target Selection*" moguće je odrediti računala koja će se ispitati. Adrese je moguće unijeti ručno, pročitati iz datoteke ili čak napraviti transfer zone s lokalnog DNS poslužitelja. Neki od mogućih formata za unos IP adresa prikazan je na slici 4.16.

```
192.168.0.1, 192.168.0.3, 10.0.0.2
192.168.0.0/24
192.168.0.1-24
10.0.0.7-29
```

Slika 4.16. Formati za unos IP adresa

Access Rules

U izborniku "*Access Rules*" mogu se podešavati pravila za ograničavanje provjere određenih računala. Više o pravilima može se pronaći u poglavlju 4.3.2.

Plugin Preferences

Neki sigurnosni testovi pružaju mogućnost dodatnog konfiguriranja od strane korisnika. Opcije koje ovi testovi nude mogu se podesiti u izborniku "*Plugin Preferences*".

Knowledge Base

Gotovo svi sigurnosni testovi zapisuju informacije u zajedničku bazu znanja (eng. *knowledge base*). Sve informacije zapisane tijekom jedne provjere mogu se iskoristiti u svim budućim provjerama. U izborniku "*Knowledge Base*" moguće je podesiti vrijeme čuvanja informacija u bazi znanja kao i mnoge druge napredne opcije.

4.5 Lokalne provjere

Osim standardnih sigurnosnih testova koji se izvode putem mreže, OpenVAS nudi i niz lokalnih sigurnosnih provjera. Radi se o testovima koji koriste protokole SSH (eng. *Secure Shell*) i SMB (eng. *Server Message Block*) za spajanje na UNIX/Linux, odnosno Windows sustave. Nakon spajanja ispituje se prisutnost sigurnosnih zakrpi te inačice instaliranih paketa, a sve u svrhu otkrivanja potencijalnih sigurnosnih ranjivosti. Ovi testovi su puno pouzdaniji i temeljitiji od uobičajenih testova, ali zahtijevaju dodatan trud od korisnika (unošenje lozinki, certifikati i dr.). Operacijski sustavi za koje postoje lokalni testovi u programu OpenVAS su:

- Windows
- Debian
- AIX
- CentOS
- Fedora
- FreeBSD
- Gentoo
- HP-UX
- MacOS X
- Mandrake

- Red Hat
- Solaris
- SuSE
- Ubuntu

5. Programski jezik NASL

Najveća prednost programskog alata OpenVAS je njegova modularnost. Primjerice, administrator u čijoj organizaciji sigurnosna politika zabranjuje korištenje Telnet servisa lako može napisati novi sigurnosni test koji će provjeriti prisutnost ovog servisa na svim računalima u mreži i uključiti ga u redovitu provjeru ranjivosti alatom OpenVAS. Sigurnosni testovi za OpenVAS pišu se u programskom jeziku NASL (eng. *Nessus Attack Scripting Language*). Lakoća kodiranja u ovom jeziku još je jedan razlog više za korištenje ovog alata.

Programski jezik NASL dizajniran je 1998. godine za programski alat Nessus. Prvu inačicu interpretera napisao je tvorac Nessusa Renaud Deraison kao nastavak osobnog projekta "pkt_forge". Budući da je prva inačica sadržavala znatan broj nedostataka, 2002. godine Michel Arboi je napisao novi interpreter, pa je nova inačica NASL-a poznata kao NASL2. Neke od razlika uvedenih novom inačicom interpretera su:

- NASL2 interpreter koristi Bison parser, pa NASL2 ima strožu sintaksu i može rukovati složenijim izrazima,
- NASL2 ima više ugrađenih funkcija,
- NASL2 ima više ugrađenih operatora,
- NASL2 je 16 puta brži,
- korisničke funkcije mogu rukovati nizovima i
- većina NASL skripti mogu se izvesti NASL2 interpeterom.

5.1 Prednosti i mane

Prvotna ideja prilikom osmišljavanja jezika NASL bila je stvoriti jezik koji će biti jednostavan, lako razumljiv i maksimalno prilagođen izradi sigurnosnih testova. Također, bitan faktor je bila i mogućnost jednostavnog razmjenjivanja testova među korisnicima bez obzira na operacijski sustav.

Obzirom da danas postoji veliki broj skriptnih jezika opće namjene, postavlja se pitanje – zašto NASL? Odgovor je jednostavan. Nijedan od tih jezika (Perl, Python itd.) nije siguran, u smislu da je moguće pokrenuti napisanu skriptu sa sigurnošću da se ona neće izvršiti nad nijednim drugim računalom osim onog zadanog od strane korisnika. Također, svi skriptni jezici "opće namjene" troše znatno više memorije od NASL-a. I posljednja, ali ne i manje važna, prednost NASL-a nad drugim skriptnim jezicima su ugrađene funkcije. Kod drugih skriptnih jezika potrebno je instalirati dodatne module (npr. Net::RawIP kod Perl-a) za rad s mrežnim paketima, dok su ove funkcije kod NASL-a ugrađene u sam jezik. Još neke prednosti NASL-a su:

- gramatika jezika vrlo je slična programskom jeziku C, pa većini programera savladavanje jezika ne predstavlja problem,
- sigurnosni testovi napisani u NASL-u su prenosivi i lako se prilagođavaju specifičnim potrebama korisnika,
- NASL koristi malo računalnih resursa što pogoduje izvođenu testova prema više računala odjednom.

Naravno, zbog svoje specifične primjene, NASL ima prilično ograničene mogućnosti u usporedbi sa skriptnim jezicima opće namjene. Neka ograničenja ovog jezika su:

- podržava samo jednostavne tipove podataka, npr. u NASL-u ne postoje brojevi s pomičnom točkom,
- ima manji broj ugrađenih funkcija u usporedbi s ostalim jezicima,
- sporiji je od mnogih skriptnih jezika, a raspolaganje memorijom nije optimalno riješeno,
- ne podržava strukture podataka,
- ne postoji odgovarajući alat za pronalaženje grešaka (eng. *debugger*).

Iako ova ograničenja čine NASL nepogodnim za razvoj programa druge namjene, ona ne predstavljaju veliki problem prilikom pisanja sigurnosnih testova.

5.2 Sintaksa

Veliki dio sintakse jezika NASL preuzet je od programskog jezika C. Ipak, mnogi elementi jezika C nisu prisutni u NASL-u. Tako NASL ne posjeduje objekte niti mogućnost ručne alokacije memorije, a varijable se ne moraju deklarirati. Razlog sličnosti s jezikom C je lakše učenje, što mnogim programerima čini NASL privlačnim jezikom za pisanje sigurnosnih testova. U nastavku poglavlja dan je pregled osnovnih sintaksnih elemenata jezika NASL.

5.2.1 Naredbe i komentari

Naredbe u programskom jeziku NASL izgledaju isto kao i u C-u. Tako svaka naredba završava znakom `;`. Jednostavna naredba pridruživanja u jeziku NASL prikazana je na slici 5.1.

```
a = "string";
```

Slika 5.1. Naredba pridruživanja

Komentari u NASL-u počinju oznakom `#`. Komentarom se smatra samo tekst koji se proteže od navedenog znaka do kraja tekućeg retka. Primjeri ispravnih i neispravnih komentara prikazani su na slici 5.2.

```
# ispravni komentari
a = 7; # postavi a na vrijednost 7

#
 neispravni komentari
#
a = # postavi a na vrijednost 7 # 7;
```

Slika 5.2. Primjeri komentara

5.2.2 Varijable, tipovi podataka i konstante

U NASL-u nije potrebno deklarirati varijable prije upotrebe. Intepreter vodi brigu o tipovima varijabli, pa će, u slučaju pokušaja povezivanja varijabli različitog tipa prijaviti pogrešku. Varijable se mogu eksplicitno okarakterizirati lokalnim ili globalnim, što je posebno korisno u korisnički definiranim funkcijama. Na slici 5.3 prikazan je primjer deklaracije globalnih i lokalnih varijabli.

```
local_var loc;
global_var glob;
```

Slika 5.3. Deklaracija lokalnih i globalnih varijabli

Također, nije potrebno voditi brigu o alokaciji memorije, jer se ona zauzima i oslobađa dinamički.

Jezik NASL podržava četiri osnovna tipa podataka:

- **integer** – cijeli broj
- **string** – znakovni niz
- **array** – polja stringova ili cijelih brojeva
- vrijednost **NULL**

Članovi polja dohvaćaju se operatorom `[]`. Baš kao i u jeziku C prvi element polja ima indeks 0. Osim brojevima, članove polja moguće je indeksirati i znakovnim nizovima. Slično kao kod funkcija s raspršenim vrijednostima.

Vrijednost NULL dodjeljuje se varijablama koje se koriste, a nije im eksplicitno pridijeljena vrijednost. Također, u slučaju kritične pogreške, funkcije ugrađene u jezik NASL vraćaju NULL.

Jezik NASL podržava i baratanje logičkim operacijama. Pritom je vrijednost NULL jednaka neistini, cijeli brojevi istiniti su osim u slučaju kada su 0, a znakovni nizovi su istiniti ako nisu prazni. Polja su uvijek istinita, čak i kada nemaju nijednog člana.

Cijeli brojevi se u NASL-u mogu pisati u četiri oblika: decimalnom, heksadecimalnom, oktalnom i binarnom. Na slici 5.4 prikazan je primjer zapisa broja 1903 u svim oblicima.

```
b = 1903               # decimalni oblik
b = 0x76F              # heksadecimalni oblik
b = 03557              # oktalni oblik
b = 0b11101101111      # binarni oblik
```

Slika 5.4. Zapisi brojeva u NASL-u

Znakovni nizovi se mogu prikazati u dva oblika:

- čisti (eng. *pure*) – nizovi omeđeni jednostrukim navodnicima (npr. `'string\n'`) i
- nečisti (eng. *impure*) – nizovi omeđeni dvostrukim navodnicima (npr. `"string\n"`).

Razlika između ova dva prikaza znakovnih nizova je u značenju prekidnog znaka \. Kod čistih znakovnih nizova on predstavlja znak koji se koristi za kodiranje neispisivih znakova (eng. *nonprintable characters*). U nečistim znakovnim nizovima on se ne tumači kao poseban znak.

Osim korištenja čistog oblika prikaza znakovnog niza, za unos neispisivih znakova može se koristiti i ugrađena funkcija `string()`. Više o ovoj, i drugim funkcijama za obradu znakovnih nizova, biti će riječi kasnije.

U NASL-u postoji puno predefiniranih konstanti koje programerima pomažu prilikom pisanja sigurnosnih testova. Tako, primjerice, konstante TRUE i FALSE imaju vrijednost 1 i 0, respektivno.

5.2.3 Operatori

Gotovo svi standardni operatori iz jezika C rade i u NASL-u. Na slici 5.5 je primjer s operatorima iz C-a podržanima u NASL-u.

```
a = b + 1;              # zbrajanje
a = b - 1;              # oduzimanje
a = (b * c) / d;        # mnozenje i dijeljenje
a = b % c;              # operator modulo
a = (b | c) & d;        # binarni operatori
a += 1;                 # mogu se kombinirati operatori +, -, * i / s
                        # operatorom pridjeljivanja
a = b << 2              # operator posmaka
```

Slika 5.5. Standardni operatori

Za razliku od jezika C, operatori + i − mogu se koristiti i za operacije nad znakovnim nizovima. Operator + tako koristi se za spajanje znakovnih nizova, dok se operator − koristi za uklanjanje podniza iz znakovnog niza. U sljedećem primjeru na slici 5.6 prikazane su mogućnosti korištenja ovih operatora.

```
a = "protokol";
b = "TCP je najbolji protokol";
c = a + b;  # c = "protokol TCP je najbolji protokol"
c = b - a;  # c = "TCP je najbolji "
```

Slika 5.6. Korištenje operatora nad znakovnim nizovima

Osim operatora naslijeđenih iz jezika C, postoje i dva operatora specifična za jezik NASL:

- operator '><' – koristi se za utvrđivanje prisutnosti jednog znakovnog niza unutar drugog. Vraća vrijednosti TRUE ili FALSE.

```
a = "OpenVAS";
b = "Dobro je koristiti OpenVAS";

if (a >< b) {
  # ovaj dio ce se izvesti buduci da je a unutar b
  display (a, "se nalazi unutar", b);
}
```

Slika 5.7. Korištenje operatora ><

- operator 'x' – koristi se za ponavljanje izvođenja neke funkcije zadani broj puta. U sljedećem primjeru funkcija `send_packet()` izvesti će se 19 puta. Iako bi se isto moglo izvesti programskim petljama, ovakva izvedba čini skriptu značajno bržom.

```
send_packet(udp) x 19;
```

Slika 5.8. Korištenje operatora x

5.2.4 Naredbe za kontrolu toka programa

Za kontrolu toka programa u jeziku NASL koriste se sljedeće naredbe:

- `if`
- `while`
- `repeat`
- `for`
- `foreach`

Jedina naredba koja ne postoji u jeziku C je `foreach`. Ova naredba pojavljuje se u mnogim drugim jezicima i izuzetno je korisna za dohvaćanje elemenata liste ili nizova. Na slici 5.9 prikazani su formati naredbi za kontrolu toka programa u NASL-u.

```
if ( <uvjet> )
  <blok>
else
  <blok>

while ( <uvjet> )
  <blok>

repeat
  <blok>
until ( <uvjet> );

for ( <izraz1>; <uvjet>; <izraz2> )
  <blok>
foreach var ( <polje>)
  <blok>
```

Slika 5.9. Format naredbi za kontrolu toka programa

U gornjem primjeru <blok> se odnosi na niz naredbi oblika prikazanog na slici 5.10.

```
{
  # niz naredbi
}
```

Slika 5.10. Format bloka programskih naredbi

5.2.5 Korisnički definirane funkcije

Osim korištenja ugrađenih funkcija, u jeziku NASL moguće je napisati i vlastite funkcije. Oblik korisnički definirane funkcije prikazan je na slici 5.11.

```
function ime_funkcije ( argument1, argument2, …)
{
  #
  # tijelo funkcije
  #
  return ( <vrijednost> ); # ovo je opcionalno
}
```

Slika 5.11. Korisnički definirana funkcija

Vraćanje vrijednosti nije obavezno u korisničkim funkcijama. U slučaju potrebe koristi se funkcija `return()`. Važno je primijetiti da se radi o funkciji, a ne o ključnoj riječi, pa se povratna vrijednost mora staviti u zagrade. Rekurzivni pozivi su također dopušteni, no korisničke funkcije ne smiju sadržavati druge korisničke funkcije (ugnježđivanje nije dozvoljeno). Na slici 5.12 prikazana je korisnički definirana funkcija za računanje faktorijela.

```
function fact (n)
{
  if((n == 0)||(n == 1))
    return(n);
  else
    return(n*fact(n:n-1));
}
```

Slika 5.12. Funkcija za računanje faktorijela

Funkcije bez argumenata u NASL-u pozivaju se isto kao i u C-u, no pozivanje funkcija s argumentima u nekim slučajevima se razlikuje. Argumenti funkcija u NASL-u mogu biti:

- **imenovani** – argumenti koji se deklariraju prilikom definiranja funkcije
- **neimenovani** – argumenti koji se ne deklariraju kod definiranja funkcije

Razlika je u tome što je prilikom poziva funkcije uz vrijednost imenovanog argumenta potrebno navesti i njegovo ime. Također, nije nužno navesti sve imenovane argumente niti je bitan redoslijed njihovog navođenja. Funkcija za računanje

faktorijela primjer je funkcije s imenovanim argumentima. Na slici 5.13 je primjer pozivanja takve funkcije.

```
display ( "5! = ", fact ( n:5 ), "\n" );
```

Slika 5.13. Poziv funkcije za računanje faktorijela

Neimenovane funkcije primaju samo jedan argument ili više argumenata istog tipa. Vrijednosti neimenovanih argumenata dohvaćaju se preko posebnog polja `_FCT_ANON_ARGS`. Važno je napomenuti da se preko ovog polja ne mogu dohvatiti imenovani argumenti, niti se u njega može pisati. Također, u jednoj funkciji je moguće kombinirati imenovane i neimenovane argumente. Jedna takva funkcija prikazana je na slici 5.14.

```
function zbroji ( prikazi )
{
  local_var i, rezultat;
  rezultat = 0;
  for ( i = 0; _FCT_ANON_ARGS[i]; i++ )
    rezultat += _FCT_ANON_ARGS[i];
  if (prikazi)
    display ( "Rezultat je ", rezultat );
  return (rezultat);
}

# primjeri poziva funkcije zbroji()
a = zbroji ( 1, 2, 3 );            # vrijednost se pridruzuje varijabli a
zbroji ( 1, 2, 3, prikazi: 1);     # vrijednost se prikazuje na ekranu
```

Slika 5.14. Funkcija s neimenovanim argumentima

5.3 Ugrađene funkcije

NASL sadrži veliki broj ugrađenih funkcija. Većina tih funkcija koristi se za baratanje mrežnim paketima i znakovnim nizovima jer upravo takve funkcije su potrebne stručnjacima prilikom pisanja sigurnosnih testova. U ovom poglavlju dan je pregled najvažnijih funkcija.

5.3.1 Opće funkcije

Osim funkcija namijenjenih obradi znakovnih nizova i mrežnih paketa, koje će biti detaljnije opisane kasnije, NASL sadrži i niz drugih korisnih funkcija koje pomažu programerima. To su uglavnom funkcije za kontrolu toka programa, baratanje složenim tipovima podataka i sl. Neke od tih funkcija su sljedeće:

- `exit()` – koristi se za prekid izvođenja programa. Kao argument može primiti cijeli broj koji označava uzrok prekida.
- `display()` – uzima neodređeni broj neimenovanih argumenata i ispisuje ih na zaslon. Prije ispisivanja svaki argument se pretvara u znakovni niz pozivom funkcije `string()`.

```
display( "U kosari ima ", br_jabuka, " jabuka");
```

Slika 5.15. Korištenje funkcije display()

- `isnull()` – koristi se za provjeru vrijednosti varijabli. Ukoliko varijabla nije inicijalizirana funkcija vraća vrijednost TRUE, inače vraća FALSE.
- `sleep()` i `usleep()` – kao argument uzimaju cijeli broj te čekaju zadani broj sekundi, odnosno milisekundi.
- `rand()` – koristi se za generiranje slučajnih brojeva.

5.3.2 Mrežne funkcije

Pisanje sigurnosnih testova gotovo uvijek zahtijeva napredno manipuliranje mrežnim paketima. Sukladno tome, NASL ima ugrađene funkcije za slanje, primanje i obradu paketa svih važnijih mrežnih protokola. Važno je napomenuti da ove funkcije nemaju mogućnost postavljanja odredišne IP adrese, jer NASL sadrži mehanizam koji osigurava da se svi mrežni paketi šalju samo računalu koje se provjerava.

Funkcije `open_sock_tcp()` i `open_sock_udp()` koriste se za otvaranje priključka (eng. *socket*) s TCP odnosno UDP pristupom na udaljenom računalu. Obje funkcije koriste neimenovane argumente. Primjer njihovog korištenja je na slici 5.16.

```
# otvaranje TCP prikljucka
socket1 = open_sock_tcp(80);
# otvaranje UDP prikljucka
socket2 = open_sock_udp(194);
```

Slika 5.16. Funkcije za otvaranje mrežnih priključaka

Također, postoje i funkcije `open_priv_sock_tcp()` i `open_priv_sock_udp()` koje se koriste kada korisnik želi otvoriti vezu s privilegiranog mrežnog pristupa. Priključak se zatvara pomoću funkcije `close()`.

Nakon otvaranja veze, za slanje i primanje podataka koriste se sljedeće funkcije:

- `recv(socket:< socketname>, length:<length> [,timeout : <timeout>)`
  Funkcija čita podatke duljine length s priključka socketname. Opcija timeout nije obavezna.
- `recv_line(socket:<socketname>, length:<length> [, timeout: <timeout>])`
  Funkcija radi isto što i recv(), osim što prestaje s čitanjem ako naiđe na znak za novi redak ('\n'). Ova funkcija radi samo s TCP priključcima.
- `send(socket:<socket>, data:<data> [, length:<length>])`
  Funkcija šalje podatke na priključak socket sve do duljine length ili dok ne naiđe na znak NULL.

Argument `timeout` ima podrazumijevanu vrijednost 5 sekundi. Ukoliko to vrijeme istekne, funkcija će vratiti FALSE.

Na slici 5.17 dan je jednostavan primjer čitanja pozdravne poruke, tzv. bannera, koju FTP (eng. *File Transfer Protocol*) poslužitelja šalje korisniku prilikom spajanja. U primjeru se koriste funkcije opisane ranije u tekstu.

```
soc = open_sock_tcp(21);
if(soc)
{
data = recv_line(socket:soc, length:1024);
if(data)
{
display("Banner: \n", data, "\n");
}
else
{
display("Pristup FTP posluzitelju je filtriran\n");
}
close(soc);
}
```

Slika 5.17. Čitanje bannera

Osim ovih osnovnih funkcija za otvaranje veze prema udaljenom računalu, NASL sadrži i neke funkcije za baratanje protokolima aplikacijske razine. Neke od tih funkcija su:

- `ftp_log_in(socket:<soc>, user:<login>, pass:<pass>)`
  Funkcija se preko priključka `soc` pokušava spojiti na udaljeni FTP poslužitelj s korisničkim imenom `login` i lozinkom `pass`. U slučaju uspješne autentikacije vraća TRUE, a u slučaju greške FALSE.
- `is_cgi_installed(<name>)`
  Funkcija provjerava da li CGI (eng. *Common Gateway Interface*) sučelje `name` postoji na udaljenom poslužitelju. U slučaju uspjeha vraća mrežni pristup *web* poslužitelja na kojem je sučelje pronađeno, te FALSE u slučaju greške.
- `http_<operation>(item:<item>, port:<port>)`
  Postoji niz funkcija za korištenje HTTP protokola. To su funkcije `http_get()`, `http_head()` i `http_post()`. Sve imaju isti prototip u kojem pojedini elementi imaju sljedeće značenje
  - `item` – ime *web* dokumenta koji se dohvaća,
  - `port` – pristup *web* poslužitelja na kojem se stranica nalazi.

Iako se pomoću dosad navedenih mrežnih funkcija mogu obavljati osnovna spajanja na udaljeni poslužitelj, postoji niz funkcija koje se koriste za lažiranje mrežnih paketa. Te funkcije kao argumente primaju sve elemente i polja paketa koje formiraju. Neke od tih funkcija su:

- `forge_ip_packet( ip_hl : <ip_hl>, ip_v : <ip_v>, ip_tos : <ip_tos>, ip_len : <ip_len>, ip_id : <ip_id>, ip_off : <ip_off>, ip_ttl : <ip_ttl>, ip_p : <ip_p>, ip_src : <ip_src>, ip_dst : <ip_dst>, [ip_sum:<ip_sum>])`
- `forge_tcp_packet(ip:<ip_packet>, th_sport:<source_port>, th_dport : <destination_port>, th_flags : <tcp_flags>,`

```
th_seq:<sequence_number>,th_ack:<acknowledgement_number>,
[th_x2:<unused>],        th_off:<offset>,th_win:<window>,
th_urp:<urgent_pointer>, [th_sum:<checkum>],[data:<data>])
```

- `forge_udp_packet(ip:<ip_packet>, uh_sport:<source_port>, uh_dport : <destination_port>, uh_ulen : <length>, [uh_sum : <checksum>], [data : <data>])`
- `forge_icmp_packet( ip:<ip_packet>, icmp_code:<icmp_code>, icmp_type:<icmp_type>, icmp_seq:<icmp_seq>, icmp_id:<icmp_id>, [data:<data>])`
- `forge_igmp_packet( ip:<ip_packet>, code:<igmp_code>, type:<igmp_type>, group:<igmp_group>, [data:<data>])`

Iz prototipova funkcija vidljivo je da je, ovisno o protokolu, moguće ručno podesiti većinu polja željenog paketa. Dok je za IP, TCP i UDP pakete moguće unijeti vrijednosti gotovo svih postojećih polja, protokoli ICMP (eng. *Internet Control Message Protocol*) i IGMP (eng. *Internet Group Management Protocol*) nisu još u potpunosti podržani. Važno je napomenuti da se vrijednost prenesena parametrom `ip_dst` kod funkcije `forge_ip_packet()` ignorira jer, kao što je već spomenuto, NASL skripte se izvode samo na udaljenim računalima zadanima prilikom izvođenja. Sve spomenute funkcije vraćaju novostvorene pakete. Vrijednosti polja u paketima mogu se dohvaćati, ili dodatno konfigurirati sljedećim funkcijama:

- `get_ip_element()`
- `set_ip_elements()`
- `get_tcp_elements()`
- `set_tcp_elements()`
- `get_udp_elements()`
- `set_udp_elements()`
- `get_icmp_element()`
- `set_icmp_elements()`

Navedene funkcije kao argumente primaju pakete te polja koja treba promijeniti. Nakon stvaranja, paketi se šalju pomoću funkcije `send_packet()`.

5.3.3 Funkcije za obradu znakovnih nizova

Prilikom pisanja sigurnosnih tekstova često se javlja potreba za naprednijom obradom znakovnih nizova. To može značiti traženje određenog uzorka u nekom znakovnom nizu, pa sve do slaganja zloćudnih znakovnih nizova koji provjeravaju udaljeni *web* poslužitelj na poznatu ranjivost. Upravo zato, jezik NASL sadrži veliki broj ugrađenih funkcija za napredno baratanje znakovnim nizovima.

U jeziku NASL znakovni nizovi se tretiraju jednako kao i brojevi. Zato je moguće upotrijebiti neke standardne operatore (poput ==, < i >) nad znakovnim nizovima. U primjeru na slici 5.18 uspoređuju se dva znakovna niza koja predstavljaju otkrivene inačice nekog paketa.

```
a = "version 1.2.3";
b = "version 1.4.1";
if(a < b){
  #
  # Naredbe u ovom bloku ce se izvrsiti jer je 1.2.3 manja od 1.4.1
  #
}
```

Slika 5.18. Uspoređivanje znakovnih nizova

Znakovni nizovi se, također, mogu i zbrajati i oduzimati kako je prikazano na slici 5.19.

```
a = "version 1.2.3";
b = a - "version ";     # b je jednak "1.2.3"
a = "ovo je test";
b = " je ";
c = a - b;              # c je jednak "ovo test"
a = "test";
a = a+a;                # a je jednak "testtest"
```

Slika 5.19. Operacije sa znakovnim nizovima

Osim mogućnosti standardnih operatora usporedbe, zbrajanja i množenja, te prije spomenutog operatora ><, NASL sadrži i niz drugih funkcija za napredno baratanje znakovnim nizovima:

- `ereg(pattern:<pattern>, string:<string>)`
  Funkcija `egrep()` koristi se za pronalaženje uzorka unutar znakovnog niza. Uzorci se zadaju u standardnom formatu regularnih izraza.

```
if(ereg(pattern:".*", string:"test"))
{
  display("Always executed\n");
}

mystring = recv(socket:soc, length:1024);
if(ereg(pattern: "SSH-.*-1\..*", string : mystring))
{
  display("SSH 1.x je pokrenut na ovom racunalu");
}
```

Slika 5.20. Korištenje funkcije ereg()

- `ereg_replace(pattern:<pattern>, replace:<replace>, string:<string>);`
  Za razliku od funkcije `ereg()`, `ereg_replace()` ima dodatnu mogućnost zamjene pronađenog uzorka nekim novim uzorkom.
- `egrep(pattern : <pattern>, string: <string>)`
  Funkcija vraća prvi redak unutar znakovnog niza string u kojem je pronađen uzorak `pattern`. U slučaju neuspjeha vraća FALSE.

```
# programski kod za pronalazak tipa web posluzitelja pomocu
# funkcije egrep()
soc = open_soc_tcp(80);
str = string("HEAD / HTTP/1.0\r\n\r\n");
send(socket:soc, data:str);
r = recv(socket:soc, length:1024);
server = egrep(pattern:"^Server.*", string : r);
if(server)display(server);
```

Slika 5.21. Korištenje funkcije egrep()

- `crap(length:<length>, [data:<data>])`
  Funkcija `crap()` posebno je korisna kod izvođenja napada se preljevom spremnika. Njena osnovna funkcionalnost je stvaranje znakovnog niza s duljinom `length`. Argument `data` je opcionalan, a predstavlja uzorak koji se koristi za stvaranje niza. Podrazumijevani uzorak je znak '`X`'.

```
a = crap(5);            # a = "XXXXX";
b = crap(4096);         # b = "XXXX...XXXX" (4096 X-eva)
c = crap(length:12,     # c = "hellohellohe" (duljina = 12);
data:"hello");
```

Slika 5.22. Korištenje funkcije crap()

- `strlen(string)`
  Funkcija `strlen()` kao argument prima znakovni niz, a vraća njegovu duljinu.

```
a = strlen("abcd"); # a je jednak 4
```

Slika 5.23. Korištenje funkcije strlen()

- `raw string()`
  Funkcija `raw_string()` kao argumente prima niz cijelih brojeva koje pretvara u analogni niz znakova.

```
a = raw_string(80, 81, 82); # a je jednak 'PQR'
```

Slika 5.24. Korištenje funkcije raw_string()

5.4 Biblioteke funkcija

Osim ugrađenih funkcija, za programski jezik NASL napisane su brojne biblioteke koda. Tu se većinom radi o bibliotekama koje olakšavaju rad s raznim mrežnim protokolima. Većina tih biblioteka, barem one besplatne, datiraju još iz razdoblja dok je Nessus bio slobodno raspoloživ alat. Ostale su uglavnom izradili sigurnosni stručnjaci koji razvijaju alat OpenVAS.

Biblioteke koje dolaze s programskim paketom OpenVAS nalaze se u datotekama s nastavkom ".inc". Za korištenje funkcija iz ovih biblioteka potrebno je uključiti ih u skriptu naredbom `include()`.

```
include (openvas-https.inc);
```

Slika 5.25. Uključivanje vanjskih biblioteka

Sve biblioteke dostupne kod korištenja alata OpenVAS može se izlistati naredbom `ls` kako je prikazano na slici 5.26.

```
goran@gubuntu:/opt/openvas/lib/openvas/plugins$ ls *.inc
backport.inc                  ldap.inc                pkg-lib-deb.inc
secpod_ie_supersede.inc       solaris.inc             version_func.inc
debian_package.inc            misc_func.inc           pkg-lib-gentoo.inc
secpod_reg.inc                ssh_func.inc            wmi_file.inc
default_account.inc           netop.inc               pkg-lib-hpux.inc
secpod_smb_func.inc           ssl_funcs.inc           wmi_hardware.inc
dump.inc                      network_func.inc        pkg-lib-rpm.inc
sip.inc                       telnet_func.inc         wmi_misc.inc
ftp_func.inc                  nfs_func.inc            plugin_feed_info.inc
slackware.inc                 tftp.inc                wmi_os.inc
global_settings.inc           nntp_func.inc           pop3_func.inc
slad_ssh.inc                  toolcheck.inc           wmi_proc.inc
http_func.inc                 openvas-https.inc       qpkg.inc
smbcl_func.inc                ubuntu.inc              wmi_rsop.inc
http_keepalive.inc            pingpong.inc            revisions-lib.inc
smb_nt.inc                    uddi.inc                wmi_svc.inc
imap_func.inc                 pkg-lib-bsd.inc         secpod_activex.inc
smtp_func.inc                 url_func.inc            wmi_user.inc
```

Slika 5.26. Popis vanjskih biblioteka alata OpenVAS

5.5 Baza znanja

Prilikom svake sigurnosne provjere pokreću se tisuće sigurnosnih testova. Kako bi se izbjeglo da testovi rade dupli posao omogućena je razmjena informacije među testovima pomoću tzv. baze znanja (eng. *knowledge base*).

Baza znanja podijeljena je u kategorije. Tako, primjerice, kategorija "Services" sadrži brojeve mrežnih pristupa na kojem se nalaze otkriveni servisi. Očekivano je da će element "Services/smtp" sadržavati vrijednost 25, jer je to uobičajeni mrežni pristup za SMTP (eng. *Simple Mail Transfer Protocol*) servis, no on može biti i sakriven npr. na pristupu 2500. Ukoliko neki od pokrenutih sigurnosnih testova to otkrije, može to zapisati u bazu znanja i time omogućiti drugim testovima da iskoriste ovu informaciju.

Čitanje i pisanje u bazu znanja omogućeno je sljedećim funkcijama:

- `get_kb_item(<name>)`
  Funkcija dohvaća vrijednost elementa name iz baze znanja.
- `set_kb_item(name:<name>, value:<value>)`
  Funkcija zapisuje vrijednost value u bazu znanja na mjesto elementa name.

5.6 Pisanje NVT skripti

Da bi se skripte napisane u programskom jeziku NASL mogle koristiti u alatima kakav je OpenVAS one moraju imati posebnu strukturu. Prilikom prvog pokretanja skripte, alatu moraju predati određene informacije (identifikator, ime, opis, autor itd.) kako bi ih on na neki način "registrirao". Ove informacije programu se predaju u dijelu skripte koji se naziva "registracija". Drugi dio skripte je samo ispitivanje ranjivosti. Nakon izvođenja testa, skripta ponovo mora predati programu informaciju o postojanju ranjivosti. Ova informacija predaje se putem posebnih funkcija koje će biti opisane u nastavku ovog poglavlja.

5.6.1 Struktura skripte

Svaka NVT skripta pisana za programski alata OpenVAS mora imati dva dijela:

- **registracija** – u ovom dijelu se pomoću posebnih funkcija OpenVAS-u predaju sve informacije o skripti i ranjivosti koju ispituje,
- **ispitivanje** – dio skripte u kojem se obavlja provjera ranjivosti na udaljenom sustavu.

Struktura skripte prikazana je na slici 5.27.

```
#
# NVT skripta napisana za program OpenVAS
#
if(description)
{
#
# REGISTRACIJA
#
exit(0);
}
#
# ISPITIVANJE
#
```

Slika 5.27. Struktura NVT skripte

Varijabla description je ugrađena u programski alat OpenVAS. Prilikom prvog pokretanja skripte (registracija) ova varijabla je postavljena na vrijednost 1. Tako se izvrše samo funkcije koje obavljaju registraciju skripte (zbog poziva funkcije `exit()`). U drugom pozivu skripte (ispitivanje) varijabla je postavljena na 0.

5.6.2 Registracija

Dio skripte koji izvodi registraciju mora sadržavati sljedeće funkcije:

- `script_id(<id>)`
  Ova funkcija postavlja jedinstveni identifikator za skriptu. Kod programskog paketa OpenVAS svaki programer dobiva raspon slobodnih identifikatora koji može koristiti za svoje skripte.

- `script_version(<version>)`
  U ovoj funkciji postavlja se inačica skripte. Oblik određivanja inačica ostavljen je na izbor programeru.
- `script_name(<name>)`
  Postavlja ime skripte.
- `script_description(<desc>)`
  Ovoj funkciji se kao argument predaje opis skripte. Tu se misli na opis ranjivosti koja se skriptom ispituje i način rada skripte.
- `script_summary(<summary>)`
  Sadrži kratki opis funkcije koju skripta obavlja (ili ranjivosti koju provjerava).
- `script_category(<category>)`
  Sadrži kategoriju kojoj skripta pripada. Svaka skripta mora biti svrstana u jednu od sljedeće četiri kategorije:
  - `ACT_ATTACK` – za skripte koje izvode napade koji mogu napadačima dobaviti određene privilegije na sustavu,
  - `ACT_GATHER_INFO` – za skripte koje ne izvode napade već samo prikupljaju informacije o sustavu,
  - `ACT_DENIAL` – za skripte koje mogu ozbiljno naštetiti ispitivanom sustavu (npr. srušiti neki servis),
  - `ACT_SCANNER` – za skripte koje pretražuju mrežne pristupe i servise.
- `script_copyright(<copyright>)`
  Funkcija koja prima informacije o pravima kopiranja skripte i autoru.
- `script_family(<family>)`
  Svaka skripta svrstava se u jednu od postojećih familija. Ovo pomaže korisnicima da lakše odaberu koje skripte žele pokrenuti u ovisnosti o vrsti ciljnih sustava (Windows računala, poslužitelji i dr.). Familije koje postoje u programu OpenVAS su:
  - *Brute force attacks*
  - *Web application abuses*
  - *CISCO*
  - *Default Accounts*
  - *Denial of Service*
  - *Finger abuses*
  - *Firewalls*
  - *FTP*
  - *Gain a shell remotely*
  - *Netware*
  - *Peer-To-Peer File Sharing*
  - *Port scanners*
  - *Remote file access*
  - *RPC*
  - *Service detection*
  - *Settings*
  - *SMTP problems*
  - *SNMP*
  - *Useless services*
  - *Windows : Microsoft Bulletins*

- *Windows*
- *AIX Local Security Checks*
- *Debian Local Security Checks*
- *FreeBSD Local Security Checks*
- *Gentoo Local Security Checks*
- *MacOS X Local Security Checks*
- *Red Hat Local Security Checks*
- *Solaris Local Security Checks*
- *SuSE Local Security Checks*
- *Fedora Local Security Checks*
- *CentOS Local Security Checks*
- *Ubuntu Local Security Checks*
- *Mandrake Local Security Checks*
- *HP-UX Local Security Checks*
- *SLAD*
- *Web Servers*
- *Buffer overflow*
- *Privilege escalation*
- *Credentials,*
- *Malware*
- *Databases*
- *General*

5.6.3 Ispitivanje

U dijelu skripte namijenjenom samom ispitivanju nalazi se programski kod koji utvrđuje postojanje neke ranjivosti na udaljenom računalu. Osim naredbi namijenjenih utvrđivanju ranjivosti, ovaj dio skripte sadrži i naredbu za prijavljivanje ranjivosti (ukoliko ona postoji). Ova naredba također se koristi za interakciju s alatom OpenVAS koji informacije dobivene ovim putem prosljeđuje korisnicima u obliku izvještaja. Funkcije koje se mogu iskoristiti za prijavljivanje ranjivosti su:

- `security_info(port:<port>, data:<data> [,protocol:<proto>]);`
  Ova funkcija koristi se kada skripta ne prijavljuje ranjivost već samo neke informacije o ispitivanom sustavu (operacijski sustav, otkrivene servise i sl.)
- `security_warning(port:<port>, data:<data> [,protocol:<proto>]);`
  Ova funkcija koristi se za prijavljivanje ranjivosti srednje razine rizika.
- `security_hole(port:<port>, data:<data> [,protocol:<proto>]);`
  Pomoću ove funkcije prijavljuju se ozbiljne ranjivosti koje mogu narušiti sigurnost ispitivanog sustava.

5.6.4 Primjer

U nastavku (slika 5.28) je dan primjer NVT skripte koja provjerava prisutnost servisa SSH (eng. *Secure Shell*) na udaljenom računalu. Skripta ima strukturu koju zahtjeva alat OpenVAS. Ovo je dobar primjer kod kojeg se može vidjeti kako se jezik NASL i alat OpenVAS mogu iskoristiti i za drugačije provjere od samih sigurnosnih ranjivosti.

```
#
# Provjera SSH servisa
#
if(description)
{
  script_name("Provjera servisa SSH");
  script_description("Ova skripta provjerava prisutnost servisa SSH");
  script_summary("spajanje na TCP pristup 22");
  script_category(ACT_GATHER_INFO);
  script_family(english:"Service detection");
  script_copyright(english:"Copyright XYZ");
  script_dependencies("find_service.nes");
  exit(0);
}
#
# servis SSH moze biti pokrenut na bilo kojem mreznom pristupu
# zato se prvo provjerava baza znanja
#
port = get_kb_item("Services/ssh");
if(!port)port = 22;
# pretpostavljamo da servisa nema
ok = 0;
if(get_port_state(port))
{
  soc = open_sock_tcp(port);
  if(soc)
  {
    # provjeravamo da su podaci poslani od SSH servisa
    data = recv(socket:soc, length:200);
    if("SSH" >< data)ok = 1;
  }
  close(soc);
}
#
# upozorenje korisniku
#
if(!ok)
{
  report = "SSH servis nije pokrenut !";
  security_warning(port:22, data:report);
}
```

Slika 5.28. NVT skripta za provjeru SSH usluge

6. Praktični rad

U praktičnom dijelu ovog diplomskog rada ostvarene su programske skripte za detekciju operacijskog sustava, servisa *rsync* i *web* aplikacije WebAPP na udaljenom računalu. Sve skripte ostvarene su u programskom jeziku NASL.

6.1 Detekcija operacijskog sustava udaljenog računala

Programska skripta za detekciju operacijskog sustava udaljenog računala implementirana je po uzoru na program otvorenog koda *xprobe*. Riječ je o programu koji za detektiranje operacijskog sustava koristi tzv. X logiku. X logika koristi analizu različitih ICMP (eng. *Internet Control Message Protocol*) paketa primljenih od udaljenog računala za utvrđivanje njegovog operacijskog sustava.

6.1.1 X logika

Idejni začetnici X logike su Fyodor Yarochkin i Ofir Arkin. Ova logika nastala je kao posljedica rezultata studije "ICMP Usage in Scanning" Ofira Arkina iz 2000. godine. Prilikom izrade ove studije i proučavanja ICMP protokola, autori su primijetili da različiti operacijski sustavi na različite načine implementiraju ICMP protokol u okviru svojeg TCP/IP stoga. Upravo te razlike i nekonzistentnosti u implementaciji standardnog ICMP protokola mogu se iskoristiti za razlikovanje operacijskih sustava.

Ovaj način detekcije operacijskih sustava nudi se kao alternativa klasičnim metodama koje su dosad koristile uglavnom pakete TCP (eng. *Transfer Control Protocol*) protokola za razlikovanje različitih implementacija TCP/IP stoga. Ove klasične metode posebne su probleme imale s razlikovanjem različitih inačica operacijskih sustava Windows, budući da su promjene u TCP/IP stogu bile minorne.

Još jedna od prednosti ICMP metode detekcije jest mali broj mrežnih datagrama koje je potrebno izmijeniti s ciljnim sustavom kako bi se otkrio njegov operacijski sustav. Obično se koristi pet različitih ICMP poruka koje se šalju ciljnom sustavu, no čak i slanje samo jedne takve poruke može pomoći u razlikovanju do 8 različitih operacijskih sustava. U najgorem slučaju se šalje 5 mrežnih datagrama i isto toliko ih se prima, što ovu metodu čini bržom od uobičajenih.

Osim brzine, prednost X metode je i nevidljivost za IDS (eng. *Intrusion Detection System*) sustave. Razlog tome je što se ovom metodom ne šalju nikakvi maliciozni paketi. Radi se o paketima koji se na normalnim mrežama pojavljuju svakodnevno i ne predstavljaju nikakvu opasnost za računala na mreži.

Slijede primjeri nekih nekonzistentnosti u implementaciji ICMP protokola kod raznih operacijskih sustava:

- **veličina podatkovnog dijela ICMP poruke o grešci**
  Svaka ICMP poruka o grešci sadrži IP zaglavlje i barem prvih 8 podatkovnih bajtova datagrama koji je uzrokovao pogrešku (eng. *offending datagram*). Prema dokumentu RFC 1122 može biti poslano i više od 8 bajtova. Dok većina doista šalje samo prvih 8 bajtova, neki operacijski sustavi u svojim ICMP porukama šalju više od 8 bajtova. Neki od tih sustava su Linux sustavi

zasnovani na jezgri 2.0.x/2.2.x/2.4.x, zatim Sun Solaris 2.x, MacOS X 7.x-9.x, HP-UX 11.x i dr.

- **polja originalnog paketa iz ICMP poruke o grešci**
  Mnogi operacijski sustavi mijenjaju razna polja iz IP zaglavlja datagrama koji je uzrokovao slanje ICMP poruke o grešci. To je još jedna karakteristika koju je moguće iskoristiti za razlikovanje operacijskih sustava. Neka od polja koja se često mijenjaju su:
  - *IP Total Length*
  - *IPID*
  - *Offset*
  - *IP Header Checksum*
  - *UDP Header Checksum*
- ***Precedence* bitovi kod ICMP poruka o grešci**
  Svaki IP paket ima 8-bitno polje "TOS" (eng. *Type of Service*). "TOS" polje dijeli se na:
  - *Precedence* – 3-bitno polje koje se koristi za određivanje prioriteta paketa,
  - *Type-of-Service* – koristi se za određivanje načina prosljeđivanja IP paketa,
  - *MBZ* (eng. *Must Be Zero*) – polje koje se ne koristi i mora biti 0.

  Prema dokumentu RFC 1812 ICMP *Source Quench Error* poruke moraju imati *Precedence* polje postavljeno na istu vrijednost kao i poruka koja je izazvala pogrešku. Sve ostale ICMP poruke o grešci moraju ovo polje imati postavljeno na vrijednost 6 ili 7. Budući da mnogi operacijski sustavi ne poštuju ovo pravilo, to je još jedan od faktora pomoću kojih ih je lako razlikovati.

Ovo su samo neki od mnogih primjera nekonzistentnosti u raznim implementacijama ICMP protokola. U ostvarenoj programskoj skripti analizira se pet različitih ICMP poruka o greškama i prema njima se određuje operacijski sustav udaljenog računala. ICMP poruke koje se ispituju, kao i odgovarajuća polja u njima su sljedeći:

- *ICMP Echo*
  - `icmp_echo_reply`
  - `icmp_echo_code`
  - `icmp_echo_ip_id`
  - `icmp_echo_tos_bits`
  - `icmp_echo_df_bit`
  - `icmp_echo_reply_ttl`
- *ICMP Timestamp*
  - `icmp_timestamp_reply`
  - `icmp_timestamp_reply_ip_id`
  - `icmp_timestamp_reply_ttl`
- *ICMP Address Mask*
  - `icmp_addrmask_reply`
  - `icmp_addrmask_reply_ip_id`
  - `icmp_addrmask_reply_ttl`
- *ICMP Info Request*
  - `icmp_info_reply`

    - icmp_info_reply_ip_id
    - icmp_info_reply_ttl
- *ICMP Port Unreachable*
    - icmp_unreach_reply
    - icmp_unreach_precedence_bits
    - icmp_unreach_df_bit
    - icmp_unreach_ip_id
    - icmp_unreach_echoed_dtsize
    - icmp_unreach_reply_ttl
    - icmp_unreach_echoed_udp_cksum
    - icmp_unreach_echoed_ip_cksum
    - icmp_unreach_echoed_ip_id
    - icmp_unreach_echoed_total_len
    - icmp_unreach_echoed_3bit_flags

6.1.2 Implementacija

X logika je implementirana u programskoj skripti preko modula koji analiziraju odgovarajuće ICMP poruke primljene od ciljnog sustava. Rezultati analize svakog modula spajaju se u jedinstveni otisak (eng. *fingerprint*) operacijskog sustava. Otisci se uspoređuju s bazom već poznatih te se s određenom vjerojatnošću utvrđuje jedan ili više mogućih operacijskih sustava. Primjer jednog otiska iz baze prikazan je na slici 6.1. Svi otisci u ovoj programskoj implementaciji dobiveni su iz datoteke `xprobe.conf`.

```
FreeBSD 5.4,y,!0,!0,!0,1,<64,y,!0,<64,n,!0,<64,n,!0,<64,y,0,1,!0,8,<64,0,OK,OK,OK,OK
```

Slika 6.1. Otisak operacijskog sustava FreeBSD 5.4

Moduli su izrađeni pomoću NASL funkcija za lažiranje IP paketa koje se koriste za slaganje ICMP zahtjeva. Funkcije koje se koriste su `forge_ip_packet()`, `forge_udp_packet()` i `forge_icmp_packet()`. Svi paketi koji se šalju ispitivanom sustavu, osim paketa u funkciji `ModuleE()`, su paketi ICMP protokola. Program je lako nadograditi i novim modulima koji koriste pakete drugih protokola za poboljšanu detekciju. U funkciji `ModuleE()` ciljnom sustavu se šalje UDP paket na slučajni mrežni pristup, jer se na taj način uzrokuje slanje ICMP *Port Unreachable* poruke. Primjer slaganja ICMP *Address Mask* paketa iz funkcije `ModuleC()` prikazan je na slici 6.2.

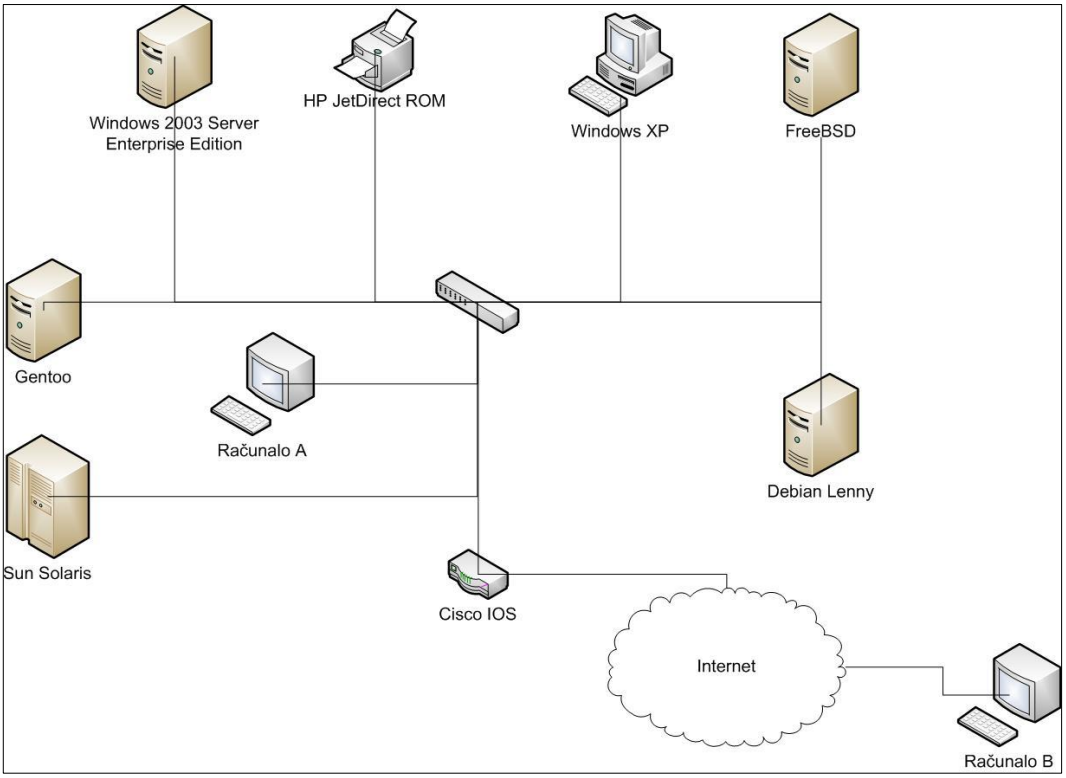
```
mICMP_ADDRMASK = 17;
IP_ID = 0xBABA;
ICMP_ID = rand() % 65536;

ip_packet =
  forge_ip_packet(ip_id  : IP_ID,
                  ip_p   : IPPROTO_ICMP,
                  ip_src : this_host());
icmp_packet =
  forge_icmp_packet(icmp_type : ICMP_ADDRMASK,
                    icmp_id   : ICMP_ID,
                    data      : crap(length:4, data:raw_string(0)),
                    ip        : ip_packet);
```

Slika 6.2. Slaganje ICMP Address Mask paketa

6.1.3 Rezultati ispitivanja

Ispitivanje ostvarene programske skripte izvedeno je nad 8 različitih operacijskih sustava i to s dva različita računala. Prvo ispitivanje izvedeno je s računala A koje se nalazilo u istoj lokalnoj mreži kao i ciljni sustavi, dok je drugo izvedeno s računala B koje se nalazilo izvan lokalne mreže. Konfiguracija mreže ilustrirana je na slici 6.3. Ispitana računala pokretana su sljedećim operacijskim sustavima:

- *Cisco IOS*
- *HP JetDirect ROM*
- *Windows 2003 Server Enterprise Edition*
- *Windows XP*
- *Sun Solaris*
- *FreeBSD*
- *Debian Lenny*
- *Gentoo*



Slika 6.3. Ispitno okruženje

Rezultati dobiveni korištenjem ostvarene programske skripte uspoređeni su s rezultatima dobivenim korištenjem slobodno raspoloživih programa *xprobe* i *nmap*. Radi se o alatima koji sadrže mogućnost detekcije operacijskog sustava udaljenog računala. Program *xprobe* je originalna implementacija X logike, a izradili su ga upravo Fyodor Yarochkin i Ofir Arkin. Alat *nmap* je popularan mrežni alat koji za detekciju operacijskog sustava koristi analizu TCP paketa. Oba alta pokrenuta su s istih računala kao i NASL skripta. Rezultati ispitivanja prikazani su u tablicama 6.1 i 6.2. Dobiveni točni rezultati označeni su oznakom "+", a analogno, ukoliko su rezultati bili netočni ili program nije mogao pronaći operacijski sustav, označeni su oznakom "–".

Treba uzeti u obzir da su rezultati ovise o trenutnoj situaciji na mreži i nisu nužno reproducibilni. Primjerice, ukoliko zbog opterećenja neki paket ne dođe do programa za detekciju, to će svakako utjecati na kvalitetu njegovih rezultata, ali to su faktori na koje se ne može utjecati. Zbog realnije slike koja se dobije ispitivanjem alata na pravoj računalnoj mreži nisu korišteni laboratorijski uvjeti za ispitivanje.

Tablica 6.1. Rezultati ispitivanja s računala u lokalnoj mreži

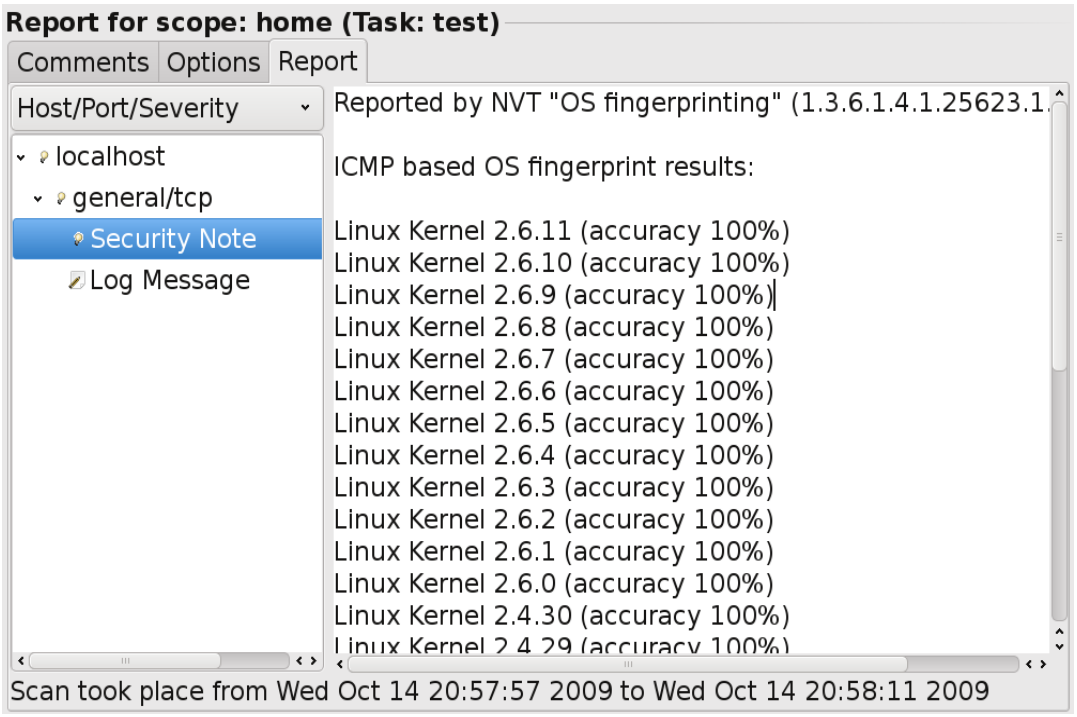
OS	Računalo A		
	nmap	xprobe	os_fingerprint.nasl
Cisco IOS	+	+	+
HP JetDirect ROM	+	+	+
Microsoft Windows 2003 Server Enterprise Edition	+	-	+
Windows XP	+	+	+
Sun Solaris	+	+	+
FreeBSD	+	+	+
Debian Lenny	+	+	+
Gentoo	+	+	+
Ukupno	**8**	**7**	**8**

Rezultati dobiveni ispitivanjem s računala koje se nalazi u lokalnoj mreži s ispitivanim poslužiteljima pokazali su odlične rezultate za NASL skriptu. Valja napomenuti kako se u lokalnoj mreži nisu nalazili vatrozidovi niti drugi mrežni uređaji koji filtriraju promet što je svakako utjecalo na kvalitetu rezultata. I druga dva ispitana alata (*nmap* i *xprobe*) postigli su dobre rezultate. Budući da se radi o cijenjenim mrežnim alatima, rezultati dobiveni NASL skriptom još više dobivaju na vrijednosti.

Tablica 6.2. Rezultati ispitivanja s računala izvan lokalne mreže

OS	Računalo B		
	nmap	xprobe	os_fingerprint.nasl
Cisco IOS	+	+	-
HP JetDirect ROM	+	+	-
Microsoft Windows 2003 Server Enterprise Edition	+	-	-
Windows XP	+	+	+
Sun Solaris	+	-	-
FreeBSD	+	-	+
Debian Lenny	+	+	+
Gentoo	-	+	+
Ukupno	**7**	**5**	**4**

Ispitivanje s drugog računala, koje se nalazilo na Internetu (izvan lokalne mreže), pokazalo je značajno lošije rezultate NASL skripte. Razloge tome može se tražiti u činjenici da su paketi putem prolazili kroz nekoliko mrežnih uređaja (rutera, vatrozida i dr.) koji često mijenjaju pojedina polja u zaglavljima IP paketa. Može se zaključiti kako je detekcija operacijskog sustava pomoću ICMP paketa puno pouzdanija kada se koristi unutar lokalne računalne mreže. Primjer korištenja ove skripte u sklopu alata OpenVAS prikazan je na slici 6.4.



Slika 6.4. Rezultati skripte os_fingerprint.nasl

6.2 Detekcija servisa rsync

Programski alat *rsync* koristi se za sinkronizaciju datoteka i direktorija na različitim računalima. Posebno je popularan jer u svom radu koristi posebne algoritme kako bi minimizirao količinu podataka koji se razmjenjuju među računalima i to na način da prenosi samo izmjene koje su se dogodile od zadnje sinkronizacije. Za prijenos podataka ovim programom može se koristiti tuneliranje SSH protokolom ili protokol koji *rsync* koristi u poslužiteljskom načinu rada.

Programska skripta `rsync_modules.nasl` koristi se protokolom koji je implementiran u *rsync* poslužitelj kako bi detektirala njegovu prisutnost na sustavu te dohvatila listu modula definiranih u datoteci `rsyncd.conf`. Pod modulima se misli na direktorije čiji je sadržaj putem *rsync* usluge dostupan na mreži. Primjer sadržaja datoteke `rsyncd.conf` prikazan je na slici 6.5. U ovoj datoteci moguće je definirati neke globalne opcije za *rsync* poslužitelj, kao i pravila koja važe za pojedine module.

```
# GLOBAL OPTIONS

motd file=/etc/motd
log file=/var/log/rsyncd
pid file=/var/run/rsyncd.pid

# MODULE OPTIONS

[ftp]
      comment = public archive
      path = /var/www/pub
      use chroot = yes
      lock file = /var/lock/rsyncd
      read only = yes
      list = yes
      uid = nobody
      gid = nogroup
#     secrets file = /etc/rsyncd.secrets
      strict modes = yes
      ignore errors = no
      ignore nonreadable = yes
      transfer logging = no
      timeout = 600
      refuse options = checksum dry-run
```

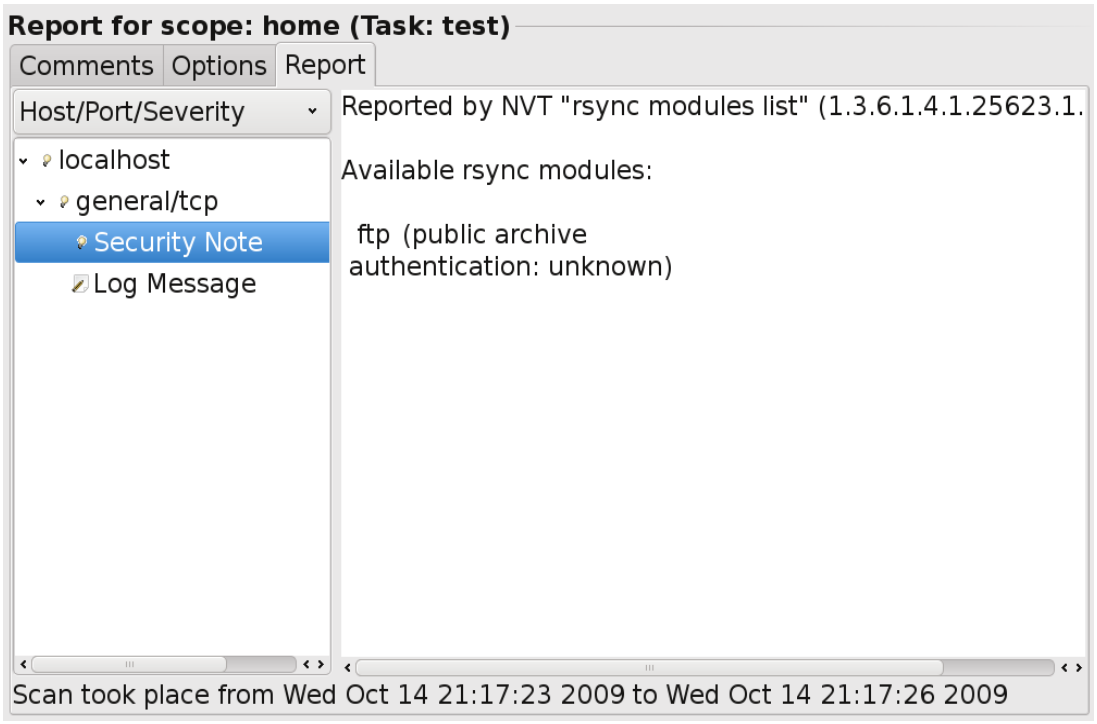
Slika 6.5. Sadržaj datoteke rsyncd.conf

Skripta `rsync_modules.nasl` koristi protokol prikazan na slici 6.6. za spajanje na *rsync* poslužitelj i dohvaćanje liste dostupnih modula. Protokol je opisan u datoteci `csprotocol.txt` koja dolazi s izvornim kodom programa *rsync*.

```
# <version> - inacica protokola definirana u datoteci rsync.h
# <subprotocol> - podinacica protokola
# <MOTD> - Message Of The Day - pozdravna poruka
# AUTHREQD - potrebna je autentikacija
# OK – nije potrebna autentikacija

# Protokol:

@RSYNCD:    <version>.<subprotocol>
@CLIENT:    <version>.<subprotocol>
@RSYNCD:    <MOTD>
@CLIENT:    #list
@RSYNCD:    <lista_modula>
@CLIENT:    <ime_modula>
@RSYNCD:    AUTHREQD <challenge> | OK
```

Slika 6.6. Protokol rsync poslužitelja

Za izradu skripte nisu korištene vanjske biblioteke, već isključivo ugrađene funkcije za komunikaciju TCP protokolom. Rezultati koje ova skripta dohvaća dojavljuju se korisniku u obliku ranjivosti niske razine rizika. Rezultati se zapisuju i u bazu znanja kako bi ih drugi sigurnosni testovi mogli koristiti. Rezultati korištenja skripte prikazani su na slici 6.7.



Slika 6.7. Rezultati skripte rsync_modules.nasl

6.3 Detekcija *web* aplikacije WebAPP

WebAPP (eng. *Web Automated Perl Portal System*) je popularna CMS (eng. *Content Managment System*) aplikacija koja se najčešće koristi za vođenje internetskih zajednica (eng. *community*), a napisana je u programskom jeziku Perl. Kako bi se jedna ovakva aplikacija uspješno detektirala potrebno je pronaći zajedničke karakteristike svih stranica koje ju koriste. Budući da se radi o *web* stranici, karakteristike je najlakše pronaći unutar HTML (eng. *Hypertext Markup Language*) koda generiranog aplikacijom.

Upravo uzorci (eng. *pattern*) pronađeni unutar HTML koda iskorišteni su u skripti `webapp_detect.nasl` za uspješnu detekciju prisutnosti i inačice aplikacije na poslužitelju. Pronađeni uzorci prikazani su na slici 6.8.

```
# uzorak 1
<meta name="Generator" content="WebAPP - Automated Perl Portal \ 0.9.9.9
RC4">

# uzorak 2
<meta name="Generator" content="WebAPP, Web Automated Perl Portal. \ WebApp
1.0 Build 3">

# uzorak 3
<meta name="Generator" content="WebAPP 0.9.9.8">

# uzorak 4
This site was made with  <a href="http://www.web-app.org/cgi- \
bin/index.cgi" class="webapplink">WebAPP - Automated Perl Portal \
</a> <a href=http://www.sitespot.biz/?action=ver \
class="webapplink">v0.9.9.9 RC4</a>

# uzorak 5
This site was made with <a href="#VERSION" \
onClick="version=dhtmlmodal.open('http://www.web-app.net/cgi- \
bin/index.cgi?action=ver','iframe','http://www.web-app.net/cgi \
-bin/index.cgi?action=ver', 'WebAPP Version', \
'width=600px,height=300px,left=120px,top=105px,resize=1,scrolling= \
1 center=1,recal'); return false"><b>WebAPP, Web Automated Perl \ Portal.
v2.0 build 001</b></a>

# uzorak 6
This site was made with  <a href="http://www.web-app.org/cgi \
-bin/index.cgi" class="webapplink">WebAPP</a> <a \
href=http://www.demoapp.org/cgi-bin/index.cgi?action=ver \
class="webapplink">v0.9.9.8</a>,<br> a web portal system written \
in Perl.<br>
```

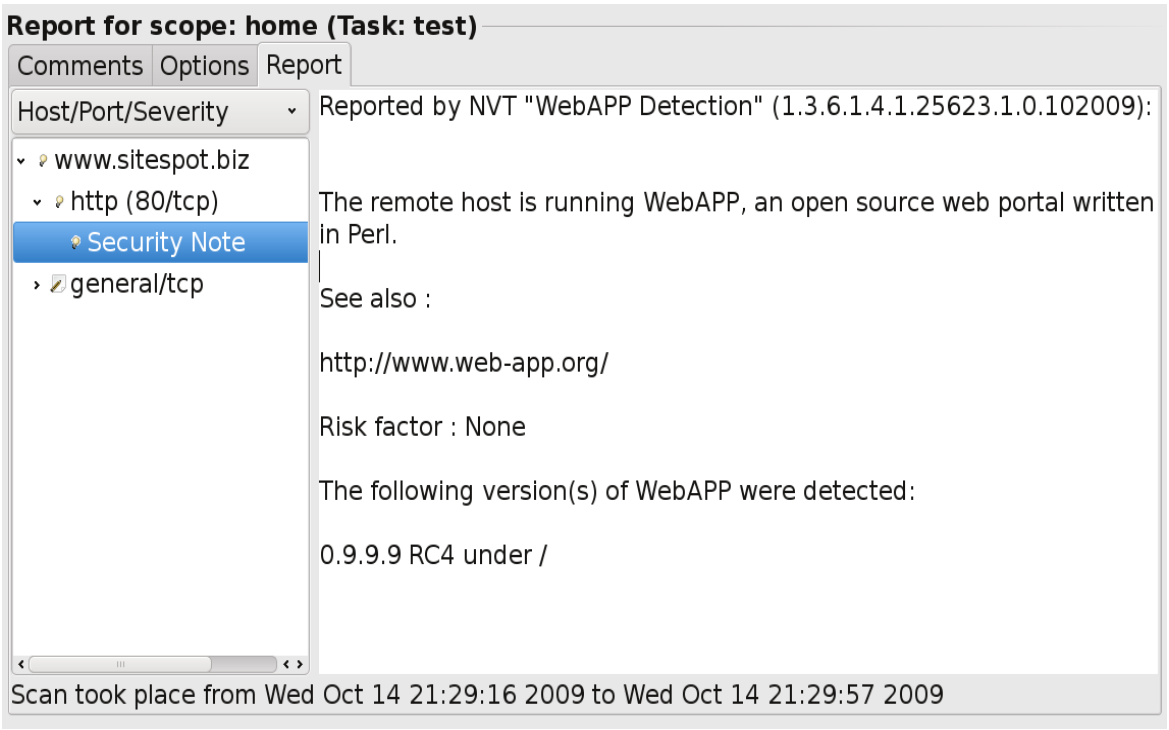
Slika 6.8. Uzorci iz HTML koda

Lako je uočiti kako se prva tri uzorka, baš poput druga tri, malo razlikuju. Kako bi se obuhvatili svi pronađeni uzorci, u skripti se koriste regularni izrazi prikazani na slici 6.9. Budući da se prva tri uzorka manje međusobno razlikuju od druga tri, skripta se za potrebe detekcije inačice prvo oslanja na prvi regularni izraz, dok se drugi koristi samo u slučaju neuspjeha. Za izradu regularnih izraza korišteni su standardni simboli za Linux/UNIX programe egrep i grep.

```
# regularni izraz 1
<meta name=.Generator. content=.WebAPP[^0-9]*([^>"]*)

# regularni izraz 2
This site was made with[^>]*>WebAPP([^>]*>)*[^>]*>v([0-9.]*)
```

Slika 6.9. Regularni izrazi

Skripta, baš poput skripte za detekciju servisa *rsync*, u slučaju korištenja s programom OpenVAS prijavljuje korisniku detektirane inačice aplikacije WebAPP te unosi odgovarajuće informacije u bazu znanja. Primjer korištenja prikazan je na slici 6.10.



Slika 6.10. Rezultati skripte webapp_detect.nasl

7. Zaključak

U vremenu kada su sigurnosni napadi i provale u računalne sustave svakodnevica, održavanje sigurnosti je delikatan zadatak. Povećanjem korisničkih potreba informacijski sustavi postaju sve složeniji, a sigurnost je često zadnja stvar na koju dizajneri takvih sustava misle. Iako se neki pomaci vide, pogotovo u razvitku modernih operacijskih sustava, činjenica da je čovjek najslabija karika u lancu sigurnosti nekog sustava teško će se promijeniti. Ipak, postoje razni oblici zaštite koji se mogu upotrijebiti u borbi protiv računalnog kriminala. Redovita provjera ranjivosti računalnih sustava svakako je jedan od važnih koraka u održavanju sigurnosti svakog računalnog sustava.

Na tržištu postoje brojni automatizirani alati za provjeru ranjivosti računalnih sustava, no većina njih su komercijalni. Čak i nekad popularan i slobodno raspoloživ alat Nessus, izgubio je dio svoje publike prelaskom u komercijalne alate. Programski alat OpenVAS, koji je na neki način naslijedio Nessus na području slobodno raspoloživih alata za provjeru ranjivosti, preuzeo je od Nessus-a brojne dobre osobine poput modularnosti, klijent-poslužitelj arhitekture i sl. No ipak, i sam se dalje razvijao u dobrom smjeru pa danas predstavlja jedan od najkompletnijih i najpouzdanijih alata na tržištu.

Iako danas postoje brojni skriptni jezici u kojima bi se mogli pisati sigurnosni testovi, iskustvo je pokazalo kako je za njihovu izradu najbolje koristiti specijalizirane jezike, poput NASL-a. Iako je ovaj jezik u mnogim područjima ograničen u usporedbi s drugim skriptnim jezicima, pisanje sigurnosnih testova čini vrlo jednostavnim zadatkom.

U praktičnom dijelu ovog diplomskog rada razvijene su programske skripte za detekciju operacijskog sustava udaljenog računala kao i nekih uobičajenih servisa. Skripta za detekciju operacijskog sustava, koja implementira tzv. X logiku, u usporedbi s drugim alatima za istu namjenu pokazala je odlične rezultate prilikom ispitivanja unutar lokalne računalne mreže. Iako su rezultati kod ispitivanja iz vanjske mreže bili nešto lošiji, skripta se lako može nadograditi novim modulima koji mogu koristiti i protokole aplikacijske razine, što bi svakako značajno poboljšalo rezultate detekcije.

8. Literatura

[1] Kimberly Graves: *Official Certified Ethical Hacker Review Guide*, Wiley Publishing, 2007.

[2] Eric Cole, Ronald Krutz, James Conley: *Network Security Bible*, Wiley Publishing, 2005.

[3] *Ethical Hacking (EC-Council Exam 312-50): Student Courseware*, OSB, 2004.

[4] Jan-Oliver Wagner, Michael Wiegand, Tim Brown, Carsten Koch Mauthe, Geoff Galitz: *OpenVAS Compendium*, 2009.

[5] Michel Arboi: *The NASL2 reference manual*, 2005.

[6] Renaud Deraison: *The Nessus Attack Scripting Language Reference Guide*, 2005.

[7] Ofir Arkin, Fyodor Yarochkin: X Remote ICMP based OS fingerprinting techniques, 2001.