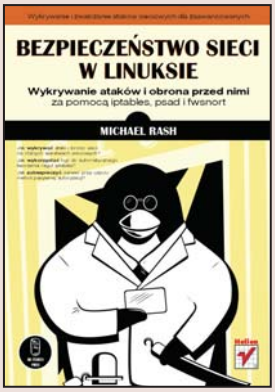
Bezpieczeństwo sieci w Linuksie. Wykrywanie ataków i obrona przed nimi za pomocą iptables, psad i fwsnort



Autor: Michael Rash
Tłumaczenie: Andrzej Stefański
ISBN: 978-83-246-1539-1
Tytuł oryginału: Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort
Format: 170x230, stron: 352
Przykłady na ftp: 1038 kB

Bezpieczeństwo sieci w Linuksie. Wykrywanie ataków i obrona przed nimi za pomocą iptables, psad i fwsnort

Wykrywanie i zwalczanie ataków sieciowych dla zaawansowanych

- Jak wykrywać ataki i bronić sieci na różnych warstwach sieciowych?
- Jak wykorzystać logi do automatycznego tworzenia reguł iptables?
- Jak zabezpieczyć serwer przy użyciu metod pasywnej autoryzacji?

Każdy administrator sieci wie, jak ważna jest jej ochrona przed atakami z zewnątrz. Wie również, że jest to nieustająca walka z coraz bardziej zaawansowanymi technikami.

Raz postawiony firewall nie jest w stanie zapewnić sieci całkowitego bezpieczeństwa w nieskończoność, dlatego ciągłe poszerzanie swojej wiedzy i umiejętne jej zastosowanie jest nieodłączną częścią pracy administratorów. Jeśli jesteś odpowiedzialny za utrzymanie bezpieczeństwa sieci, książka ta jest dla Ciebie. Stanowi źródło wiedzy z zakresu wykorzystania oprogramowania open source i systemu Linux w walce o bezpieczeństwo sieci.

Michael Rash, autor wielu publikacji i książek, jest ekspertem w dziedzinie ochrony sieci. W książce Bezpieczeństwo sieci w Linuksie. Wykrywanie ataków i obrona przed nimi za pomocą iptables, psad i fwsnort przedstawia sposób łączenia metod zabezpieczeń w systemie Linux przy użyciu iptables, Snort oraz psad, fwsnort i fwknop, których sam jest twórcą. Na praktycznych przykładach udowadnia skuteczność zastosowanych technologii, które są porównywalne z komercyjnymi, i pokazuje sposoby ich wykorzystywania. Dołączone do książki skrypty umożliwiają przetestowanie omawianych technologii w praktyce.

- Konfiguracja iptables
- Atak i obrona w warstwach modelu OSI
- Instalacja i konfiguracja psad
- Integracja psad z oprogramowaniem zewnętrznym
- Konfiguracja systemu wykrywania włamań Snort
- Przetwarzanie sygnatur Snort na reguły iptables
- Automatyzacja przetwarzania sygnatur poprzez fwsnort
- Analiza i raporty psad
- Instalacja i konfiguracja fwknop
- Wizualizacja logów iptables

Poznaj niekomercyjne metody skutecznej ochrony sieci!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

PODZIĘKOWANIA 13

PRZEDMOWA 15

WPROWADZENIE 19
Dlaczego wykrywać ataki przy pomocy iptables? 20
Co z dedykowanymi sieciowymi systemami wykrywania włamań? 21
Głęboka obrona 22
Wymagania 22
Literatura 23
Strona internetowa 24
Podsumowanie rozdziałów 24

1
KONFIGUROWANIE IPTABLES 27
iptables 27
Filtrowanie pakietów z iptables 28
Tabele 29
Łańcuchy 29
Dopasowania 30
Cele 30
Instalacja iptables 31
Konfiguracja jądra 32
Najważniejsze opcje kompilacji Netfilter 33
Kończenie konfiguracji jądra 35
Ładowalne moduły jądra kontra opcje wkompilowane i bezpieczeństwo 35
Bezpieczeństwo i minimalna kompilacja 36
Kompilacja i instalacja jądra 37
Instalacja binariów iptables działających w przestrzeni użytkownika 38
Domyślna polityka bezpieczeństwa iptables 39

Wymagania polityki bezpieczeństwa ... 39
Początek skryptu iptables.sh ... 41
Łańcuch INPUT ... 42
Łańcuch OUTPUT ... 44
Łańcuch FORWARD ... 45
Translacja adresów sieciowych (NAT) ... 46
Uaktywnianie polityki bezpieczeństwa ... 47
Programy iptables-save i iptables-restore ... 47
Testowanie polityki bezpieczeństwa: TCP ... 50
Testowanie polityki bezpieczeństwa: UDP ... 52
Testowanie polityki bezpieczeństwa: ICMP ... 53
Podsumowanie ... 54

2
ATAK I OBRONA W WARSTWIE SIECIOWEJ ... *55*
Logowanie nagłówków warstwy sieci w iptables ... 56
Logowanie nagłówka IP ... 56
Definicje ataków na warstwę sieci ... 59
Nadużycia w warstwie sieci ... 60
ICMP Ping z Nmap ... 60
Fałszowanie pakietów IP (IP spoofing) ... 61
Fragmentacja pakietów IP ... 63
Niskie wartości TTL ... 63
Atak smerfów (the Smurf Attack) ... 65
Ataki DDoS ... 65
Ataki IGMP jądra Linuksa ... 66
Odpowiedzi w warstwie sieci ... 66
Filtrowanie w warstwie sieci ... 67
Odpowiedź wyzwalana w warstwie sieci ... 67
Łączenie odpowiedzi w różnych warstwach ... 68

3
OBRONA I ATAK W WARSTWIE TRANSPORTOWEJ ... 71
Logowanie nagłówków warstwy transportowej za pomocą iptables ... 72
Logowanie nagłówka TCP ... 72
Logowanie nagłówka UDP ... 74
Definicje ataków na warstwę transportową ... 75
Nadużycia w warstwie transportowej ... 75
Skanowanie portów ... 76
Przemiatanie portów ... 84
Ataki przewidujące sekwencję TCP ... 85
SYN flood ... 85
Obrona w warstwie transportowej ... 86
Obrona protokołu TCP ... 86
Obrona protokołu UDP ... 90
Reguły firewalla i listy kontrolne routera ... 91

4
ATAK I OBRONA W WARSTWIE APLIKACJI 93
Dopasowanie ciągów znaków w warstwie aplikacji przy użyciu iptables 94
Obserwowanie rozszerzenia porównującego ciągi znaków w działaniu 94
Dopasowywanie znaków niedrukowalnych w warstwie aplikacji 96
Definicje ataków w warstwie aplikacji 97
Nadużycia w warstwie aplikacji 98
Sygnatury Snort 98
Wykorzystanie przepełnienia bufora 99
Ataki typu SQL injection 101
Czynnik ludzki 102
Szyfrowanie i kodowanie danych w aplikacji 105
Obrona w warstwie aplikacji 106

5
WPROWADZENIE DO PSAD 107
Historia 107
Po co analizować logi firewalla? 108
Możliwości psad 109
Instalacja psad 109
Administracja psad 111
Uruchamianie i zatrzymywanie psad 112
Unikalność procesu usługi 112
Konfiguracja polityki bezpieczeństwa iptables 112
Konfiguracja syslog 113
Klient usługi whois 116
Konfiguracja psad 117
/etc/psad/psad.conf 117
/etc/psad/auto_dl 123
/etc/psad/signatures 124
/etc/psad/snort_rule_dl 124
/etc/psad/ip_options 125
/etc/psad/pf.os 125
Podsumowanie 126

6
DZIAŁANIE PSAD: WYKRYWANIE PODEJRZANYCH DANYCH 127
Wykrywanie skanowania portów przy pomocy psad 128
Skanowanie TCP connect() 129
Skanowanie TCP SYN 132
Skanowanie TCP FIN, XMAS i NULL 134
Skanowanie UDP 135
Ostrzeganie i raportowanie przy pomocy psad 137
Ostrzeżenia psad wysyłane przez email 137
Raporty generowane przez psad do sysloga 140
Podsumowanie 141

7
ZAAWANSOWANE ZAGADNIENIA PSAD: OD PORÓWNYWANIA SYGNATUR DO WYKRYWANIA SYSTEMÓW OPERACYJNYCH 143
Wykrywanie ataku z wykorzystaniem reguł Snort 144
Wykrywanie skanera portów ipEye 145
Wykrywanie ataku LAND 146
Wykrywanie ruchu na 0 porcie TCP 147
Wykrywanie pakietów z zerową wartością TTL 147
Wykrywanie ataku Naptha DoS 148
Wykrywanie prób rutowania źródła 148
Wykrywanie spamu komunikatora Windows Messenger 149
Uaktualnienia sygnatur psad 150
Wykrywanie systemów operacyjnych 151
Aktywne wykrywanie systemów operacyjnych za pomocą Nmap 151
Pasywne wykrywanie systemu operacyjnego z p0f 152
Raportowanie DShield 154
Format raportów DShield 155
Przykładowy raport DShield 155
Przeglądanie danych o statusie psad 156
Analizowanie archiwalnych logów 159
Tryb informacyjny/śledzenia 160
Podsumowanie 161

8
AUTOMATYCZNA OBRONA ZA POMOCĄ PSAD 163
Zapobieganie włamaniom a aktywna obrona 163
Minusy aktywnej obrony 165
Rodzaje ataków 165
Fałszywe alarmy 166
Reagowanie za pomocą psad na atak 166
Opcje 167
Zmienne konfiguracyjne 168
Przykłady automatycznej obrony 170
Konfiguracja automatycznej obrony 170
Reakcja na skanowanie typu SYN 171
Reakcja na skanowanie UDP 173
Sprawdzanie wersji oprogramowania za pomocą Nmap 174
Reakcja na skanowanie typu FIN 174
Złośliwe fałszowanie skanowania 175
Integrowanie automatycznych odpowiedzi psad z innymi narzędziami 176
Interfejs wiersza poleceń 176
Integracja ze Swatch 178
Integracja z własnymi skryptami 179
Podsumowanie 181

9
TŁUMACZENIE REGUŁ SNORT NA REGUŁY IPTABLES 183
Dlaczego warto używać fwsnort? 184
Dogłębna obrona 185
Wykrywanie włamań zorientowane na cel i defragmentacja warstwy sieci 185
Małe wymagania 186
Bezpośrednie odpowiedzi 186
Przykłady przetłumaczonych sygnatur 187
Sygnatura Nmap command attempt 187
Sygnatura Bancos Trojan z Bleeding Snort 188
Sygnatura próby połączenia PGPNet 189
Interpretacja reguł Snort przez fwsnort 190
Tłumaczenie nagłówka reguły Snort 190
Tłumaczenie opcji reguł Snort: logowanie pakietów w iptables 192
Opcje Snort i filtrowanie pakietów w iptables 195
Niewspierane opcje reguł Snort 207
Podsumowanie 209

10
URUCHOMIENIE FWSNORT 211
Instalacja fwsnort 211
Uruchomienie fwsnort 214
Plik konfiguracyjny fwsnort 215
Budowa skryptu fwsnort.sh 217
Parametry fwsnort 221
Obserwowanie fwsnort w działaniu 222
Wykrywanie Trin00 223
Wykrywanie ruchu sieciowego związanego z wykonywaniem kodów powłoki systemu Linux 224
Wykrywanie i obrona przed trojanem Dumador 225
Wykrywanie i reagowanie na atak fałszujący pamięć podręczną DNS 227
Tworzenie białych i czarnych list 230
Podsumowanie 231

11
POŁĄCZENIE PSAD I FWSNORT 233
Łączenie wykrywania fwsnort i działań psad 234
Atak na Setup.php z WEB-PHP 234
Aktywna obrona 238
psad kontra fwsnort 238
Ograniczanie działań psad wobec ataków wykrytych przez fwsnort 239
Łączenie działań fwsnort i psad 240
Cele DROP i REJECT 242

Uniemożliwianie uaktualnień Metasploit 245
Uaktualnienia Metasploit 245
Tworzenie sygnatury 248
Zatrzymywanie uaktualnień Metasploit za pomocą fwsnort i psad 249
Podsumowanie 253

12
PORT KNOCKING A AUTORYZACJA POJEDYNCZYM PAKIETEM 255
Redukcja możliwości ataku 256
Problem ataków 0-Day 256
Odkrywanie ataków typu 0-Day 257
Wpływ na systemy wykrywania włamań wykorzystujące sygnatury 258
Dogłębna obrona 259
Wywołania portów (port knocking) 259
Uniemożliwienie wyszukiwania celów za pomocą Nmap 261
Jawne sekwencje wywołań portów 261
Zaszyfrowane sekwencje wywołań portów 263
Ograniczenia architektury techniki wywołań portów 266
Autoryzacja pojedynczym pakietem 269
Przekraczanie ograniczeń wywołań portów 270
Ograniczenia architektoniczne SPA 271
Zabezpieczanie przez utajnianie? 272
Podsumowanie 274

13
WPROWADZENIE FWKNOP 275
Instalacja fwknop 276
Konfiguracja fwknop 278
/etc/fwknop/fwknop.conf 278
/etc/fwknop/access.conf 282
Przykładowy plik /etc/fwknop/access.conf 285
Format pakietu SPA fwknop 286
Uruchomienie fwknop 289
SPA i szyfrowanie symetryczne 289
SPA i szyfrowanie asymetryczne 292
Wykrywanie i powstrzymywanie ataku odtwarzającego pakiety 296
Fałszowanie adresu źródłowego pakietu SPA 298
Integracja OpenSSH z fwknop 299
SPA przez Tor 300
Podsumowanie 302

14
WIZUALIZACJA LOGÓW IPTABLES 303
Dostrzec nietypowe 304
Gnuplot 306
Rysowanie wykresów w Gnuplot 307
Łączenie psad i Gnuplot 308
AfterGlow 309
Wizualizacje ataków iptables 310
Skanowania portów 310
Przemiatanie portów 314
Robak Slammer 318
Robak Nachi 319
Połączenia wychodzące z zaatakowanych systemów 321
Podsumowanie 323

A
FAŁSZOWANIE ATAKU 325

B
SKRYPT FWSNORT 331

SKOROWIDZ 337

4

Atak i obrona w warstwie aplikacji

Warstwa aplikacji — siódma warstwa modelu OSI — to dla niej stworzone są niższe warstwy. Gwałtowny wzrost Internetu jest możliwy dzięki niższym warstwom, ale to aplikacje są główną siłą napędową. Istnieją tysiące aplikacji korzystających z Internetu, zaprojektowanych w celu uproszczenia złożonych zadań i rozwiązania problemów różnych grup ludzi — od zwykłych konsumentów przez agencje rządowe do wielkich międzynarodowych korporacji. Najważniejszym zagadnieniem we wszystkich tych aplikacjach jest bezpieczeństwo i jak dotąd, jeśli sądzić na podstawie częstotliwości ogłaszania słabości w miejscach takich jak Bugtraq, nie jest z tym najlepiej.

Jeśli chodzi o włamania do systemu, najwięcej dzieje się w warstwie aplikacji. Najbardziej wartościowe cele jak interfejsy do banków internetowych i wrażliwych danych medycznych działają (lub są dostępne) w warstwie aplikacji i wśród zagrożeń widać trend dokonywania włamań dla zysków finansowych. Prywatność użytkowników jest w tej sytuacji naruszana przy okazji. Gdyby wymagania bezpieczeństwa były traktowane z większym priorytetem we wszystkich fazach życia aplikacji — przy projektowaniu, tworzeniu, wdrażaniu i utrzymywaniu — bylibyśmy w lepszej sytuacji.

Dopasowanie ciągów znaków w warstwie aplikacji przy użyciu iptables

Jedną z najważniejszych funkcjonalności każdego IDS jest możliwość wyszukiwania podejrzanych fragmentów złośliwego kodu w danych warstwy aplikacji. Jednak z powodu mniej restrykcyjnych wymagań co do struktury protokołów warstwy aplikacji niż w przypadku warstwy sieci czy warstwy transportowej systemy wykrywania włamań muszą być bardziej elastyczne przy przeszukiwaniu danych warstwy aplikacji.

Na przykład jeśli IDS przyjmuje, że pewna sekwencja bajtów jest nieszkodliwa (i może być zignorowana), zmiany w protokole warstwy aplikacji mogą sprawić, że to założenie będzie fałszywe, i spowodować przeoczenie przez IDS ataków przeprowadzanych w nieoczekiwany sposób. Słabości poszczególnych implementacji protokołów warstwy aplikacji mogą być wykorzystane za pomocą manipulacji w omijanych przez IDS sekcjach protokołu.

W tej sytuacji potrzebny jest elastyczny mechanizm nadzorowania danych warstwy aplikacji. Możliwość porównywania ciągów znaków ze wszystkimi danymi warstwy aplikacji w całym ruchu sieciowym jest pierwszym krokiem w dobrym kierunku i jest dostarczana przez specjalne rozszerzenie iptables (string match extension).

UWAGA *Z tego powodu podkreśliłem konieczność włączenia wsparcia porównywania ciągów znaków w „Kończenie konfiguracji jądra" na stronie 35. Porównywanie ciągów znaków będzie również intensywnie wykorzystywane w rozdziałach 9., 10. i 11. przy omawianiu fwsnort.*

Rozszerzenie umożliwiające porównywanie znaków w iptables wykorzystuje szybki algorytm przeszukiwania ciągów znaków *Boyera-Moore'a* (*http://www.cs.utexas.edu/users/moore/best-ideas/string-searching*). Algorytm ten jest często wykorzystywany w systemach wykrywania włamań, również w najlepszym otwartym IDS — Snort (*http://www.snort.org*) — ze względu na swoją zdolność szybkiego porównywania ciągów znaków z przepływającymi danymi.

UWAGA *Porównywanie ciągów znaków pojawiło się w iptables od czasów jądra 2.4, ale zmiana architektury wpływająca na sposób przechowywania struktury danych opisującej pakiet w pamięci jądra (struktury* `sk_buff` *zyskały możliwość zapisywania w pamięci nieciągłej) uniemożliwiła działanie tego rozszerzenia w jądrach od 2.6.0 do 2.6.13.5. Rozszerzenie porównujące ciągi znaków zostało przepisane dla jądra 2.6.14 i od tego momentu zostało włączone do jądra.*

Obserwowanie rozszerzenia porównującego ciągi znaków w działaniu

By przetestować rozszerzenie porównujące znaki w iptables, stworzymy prostą regułę sprawdzającą, czy działa ono tak, jak powinno. Poniższa reguła wykorzystuje cel `LOG` iptables do wygenerowania informacji w logach, gdy ciąg znaków

tester zostanie wysłany do serwera Netcat nasłuchującego na porcie 5001 TCP. (Niezbędna jest też reguła ACCEPT, by domyślna polityka bezpieczeństwa z rozdziału 1. umożliwiła ustanowienie połączenia TCP z zewnętrznego źródła).

```
[iptablesfw]# iptables -I INPUT 1 -p tcp --dport 5001 -m string --string
➥"tester" ❶ --algo bm -m state --state
➥❷ ESTABLISHED -j LOG --log-prefix "tester"
[iptablesfw]# iptables -I INPUT 2 -p tcp --dport 5001 -j ACCEPT
```

Warto zauważyć w ❶ parametr iptables `--algo bm`. Rozszerzenie porównujące ciągi znaków jest zbudowane przy wykorzystaniu mechanizmów przeszukiwania tekstów z jądra Linux (znajdujących się w katalogu `linux/lib` w źródłach jądra). Wspiera ono wiele różnych algorytmów, łącznie z **algorytmem przeszukiwania ciągów znaków *Boyera-Moore'a*** (stąd powyższe *bm*) i algorytmem *Knutha-Morrisa-Pratta* (*kmp*)[1].

Argumenty linii poleceń `--m state --state ESTABLISHED` w ❷ ograniczają zakres operacji porównywania znaków do pakietów będących częścią ustalonego połączenia TCP, co oznacza, że nie jest możliwe skłonienie iptables do porównania ciągu znaków z danymi zawartymi w pakiecie ze sfałszowanym adresem źródłowym — musi być ustanowione dwukierunkowe połączenie.

Do ustawienia serwera TCP nasłuchującego na porcie 5001 użyjemy *Netcat*, a następnie w systemie *ext_scanner*, wykorzystując ten sam program, wyślemy ciąg znaków `tester` do serwera:

```
[iptablesfw]$ nc -l -p 5001
[ext_scanner]$ echo "tester" | nc 71.157.X.X 5001
```

Następnie sprawdzimy, że w logach systemowych znajduje się zapis wygenerowany przez regułę `LOG` iptables:

```
[iptablesfw]# tail /var/log/messages | grep tester
Jul 11 04:19:14 iptablesfw kernel: tester IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X
➥DST=71.157.X.X
LEN=59 TOS=0x00 PREC=0x00 TTL=64 ID=41843 DF PROTO=TCP SPT=55363 DPT=5001
WINDOW=92 RES=0x00 ACK PSH URGP=0
```

Warto powyżej zauważyć pogrubiony prefiks `tester`. W pozostałej części zapisanego komunikatu można znaleźć potwierdzenie, że opisywany pakiet został wysłany z systemu *ext_scanner* do serwera Netcat nasłuchującego na porcie 5001.

[1] Algorytm wyszukiwania ciągów znaków Boyera-Moore'a w większości zastosowań jest bardziej wydajny niż algorytm Knutha-Morrisa-Pratta. Złożoność algorytmu BM to O (*n*/*m*), podczas gdy złożoność KMP to O (*n*), gdzie *n* oznacza długość przeszukiwanego tekstu, a *m* to długość szukanego ciągu znaków. Pod adresem *http://people.netfilter.org/pablo/textsearch* znajdują się wykresy wydajności.

UWAGA *Ten sam rezultat można osiągnąć, używając programu telnet (uruchomionego w trybie liniowym) w roli klienta zamiast programu Netcat, tak by cały ciąg znaków* `tester` *był zawarty w jednym pakiecie. Działa to całkiem dobrze, ale telnet ma poważne ograniczenia: nie może kontaktować się z serwerami UDP i trudno jest za jego pomocą wygenerować dowolny znak niedrukowalny.*

Dopasowywanie znaków niedrukowalnych w warstwie aplikacji

Działając jako klient, Netcat może komunikować się z serwerami UDP tak łatwo jak z tymi wykorzystującymi gniazda TCP. Po połączeniu z *Perlem* Netcat może wysyłać do sieci dowolne bajty, również takie, które nie są być reprezentowane jako drukowalne znaki ASCII. Jest to ważne, ponieważ wiele eksploitów wykorzystuje takie bajty; aby zasymulować działanie takiego eksploita, musimy mieć możliwość wysyłania takich samych bajtów z naszego klienta.

Na przykład załóżmy, że konieczne jest wysłanie ciągu 10 znaków reprezentujących japońskiego jena do serwera UDP nasłuchującego na porcie 5002 i że iptables musi wyłapać takie znaki. Zgodnie z zestawem znaków *ISO 8859-9* (więcej informacji po wpisaniu **`man iso_8859-9`** w linii poleceń systemu Linux) znak jena jest reprezentowany przez liczbę szesnastkową `A5`. Poniższe polecenia umożliwiają wykonanie doświadczenia.

Najpierw uruchamiamy iptables podając jako parametr `--hex-string` oraz bajty zapisane szesnastkowo zawarte pomiędzy dwoma znakami | jak poniżej:

```
[iptablesfw]# iptables -I INPUT 1 -p udp --dport 5002 -m string --hex-
↪string "|a5a5a5a5a5a5a5a5a5|" --algo bm -j LOG --log-prefix "YEN "
```

Następnie uruchamiamy serwer UDP nasłuchujący na porcie 5002[2]. W końcu wykorzystujemy Perla do wygenerowania serii 10 szesnastkowych bajtów `A5` i przekierowujemy to wyjście do programu Netcat, który wysyła je przez sieć do serwera UDP:

```
[iptablesfw]$ nc -u -l -p 5002
[ext_scanner]$ perl -e 'print "\xa5"x10' | nc -u 71.157.X.X 5002
```

[2] Uruchamianie serwera UDP w tym miejscu nie jest konieczne, ponieważ dane są wysyłane przez UDP bez wcześniejszego ustanawiania połączenia i iptables zobaczy pakiet UDP zawierający znaki jena niezależnie od tego, czy w przestrzeni użytkownika nasłuchuje serwer. Warto też zauważyć, że nie jest konieczne dodawanie reguły `ACCEPT` do polityki bezpieczeństwa, żeby wygenerowany został komunikat w logach (choć w takiej sytuacji dane nie dotrą do serwera z powodu domyślnej polityki DROP łańcucha `INPUT`). Aby zobaczyć, jak Netcat wyświetla dane po stronie serwera, należy dodać regułę ACCEPT dla portu 5002 UDP.

To wystarczy, by iptables dopasował odpowiednie dane, co można zobaczyć w logach (pogrubiony prefiks YEN):

```
[iptablesfw]# tail /var/log/messages | grep YEN
Jul 11 04:15:14 iptablesfw kernel: YEN IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X
➥DST=71.157.X.X
LEN=38 TOS=0x00 PREC=0x00 TTL=64 ID=37798 DF PROTO=UDP SPT=47731
➥DPT=5002 LEN=18
```

Definicje ataków w warstwie aplikacji

Atak w warstwie aplikacji można zdefiniować jako czynność wykonywaną w celu wykorzystania aplikacji, użytkownika lub danych aplikacji do celów innych niż przewidziane przez właściciela lub administratora aplikacji. Zazwyczaj ataki warstwy aplikacji nie opierają się na wykorzystywaniu technik niższych warstw, ale techniki te (takie jak fałszowanie pakietów IP — *IP spoofing* — oraz sklejanie sesji TCP — *TCP session splicing*) są czasem wykorzystywane do zmiany sposobu, w jaki ataki warstwy aplikacji są dostarczane do celu.

Możliwość przeprowadzania ataków warstwy aplikacji jest często wynikiem napiętych terminów przy tworzeniu programów. Programiści, pracując pod presją, nie mają wystarczająco dużo czasu na lokalizację i usuwanie błędów, co skutkuje słabościami bezpieczeństwa.

Dodatkowo wielu programistów nie rozważa następstw używania poszczególnych konstrukcji językowych, które mogą narazić aplikację na atak w niezbyt oczywisty sposób. W końcu wiele aplikacji ma złożoną konfigurację i bezpieczeństwo może zostać zmniejszone przez niedoświadczonych użytkowników uruchamiających aplikacje z włączonymi ryzykownymi opcjami.

Ataki warstwy aplikacji dzielą się na trzy kategorie:

Wykorzystujące błędy programu. Tworzenie aplikacji jest złożonym przedsięwzięciem i błędy programistyczne się zdarzają. W niektórych przypadkach błędy te mogą spowodować powstanie poważnych słabości narażonych na atak z sieci. Dobrym przykładem jest tutaj możliwość przepełnienia bufora powstała z powodu wykorzystania niebezpiecznej funkcji bibliotecznej C, słabości aplikacji web polegającej na przekazywaniu niekontrolowanych zapytań do bazy danych (co może skutkować atakami typu *SQL injection*) i słabości witryn umieszczających niefiltrowaną zawartość od użytkowników (co może skutkować atakami typu XSS — *Cross-Site Scripting*).

Wykorzystanie zaufania użytkowników. Niektóre ataki wykorzystują zaufanie użytkowników zamiast wykorzystywania błędów w oprogramowaniu. Takie ataki wyglądają na zupełnie poprawne działania w zakresie interakcji z samą aplikacją, ale celem jest wykorzystanie zaufania ludzi używających aplikacji.

Phishing jest tutaj dobrym przykładem; celem nie jest aplikacja web ani serwer pocztowy, tylko osoba czytająca sfałszowaną stronę internetową lub list elektroniczny.

Wyczerpanie zasobów. Podobnie jak w przypadku ataków *DoS* w warstwie transportowej lub sieci, aplikacja może ucierpieć z powodu nadmiaru danych wejściowych. Tego typu ataki powodują, że aplikacja przestaje być użyteczna dla wszystkich.

Nadużycia w warstwie aplikacji

Ciągle rosnąca złożoność aplikacji sieciowych powoduje, że wykorzystywanie słabości warstwy aplikacji staje się coraz łatwiejsze. W rozdziale 2. i 3. widzieliśmy kilka sposobów na kreatywne wykorzystanie warstwy sieciowej, ale tamte techniki są prozaiczne w porównaniu z niektórymi technikami wykorzystywanymi przeciwko aplikacjom.

Podczas gdy implementacje większości protokołów warstwy sieciowej i transportowej są zgodne z wytycznymi RFC, nie ma standardu opisującego, jak poszczególne aplikacje CGI powinny obsługiwać dane wprowadzane przez użytkownika z wykorzystaniem serwera WWW lub za pomocą aplikacji napisanych w językach programowania (jak C), które nie mają automatycznego sprawdzania zakresów lub zarządzania pamięcią. Czasem zupełnie nowa technika atakowania jest odkrywana i udostępniana społeczności — dobrym przykładem jest pomysł mechanizmu *HTTP Cross-Site Cooking*, który polega na wykorzystaniu ciastek www (*web cookies*) innej domeny (dokładniejszy opis na stronie *http://en.wikipedia.org/wiki/Cross-site_cooking*).

W kolejnych sekcjach opisanych jest kilka typowych ataków na warstwę aplikacji. Niektóre ataki mogą być wykryte przy wykorzystaniu dopasowań znaków iptables i odpowiednia reguła iptables jest dołączona do każdego przykładu. (Ale nie jest to pełna lista technik wykorzystywania aplikacji).

Sygnatury Snort

Jedną z lepszych metod zrozumienia ataków warstwy aplikacji jest przejrzenie zestawu sygnatur programu Snort[3]. Choć najnowsze sygnatury Snort nie są rozpowszechniane z kodem źródłowym Snort, projekt *Bleeding Snort* tworzy sygnatury najnowszych ataków w formacie programu Snort (*http://www.bleedingsnort.com*).

[3] W społeczności skupionej wokół programu Snort nazywa się sygnaturami pojedyncze reguły, ale w społeczności skupionej wokół systemów wykrywania włamań **sygnaturą** nazywa się mechanizm opisywania całych ataków. W tej książce te dwa terminy są używane wymiennie — nic nie ogranicza sygnatury do pojedynczego wzorca i dlatego poprawne jest nazywanie sygnaturami skomplikowanych opisów ataków.

UWAGA *Sygnatury Snort zostaną dokładniej omówione w rozdziale 9. Tutaj wprowadzone zostaną możliwości nadzoru warstwy aplikacji dostarczane przez Snort. Połączenie reguł iptables z sygnaturami Snort jest kluczem do wykorzystania możliwości wykrywania włamań iptables.*

Rozważmy następującą sygnaturę Snort:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-
➥ATTACKS /etc/shadow access"; content:"/etc/shadow";
➥flow:to_server,established; nocase;
➥classtype:w eb-application-activity; sid:1372; rev:5;)
```

Ta sygnatura wykrywa przesyłanie ciągu znaków `/etc/shadow` (pogrubiony powyżej) od klienta do serwera WWW. Serwer (i dowolny skrypt CGI, jaki uruchamia) najczęściej działa jako użytkownik niemający wystarczających uprawnień do odczytania pliku `/etc/shadow`, ale napastnik nie musi o tym wiedzieć, zanim nie spróbuje dostać się do pliku. Snort szuka prób odczytania pliku.

Aby iptables wygenerował do logów komunikat, gdy ciąg znaków `/etc/shadow` pojawi się w danych ustalonej sesji TCP z portem 80, należy ustawić następującą regułę:

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 80 -m state --state
➥ESTABLISHED   -m string --string "/etc/shadow"
➥--algo bm -j LOG --log-prefix "ETC_SHADOW "
```

Wykorzystanie przepełnienia bufora

Przepełnienie bufora (*buffer overflow*) to atak wykorzystujący błąd programistyczny w kodzie aplikacji polegający na tym, że rozmiar bufora jest niewystarczający do zmieszczenia skopiowanych danych; termin **przepełnienie** jest używany, ponieważ w takiej sytuacji nadpisywane są kolejne komórki pamięci za miejscem przeznaczonym na bufor. W przypadku przepełnienia bufora umieszczonego na **stosie** (*stack*) udane nadużycie powoduje nadpisanie adresu powrotu funkcji (również umieszczonego na stosie) w taki sposób, że wskazuje on na kod dostarczony przez napastnika. To umożliwia atakującemu kontrolowanie dalszego działania procesu. Inna klasa ataków przepełnienia bufora dotyczy obszarów pamięci dynamicznie alokowanych ze **sterty** (*heap*).

Słabości związane z przepełnianiem bufora występują zazwyczaj w aplikacjach C lub C++ z powodu niewłaściwego użycia pewnych funkcji bibliotecznych nieimplementujących automatycznego sprawdzania zakresów. Przykładami takich funkcji są `strcpy()`, `strcat()`, `sprintf()`, `gets()` i `scanf()` oraz złe zarządzanie pamięcią zaalokowaną ze sterty za pomocą takich funkcji jak `malloc()` i `calloc()`.

UWAGA *Znakomity opis pisania ataków przepełnienia bufora znajduje się w znanym dokumencie Smashing the Stack for Fun and Profit Alepha One (http://insecure.org/*
↪stf/smashstack.html). Hacking. Sztuka penetracji Jona Ericksona (Helion, 2004) jest również doskonałym źródłem technicznych informacji o tworzeniu eksploitów przepełniających bufor.

Przy atakach sieciowych nie ma uniwersalnego sposobu wykrywania prób przepełnienia bufora. Z drugiej strony w przypadku aplikacji przesyłającej dane przez zaszyfrowany kanał atak wypełniający bufor 50 powtórzeniami niezakodowanej litery A powinien być bardzo podejrzany. (Zaszyfrowane protokoły zazwyczaj nie przesyłają wielu powtórzeń tego samego znaku).

Jeśli istnieje taki sposób ataku i jest udostępniony publicznie, warto dodać regułę iptables wyszukującą tego typu zachowanie. Poniższa reguła może zostać użyta w przypadku komunikacji SSL. Warto zauważyć wielokrotnie powtórzoną literę A:

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 443 -m state --state
↪ESTABLISHED -m string --string
↪"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" -j LOG
↪--log-prefix "SSL OVERFLOW "
```

Ponieważ łatwo zmienić literę A wypełniającą bufor na dowolny inny znak, powyższą regułę łatwo obejść przez prostą modyfikację złośliwego kodu. Kod eksploita jest czasem używany przez zautomatyzowane robaki bez żadnych modyfikacji, dlatego powyższa strategia może być efektywna w niektórych przypadkach.

Zestaw sygnatur Snort zawiera wiele sygnatur ataków przepełnienia bufora, ale wykrywają one atak bez konieczności wykrywania specyficznych bajtów wypełniających. Czasem sam rozmiar danych dostarczonych jako argument pewnego polecenia aplikacji wskazuje na atak przepełnienia bufora. Na przykład poniższa sygnatura przepełnienia przy poleceniu `chown` serwera FTP. Wyszukuje ona co najmniej 100 bajtów danych po poleceniu `chown` w sesji FTP.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP SITE CHOWN
↪overflow attempt";
↪flow:to_server,established; content:"SITE"; nocase; content:"CHOWN";
↪distance:0; nocase;
↪isdataat:100,relative; pcre:"/^SITE\s+CHOWN\s[^\n]{100}/smi";
↪reference:bugtraq,2120; reference:cve,2001-0065;
↪classtype:attempted-admin; sid:1562; rev:11;)
```

Choć iptables nie obsługuje wyrażeń regularnych (w takim wypadku pogrubiony wyżej warunek `pcre` mógłby być zapisany w regule iptables bez zmian), możemy stworzyć dobre przybliżenie tej reguły Snort dla iptables. Na przykład poniższa reguła iptables wyszukuje ciągi znaków `site` i `chown` oraz wykorzystuje

dopasowanie `length` do wyszukania pakietu z co najmniej 140 bajtami. (Dodane jest 20 bajtów na nagłówek IP i 20 bajtów na nagłówek TCP, ponieważ dopasowanie `length` wykonywane jest z danymi nagłówka warstwy sieci).

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 21 -m state --state
➥ESTABLISHED -m string --string "site" --algo bm -m string --string
➥"chown" --algo bm -m length --length 140 -j LOG --log-prefix
➥"CHOWN OVERFLOW "
```

Ataki typu SQL injection

Ataki typu *SQL injection* wykorzystują taką sytuację w aplikacji, gdy dane wprowadzane przez użytkownika nie są sprawdzane ani poprawnie filtrowane przed włączeniem do zapytania SQL. Sprytny napastnik może wykorzystać możliwość zagnieżdżania SQL do stworzenia nowego zapytania i potencjalnego zmodyfikowania lub odczytania danych z bazy. Typowymi celami tego typu ataków są aplikacje CGI, które są wykonywane przez serwer WWW i stanowią interfejs do bazy danych.

Załóżmy na przykład, że aplikacja CGI sprawdza nazwę użytkownika i hasło w bazie, wykorzystując nazwę użytkownika i hasło wprowadzone za pomocą przeglądarki WWW przez skrypt CGI. Jeśli nazwa użytkownika i hasło nie są odpowiednio filtrowane, zapytanie wykorzystywane do przeprowadzenia weryfikacji może być podatne na wprowadzenie kodu SQL. Taki atak może zmienić zapytanie tak, że nie tylko dokona ono porównania, ale również zmodyfikuje dane dodatkowym zapytaniem. Napastnik może wykorzystać ten sposób do ustawienia hasła dowolnemu użytkownikowi, być może nawet administratorowi.

Trudno wykrywać typowe ataki SQL injection, ale kilka reguł Snort działa całkiem dobrze na wybrane typy ataków. Na przykład poniższa sygnatura *Bleeding Snort* wykrywa, że atakujący próbuje obciąć część zapytania SQL przez wprowadzanie pojedynczego znaku „'" w ❶ łącznie z dwoma znakami „-" w ❷ (z bajtami `NULL` po każdym znaku). Dwa znaki „-" powodują, że dalsza część zapytania SQL jest uznawana za komentarz, co można wykorzystać do usunięcia ograniczeń, które mogą być umieszczone dalej w zapytaniu jako dodatkowe połączenia za pomocą innych pól.

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 1433 (msg: "BLEEDING-EDGE
➥EXPLOIT MS-SQL SQL Injection closing string plus line comment"; flow:
➥to_server,established; content:❶"'|00|"; content:❷"-|00|-|00|";
➥reference:url,www.nextgenss.com/papers/more_advanced_sql_injection.pdf;
➥reference:url,www.securitymap.net/sdm/docs/windows/mssql-checklist.html;
➥classtype: attempted-user; sid: 2000488; rev:5; )
```

Tą regułę Snort można stosunkowo łatwo przetłumaczyć na regułę iptables łącznie ze znakami `NULL` dzięki opcji `--hex-string`.

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 1433 -m state
➥--state ESTABLISHED -m string --hex-string "'|00|" --algo bm -m string
➥--hex-string "-|00|-|00|" --algo bm -j LOG --log-prefix "SQL
➥INJECTION COMMENT "
```

Jedyną wadą sygnatury Snort i jej odpowiednika dla iptables jest to, że nie jest uwzględniana kolejność zawartych ciągów znaków. Jeśli pakiet będący częścią ustanowionego połączenia TCP zawiera te ciągi w odwrotnej kolejności (ze znakami `NULL` zapisanymi w notacji szesnastkowej Snort), na przykład `-|00|-|00| foo bar '|00|` zamiast `'|00| foo bar -|00|-|00|`, to zarówno sygnatura Snort, jak i reguła iptables zostanie dopasowana. W przypadku niektórych sygnatur może to spowodować wzrost ilości fałszywych alarmów, jeśli jest możliwe, by poprawne były dane zawierające fragmenty złośliwego kodu w zmienionej kolejności.

UWAGA *Więcej informacji o atakach typu SQL injection można znaleźć w dokumencie http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf.*

Czynnik ludzki

Jednymi z najbardziej problematycznych ataków w dzisiejszym Internecie są te skierowane bezpośrednio przeciwko ludziom używającym aplikacji. Tego typu ataki obchodzą najlepsze algorytmy szyfrowania i schematy uwierzytelniania dzięki wykorzystaniu ludzkiej naiwności. Na przykład jeśli napastnik skłoni osobę do użycia lub pobrania i użycia złośliwej aplikacji albo podania lub użycia podrobionego klucza szyfrującego czy hasła, może przejść nawet najbardziej zaawansowane mechanizmy bezpieczeństwa. Czasem wykorzystanie ludzi może być łatwiejsze niż znalezienie dziury w zabezpieczonym systemie, aplikacji lub schemacie szyfrowania.

Phishing

Phishingiem nazywa się atak, w którym użytkownik zostaje skłoniony do podania w nieodpowiednim miejscu danych autoryzacyjnych zasobu sieciowego takiego jak na przykład konto bankowe. Zazwyczaj atak taki przeprowadzany jest przez wysłanie do użytkowników oficjalnie wyglądającego listu elektronicznego z żądaniem zalogowania się do konta internetowego i wykonania pewnych „pilnych" czynności związanych z bezpieczeństwem jak zmiana hasła. (Gdyby nie przykre efekty udanego ataku, można byłoby to uznać za świetny dowcip). Zamieszczony odnośnik wygląda na poprawny, ale jest nieznacznie zmieniony, tak by kierować użytkownika na stronę kontrolowaną przez napastnika, ale bardzo podobną do prawdziwej. Gdy tylko zaatakowani użytkownicy odwiedzą stronę i wpiszą swoje dane, są one przechwytywane przez napastnika.

Poniżej zamieszczony jest przykładowy list, który otrzymałem ze sfałszowanego adresu *support@citibank.com* z tematem *Citibank Online Security Message*:

```
When signing on to Citibank Online, you or somebody else have made
➥several login attempts and reached your daily attempt limit. As an
➥additional security measure your access to Online Banking has been
➥limited. This Web security measure does not affect your access to
➥phone banking or ATM banking. Please verify your information
➥<a href="http:// 196.41.X.X/sys/" onMouseMove="window.status=
➥'https://www.citibank.com/us/cards/index.jsp';return true;"
➥onMouseout="window.status=''">here</a>, before trying to sign on again.
➥You will be able to attempt signing on to Citibank Online within
➥twenty-four hours after you verify your information. (You do not have
➥to change your Password at this time.)
```

Nieszkodliwe sformułowania sprawiają wrażenie serdecznego i pomocnego nastawienia („*several login attempts*" i „*You do not have to change your password…*"), a odnośnik internetowy jest chytrze zmodyfikowany. Zawiera on wbudowany fragment kodu JavaScript powodującego, że przeglądarka po jego wskazaniu wyświetla poprawny odnośnik do strony Citibanku w pasku statusu, mimo że link naprawdę wskazuje adres *http://196.41.X.X/sys* na serwerze kontrolowanym przez napastnika[4]. Serwer ten wyświetla stronę wyglądającą identycznie jak poprawna strona na prawdziwym serwerze Citibanku.

Na szczęście iptables może wykryć ten konkretny list przeglądany w sesji WWW za pomocą następującej reguły:

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 25 -m state --state
➥ESTABLISHED -m string --string ❶"http://196.41.X.X/sys/" --algo bm -m
➥string --hex-string  ❷ "window.status=|27|https://www.citibank.com" -j
➥LOG --log-prefix "CITIBANK PHISH "
```

W ❶ i ❷ reguła wykonuje dopasowanie dwóch ciągów znaków `"http://196.`
`➥41.X.X/sys/"` i `"window.status='https://www.citibank.com"` do danych przesyłanych w połączeniach TCP z portem SMTP. Pierwszy ciąg znaków w sygnaturze dopasowuje konkretny fałszywy serwer i dlatego ta reguła nie chroni przed innymi podobnymi atakami na konta Citibanku. Drugi ciąg znaków jest również istotny, ponieważ szuka adresu strony WWW Citibanku podanego jako argument właściwości `window.status` JavaScript. Podczas gdy prawdziwa strona Citibanku może również zawierać tego typu konstrukcje, kombinacja tych dwóch ciągów znaków w jednym liście elektronicznym jest bardzo podejrzana i ryzyko wywołania fałszywego alarmu przez Snort lub iptables jest niewielkie (niezależnie od kolejności wystąpienia wzorców).

[4] Nie wszystkie przeglądarki działają w ten sam sposób; widziałem Microsoft IE wyświetlający poprawny adres internetowy w pasku statusu, podczas gdy Firefox wyświetlał sfałszowany odnośnik (prawdopodobnie wynikało to z tego, że wersja Firefoxa, której używałem, nie interpretowała kodu JavaScript wbudowanego w ten sposób w znaczniki.

Należy maksymalizować efektywność sygnatur nowych ataków przez zachowanie balansu pomiędzy zwiększaniem czułości detekcji i redukowaniem częstotliwości fałszywych alarmów. Jednym z najlepszych sposobów jest wyszukiwanie wzorców, które nie powinny pojawić się przy poprawnej komunikacji w sieci. Jeśli pojawi się nowy atak na nowy cel, dobrymi kandydatami na wzorce do zawarcia w sygnaturze są adres IP złośliwego serwera (choć może być zawsze zmieniony przez napastnika) i dowolny fragment tekstu lub kodu (jak ciąg znaków `window.status` w przykładzie z Citibankiem).

Backdoory i logowanie klawiszy

Backdoorem nazywa się program, który zawiera funkcjonalność udostępnioną napastnikowi bez wiedzy uprawnionego użytkownika. Na przykład trojan *Sdbot*[5] umożliwia dostęp do systemu, łącząc się z kanałem IRC, gdzie napastnik czeka, żeby wydać polecenie. Backdoor jest napisany w ten sposób, że atakujący musi wprowadzić odpowiednie hasło, zanim jakiekolwiek polecenie zostanie wykonane. To wprowadza pewien rodzaj autoryzacji przy komunikacji z backdoorem i zapewnia, że tylko napastnik, który włamał się do systemu, może go kontrolować.

Celem backdoora jest niezauważalne umożliwienie napastnikowi wykonywania dowolnej czynności na zdalnym komputerze od zapisywania wciśniętych klawiszy w celu poznania haseł do zdalnego kontrolowania całego systemu. Niektóre backdoory uruchamiają nawet swoje własne sniffery sieciowe skonfigurowane do wyszukiwania informacji o nazwach użytkowników i hasłach z protokołów, które przesyłają takie dane czystym tekstem jak telnet lub FTP (choć przejęcie takich informacji z innych systemów jest mało prawdopodobne w sieciach opartych na przełącznikach, chyba że backdoor jest zainstalowany na urządzeniu pracującym jako gateway lub firewall). *FsSniffer* jest przykładem takiego backdoora. Jest on wykrywany za pomocą poniższej reguły Snort:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"BACKDOOR FsSniffer
➥connection attempt"; flow: ❶ to_server,established; content:
➥❷ "RemoteNC Control Password|3A|"; reference:nessus,11854;
➥classtype:trojan-activity; sid:2271; rev:2;)
```

W ❶ reguła Snort kontroluje pakiety będące częścią ustanowionego połączenia TCP i skierowane do serwera, a w ❷ reguła szuka w danych warstwy aplikacji ciągu znaków, który jednoznacznie identyfikuje próbę autoryzacji w FsSniffer[6].

Po przeniesieniu do iptables reguła ta ma postać przedstawioną poniżej. (Dopasowanie stanu `ESTABLISHED` w ❶ zapewnia, że pakiet jest częścią ustano-

[5] Więcej informacji na stronie *http://www.symantec.com/security_response/writeup.jsp?docid=*
➥*2002-051312-3628-99&tabid=2*.

[6] W zasadzie ktoś mógłby wysłać ciąg znaków `RemoteNC Control Password:` do dowolnego serwera TCP niekoniecznie w celu autoryzacji w backdoorze *FsSniffer*, ale mimo wszystko takie działanie jest podejrzane.

wionego połączenia TCP, a argument `--hex-string` w ❷ powoduje, że kod szesnastkowy `\x3A` zawarty w oryginalnej regule jest poprawnie przeniesiony).

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp -m state --state ❶
➥ESTABLISHED -m string --hex-string ❷ "RemoteNC Control Password|3A|"
➥--algo bm  -j LOG --log-ip-options --log-tcp-options --log-prefix
➥"FSSNIFFER BACKDOOR "
```

Szyfrowanie i kodowanie danych w aplikacji

Dwa czynniki utrudniają wykrywania ataków warstwy aplikacji: szyfrowanie i schematy kodowania w aplikacji. Szyfrowanie jest szczególnie problematyczne, ponieważ jest zaprojektowane tak, by deszyfracja była praktycznie niemożliwa bez odpowiedniego klucza, do którego zazwyczaj systemy IDS, IPS i firewalle nie mają dostępu[7].

Jednak niektóre eksploity warstwy aplikacji nie muszą być szyfrowane, by zadziałały. Istnieją na przykład sygnatury Snort (z konieczności operujące czystym tekstem) dla pewnych ataków na serwery SSH. W przypadku takich sygnatur Snort kontroluje ilość danych bez wykorzystywania kluczy szyfrujących SSH. Istnienie tych sygnatur uświadamia nam, że samo szyfrowanie nie jest panaceum i napastnik może w pewnych sytuacjach wykorzystać słabości w aplikacjach niezależnie od wymaganego poziomu szyfrowania. To znaczy, że słabości mogą istnieć w funkcjach, które są dostępne w sposób nieszyfrowany.

Techniki kodowania również mogą utrudniać pracę IDS. Na przykład wiele przeglądarek internetowych wspiera kodowanie gzip w celu zredukowania rozmiaru danych przesyłanych przez sieć, gdyż zazwyczaj kompresja lub dekompresja danych za pomocą szybkiego procesora trwa krócej niż przesłanie nieskompresowanych danych przez wolną sieć. Jeśli atak jest połączony z pewną ilością przypadkowych danych i skompresowany przez gzip, IDS musi rozpakować dane, żeby wykryć atak. Przypadkowe dane zapewniają, że skompresowany atak wygląda za każdym razem inaczej; bez tej przypadkowo generowanej części IDS mógłby po prostu szukać ciągu znaków po kompresji. Przy dużym ruchu sieciowym dekompresja każdej sesji web w czasie rzeczywistym jest praktycznie niemożliwa, ponieważ wiele sesji przesyła duże skompresowane pliki bez złośliwego kodu.

UWAGA *Nie wszystkie sposoby kodowania warstwy aplikacji są trudne do zdekodowania dla IDS. Na przykład dane zapisane w URL są dekodowane w czasie rzeczywistym przez preprocesor HTTP Snort i są dostępne przy wykorzystaniu słowa kluczowego uricontent w sygnaturach Snort. Jest to możliwe, jako że kodowanie*

[7] Niektóre IDS umożliwiają przechowywanie klucza SSL i sprawdzania po odszyfrowaniu danych przesyłanych do serwera WWW w postaci zaszyfrowanej.

URL opiera się na prostej operacji podstawienia kodów szesnastkowych i znaków procenta. Na przykład `A` staje się `%41`; przy odkodowaniu dokonuje się odwrotnej zamiany. Takie kodowanie nie jest trudne obliczeniowo.

Obrona w warstwie aplikacji

Z technicznego punktu widzenia obrona w warstwie aplikacji powinna wykorzystywać jedynie konstrukcje z warstwy aplikacji. Gdy na przykład użytkownik aplikacji jej nadużywa, jego konto powinno zostać wyłączone. Jeśli napastnik dokonuje ataku typu SQL injection przez aplikację CGI wykonywaną na serwerze, zapytanie powinno zostać odrzucone, a do klienta odesłany odpowiedni kod błędu HTTP. Taka reakcja nie wymaga manipulacji informacjami z nagłówka pakietu, które istnieją poniżej warstwy aplikacji.

Jednak obrona tylko w warstwie aplikacji jest niepraktyczna w firewallach i sieciowych systemach zapobiegania włamaniom, ponieważ nie są one blisko związane z samymi aplikacjami[8]. Dlatego jeśli bardzo złośliwy atak jest odkryty z pewnego adresu IP w sesji TCP (wymagającej komunikacji dwukierunkowej), bardziej korzystne jest zablokowanie komunikacji z adresem IP napastnika. Jest to obrona w warstwie sieci na atak w warstwie aplikacji.

W tej książce podkreślamy bardziej sposoby reagowania w warstwie sieci i warstwie transportowej na ataki w warstwie aplikacji niż reakcje samych aplikacji. Jest to realne dzięki możliwości tworzenia i zarządzania regułami blokującymi iptables (zarządzanymi przez projekt *psad*) wobec adresu IP napastnika oraz przez użycie celu `REJECT` do zakończenia połączenia TCP przez fwsnort. Rozdziały 10. i 11. opisują szczegółowo tego typu reakcje.

[8] Istnieją mechanizmy bezpieczeństwa ściśle związane z aplikacjami (takie jak moduł *ModSecurity* dla serwera *Apache*), ale firewalle i systemy wykrywania włamań nie mają informacji o działaniu tych mechanizmów.