Wykrywanie ingerencji sieciowych (IDS)

Michał Karbowańczyk

Spis treści

Notatki

Indeks alfabetyczny

Indeks tabel

Indeks rysunków

1 Systemy wykrywania ingerencji

Przez systemy wykrywania ingerencji (ang. Intrusion Detection Systems, IDS) rozumie się wszelkiego rodzaju oprogramowanie czy urządzenia, które analizują pracę systemów informatycznych (zarówno systemów operacyjnych i serwerów usług jak i infrastruktury sieciowej) i wykrywają nieprawidłowości mogące świadczyć o próbie (udanej lub nie) ingerencji w ich działanie.

Ingerencja a włamanie

Popularne, ale nieprawidłowe tłumaczenie angielskiej nazwy systemów IDS brzmi „systemy wykrywania włamań". Angielskie słowo „intrusion" tłumaczy się nie jako „włamanie", tylko „ingerencja", „najście", „wtargnięcie" czy „niepokojenie", a więc pojęcia znacznie szersze.

Przez włamanie rozumie się sytuację, w której atakujący (włamywacz) uzyskuje nieuprawniony dostęp do danych znajdujących się w systemie (włączając w to dane przesyłane w infrastrukturze sieciowej). Włamanie może być przeprowadzone na wiele sposobów:

- przejęcie kontroli nad procesami działającymi w systemie i umożliwienie sobie wykonywania operacji bez konieczności uwierzytelnienia w systemie;
- podszywanie się pod użytkownika systemu, na przykład poprzez skompromitowanie jego poufnych danych uwierzytelniających go w systemie;
- nabycie uprawnień większych niż przewidziane w polityce bezpieczeństwa systemu (w przypadku użytkowników danego systemu);
- w przypadku infrastruktury sieciowej – nieuprawniona zmiana jej konfiguracji lub uzyskanie dostępu do jej elementów.

Udane włamanie prowadzi do osiągnięcia skutku, którym jest z reguły:

- dostęp do danych wartościowych z punktu widzenia włamywacza;
- modyfikacja danych przynosząca korzyść włamywaczowi;
- modyfikacja lub niszczenie danych przynosząca straty właścicielowi systemu;
- wykorzystanie systemu do przeprowadzenia ataku na inne systemy (tzw. system-zombie);
- pozostawienie sobie tzw. furtki (ang. *backdoor*) umożliwiającej kontrolę nad działaniem systemu w przyszłości;
- pozostawienie „odcisku palca" (ang. *fingerprint*) – informacji możliwie szeroko dostępnej (np. w formie modyfikacji oryginalnej witryny WWW), świadczącej o udanym włamaniu, w celu zapewnienia rozgłosu włamywaczowi.

W przeprowadzaniu włamania na ogół występują przynajmniej dwie fazy: rozpoznanie możliwości ataku i przeprowadzenie właściwego ataku. Dla przykładu, włamywacz chcący przejąć kontrolę nad serwerem prawdopodobnie zacznie od zbadania dostępnych na nim usług poprzez skanowanie portów, a następnie użyje programu (ang. *exploit)* wykorzystującego słabości danego serwera usługi i przejmie kontrolę nad procesem serwera. Każda z tych czynności będzie ingerencją w rozumieniu systemów IDS.

Ponadto nie każda ingerencja jest związana z przeprowadzaniem włamania. Drugim popularnym typem ataków są działania mające na celu uniemożliwienie prawidłowego działania atakowanego systemu (ang. *Denial of Service*, *DoS*). Trzy podstawowe rodzaje takich ataków to: podawanie fałszywych danych prowadzące do błędnego działania systemu (np. „zatruwanie" tablic ARP, ang. *ARP Poisoning*), podawanie nieprawidłowych, nie uwzględnionych w standardach danych skutkujące np. zawieszeniem się procesu obsługującego te dane (np. *Ping Of Death*) oraz zalewanie (ang. *flooding)* systemu danymi prawidłowymi w ilości uniemożliwiającej właściwą pracę systemu (np. *SYN flood* – otwieranie olbrzymiej ilości połączeń TCP, często z wykorzystaniem dużej liczby systemów-zombie).

Notatki

Rodzaje systemów IDS

Wykrywanie ingerencji jest często utożsamiane z wykrywaniem ingerencji sieciowych, tj. anomalii w ruchu sieciowym lub przesyłaniu danych świadczących o potencjalnej próbie przeprowadzenia ataku na sama infrastrukturę sieciową lub na korzystające z niej systemy komputerowe. Tymczasem pojęcie ingerencji nie ogranicza się do transmisji danych w sieciach komputerowych; oprócz tego możemy mówić o ingerencjach dokonywanych lokalnie, w ramach danego systemu operacyjnego, oraz o ingerencjach dokonywanych w sprzęcie składającym się na system informatyczny.

Lokalne IDS (HIDS)

Lokalne wykrywanie ingerencji (ang. *Host-based IDS*) jest właściwie integralną częścią każdego systemu operacyjnego, który obsługuje wiele kont użytkowników. Każdy taki system zapisuje bowiem do logów informacje o nieudanym uwierzytelnieniu użytkownika, a zatem choćby najprostsze narzędzie umożliwiające analizę tych logów pod kątem nieprawidłowości można uznać za lokalny IDS. Innym przykładem lokalnego IDS jest monitorowanie ingerencji w system plików – czy to za pomocą sum kontrolnych (np. szeroko znany *Tripwire* czy jego otwarty, będący w początkowej fazie rozwoju odpowiednik *AIDE*), czy to poprzez kontrolowanie wywołań systemowych jądra (np. mechanizm *audit* w jądrze Linux). Również logi dotyczące operacji zabronionych przez mechanizm SELinux można potraktować jako przykład lokalnego IDS.

Sieciowe IDS (NIDS)

Są to systemy wykrywające ingerencje na podstawie analizy ruchu sieciowego (ang. *Network-based IDS*). Niniejszy rozdział jest poświęcony jednemu z najpopularniejszych rozwiązań w tej dziedzinie – sieciowemu systemowi wykrywania ingerencji *Snort*. Jako prosty system IDS może też służyć standardowy, zawarty w jądrze Linux i omówiony wcześniej filtr pakietów, jeżeli tylko użyte zostaną reguły zapisujące do logów pakiety stanowiące potencjalną ingerencję.

HIDS a NIDS. Hybrydowe (zintegrowane) IDS

Łatwo zauważyć, że wiele ingerencji może zostać wykryte zarówno poprzez systemy HIDS, jak i NIDS. Dla przykładu, atak polegający na próbie uwierzytelnienia w systemie do konsoli SSH poprzez odgadnięcie hasła użytkownika, zostanie zarejestrowany zarówno przez lokalny IDS (jako wiele nieudanych prób uwierzytelnienia), jak i sieciowy IDS (nawiązywanie wielu połączeń z usługą SSH).

Oba rodzaje IDS mają swoje przewagi i słabości. Systemy lokalne są przede wszystkim narażone na zakłócenie działania przez włamywacza; jeżeli ten bowiem uzyska prawa administrowania systemem, to może zarówno wyłączyć zapisywanie logów, jak i usunąć zapisy już istniejące (zacieranie śladów). Dlatego tak ważne jest skonfigurowanie systemu logów tak, aby niemożliwe było przynajmniej usunięcie informacji o czynnościach wykonywanych przez włamywacza przed przejęciem przez niego praw administracyjnych – zazwyczaj realizuje się to poprzez wysyłanie logów na inny system i/lub na nośnik trwały (np. drukowanie). Ponadto systemy te z natury są rozproszone, ich obsługa na wielu systemach operacyjnych może być uciążliwa.

Słabości te nie są cechą sieciowych IDS dzięki temu, że systemy operacyjne na których pracują NIDS nie są same w sobie celem ataków, a przy właściwej konfiguracji (tzw. tryb *stealth*) zwyczajnie nie mogą być zaatakowane, a zatem ich działanie jest gwarantowane. Nie oznacza to, że nie istnieją sposoby zabezpieczania się przez atakujących przed wykryciem przez IDS: możliwe jest bowiem stosowanie uników (ang. *evasion*) mających wprowadzić w błąd system IDS. Można tu wymienić trzy podstawowe techniki: przeprowadzanie ataku tak, aby został on uznany za prawidłowe działanie; generowanie fałszywych alarmów mających na celu „uśpienie czujności" administratora zarządzającego danym IDS czy wreszcie ukrycie swojej tożsamości, np. poprzez używanie do prowadzenia ataku wcześniej przejętego systemu-zombie. Natomiast z samej swej natury systemy NIDS nie mogą wykryć ingerencji prowadzonych bez użycia sieci (np. próba uzyskania szerszych uprawnień dokonywana w lokalnej konsoli systemu), ponadto w przypadku włamań z reguły mogą one wykryć próbę dokonania włamania, ale nie mogą ocenić skuteczności tej próby.

Wnioskiem nasuwającym się po tej pobieżnej analizie cech systemów HIDS i NIDS jest konieczność używania systemów zarówno jednego, jak i drugiego rodzaju dla zapewnienia maksymalnego bezpieczeństwa systemu informatycznego. Spowodowało to wykształcenie się kategorii hybrydowych IDS (ang. *Hybrid IDS*), które łączą funkcjonalności IDS lokalnych i sieciowych. System taki na ogól zbudowany jest z centralnego modułu zarządzającego i analizującego informacje oraz modułów – agentów, instalowanych w newralgicznych punktach systemu informatycznego i pełniących rolę HIDS lub NIDS, przesyłających dane do modułu centralnego. Można zatem powiedzieć, że hybrydowy IDS jest tak naprawdę systemem scentralizowanego monitorowania pracy właściwych systemów HIDS lub NIDS rozlokowanych w systemie informatycznym. Wiodącym przykładem tego typu rozwiązań jest darmowy, ale licencjonowany na zasadzie własnościowej projekt *Nessus*. Inny przykład to otwarty, znajdujący się w fazie rozwoju projekt *Prelude IDS*, który m. in. wykorzystuje *Snort* jako moduł sieciowego IDS. Warto dodać, że istnieje koncepcja zestandaryzowania wymiany informacji między systemami IDS (IDMEF, ang. *Intrusion Detection Message Exchange Format*), która została opublikowana jako RFC 4765, póki co mający status eksperymentalny.

? *Zobacz też: www.nessus.org www.prelude-ids.org RFC 4765*

IDS protokołów warstwy aplikacji (PIDS, APIDS)

Są to systemy typu NIDS, analizujące ruch sieciowy włączając w to zdekodowanie i interpretację transmitowanych danych. Analiza może dotyczyć właściwości danego protokołu warstwy aplikacji i mówimy wtedy o systemie PIDS (ang. *Protocol-based IDS*), np. analizator ruchu przychodzącego do serwera HTTP wykrywający użycie w nagłówku żądania metody PUT, która służy do transferu plików na serwer i z reguły nie jest wykorzystywana. W bardziej zaawansowanych, najczęściej tworzonych pod konkretne potrzeby rozwiązaniach, analizie może podlegać również semantyka przesyłanych danych, np. można śledzić zapytania SQL wysyłane przez aplikację do bazy danych i stwierdzać próbę pozyskania danych niepotrzebnych do prawidłowego działania aplikacji. Mówimy wówczas o systemie APIDS (ang. *Application and Protocol-based IDS*).

Fizyczne IDS

Kategorią systemów IDS całkowicie wykraczającą poza ramy tego opracowania są systemy wykrywające fizyczną (sprzętową) ingerencję w systemy informatyczne. Najprostszym przykładem mogą tu służyć czujniki wykrywające fakt otwarcia obudowy komputera i w konsekwencji wymuszające podanie ustalonego wcześniej hasła, zanim będzie możliwe uruchomienie komputera. Warto też wspomnieć o możliwości wykrywania prób podsłuchiwania ruchu sieciowego poprzez elektryczne podłączenie się do kabla sieciowego, na bazie zmian we właściwościach elektrycznych przewodów.

Notatki

2 Implementacja sieciowego IDS

Właściwa implementacja jest podstawowym warunkiem prawidłowego i skutecznego działania systemów typu sieciowego IDS. Przez implementację rozumiemy tutaj wybór ruchu, który będzie poddany analizie przez IDS, oraz sposób dostarczenia tych danych do IDS.

IDS a zapora sieciowa

W większości przypadków rozpatrywana topologię sieci da się sprowadzić do uproszczonego schematu, w którym mamy do czynienia z jednym lub kilkoma segmentami sieci lokalnej (LAN) oraz z połączeniem do sieci Internet, pomiędzy nimi zaś znajduje się ruter oraz zapora sieciowa (firewall) ograniczająca ruch pomiędzy siecią lokalną a siecią Internet.

Jedną z fundamentalnych decyzji jest tu umiejscowienie systemu IDS (dokładniej: połączenia dostarczającego dane do systemu IDS) względem rutera i zapory. Możliwe są tu dwa podejścia: podłączenie IDS od strony sieci lokalnej (tzw. IDS za zaporą) oraz po stronie sieci Internet (tzw. IDS poza zaporą).

Na pierwszy rzut oka wydaje się, że właściwe jest umiejscowienie systemu IDS poza zaporą. Argumentem za taką decyzją jest możliwość wykrywania prób ataków, które są zatrzymywane przez zaporę sieciową. Informacje takie mogą służyć udoskonaleniu ochrony sieci, a także umożliwiają ewentualne podjęcie kroków prawnych przeciwko osobie odpowiedzialnej za prowadzenie ataków. Z drugiej strony jednak takie umiejscowienie IDS nie pozwala na monitorowanie ruchu w sieci lokalnej, co jest istotne, jeżeli do sieci tej mogą uzyskiwać osoby niepowołane (np. sieci bezprzewodowe) lub gdy osoby mające dostęp do sieci lokalnej nie są zaufane. Z kolei umiejscowienie IDS za zaporą daje właściwości odwrotne – analizowana jest tylko ta część ruchu pochodzącego z zewnątrz, która zostaje „przepuszczona" przez zaporę sieciową, natomiast dostępny jest cały ruch wytwarzany wewnątrz sieci lokalnej.

Biorąc pod uwagę, że to w sieci lokalnej znajdują się będące przedmiotem ochrony serwery, można zaryzykować stwierdzenie, iż monitorowanie ruchu powinno się odbywać właśnie sieci lokalnej; w ten sposób można bowiem wykrywać ingerencje potencjalnie zagrażające bezpieczeństwu serwerów i/lub stacji roboczych, pochodzące zarówno z zewnątrz, jak i z wewnątrz sieci lokalnej.

Oczywiście nic nie stoi na przeszkodzie, aby analizować ruch zarówno w sieci lokalnej, jak i połączeniu z siecią Internet. Należy jednak mieć świadomość, że im większą ilość ruchu podda się analizie, tym więcej informacji będzie wynikiem tej analizy, i tym więcej pracy trzeba będzie włożyć w interpretacje wyników. W szczególności otrzymywanie dużej liczby fałszywych alarmów jest istotnym czynnikiem psychologicznym, który może prowadzić do „uśpienia czujności" administratora i w konsekwencji do przeoczenia wystąpienia rzeczywistego zagrożenia. Sytuacja taka może być wywołana przez atakującego świadomie i szczególnie narażone są na to systemy IDS umiejscowione poza zaporą. Analogiczne rozumowanie można zastosować do zwiększania ilości systemów IDS pracujących w sieci lokalnej.

Akwizycja danych dla IDS

Istotnym technicznym problemem jest sposób pozyskania danych, które mają podlegać analizie. Z oczywistych względów nie można po prostu wymusić przekazania danych do IDS, gdyż nie dotrą one wówczas do właściwego celu. Potrzebne jest więc zduplikowanie danych i przesłanie duplikatu do IDS.

Popularnym rozwiązaniem tego problemu jest port monitorujący (ang. *monitor port*, *mirror port*, *span port*) wbudowany w przełącznik lub ruter. Port taki jest powiązany z innym „zwykłym" portem lub grupą portów urządzenia, i jest przez niego wysyłana kopia danych przesyłanych portem (portami) dowiązanym.

Port monitorujący jest rozwiązaniem wygodnym, ale posiada oczywiste ograniczenia o charakterze ilościowym. Pełna duplikacja danych przy zachowaniu pełnej wydajności nie jest możliwa nawet wtedy, gdy port monitorujący jest powiązany z jednym portem urządzenia, ponieważ port urządzenia może pracować w trybie pełnego dupleksu, natomiast port monitorujący dokonuje tylko wysyłania danych.

W takim rozwiązaniu monitorowaniu podlega na ogół port, którym urządzenie jest połączone z innym przełącznikiem lub ruterem, czyli tzw. *uplink port*. Takie rozwiązanie pozwala na śledzenie ruchu pomiędzy segmentami sieci, natomiast nie zapewnia śledzenia ruchu wewnątrz danego segmentu.

Jeżeli port monitorujący jest związany z grupą portów urządzenia, to przy pełnym obciążeniu tych portów musimy liczyć się albo ze stratami danych (nie wszystkie przesyłane dane trafią do portu monitorującego), albo ze znacznym zredukowaniem wydajności monitorowanej grupy portów. W szczególności, jeżeli monitorowaniu poddamy wszystkie porty przełącznika, to jego wydajność może zostać zredukowana do wydajności koncentratora.

Port monitorujący może być również skonstruowany w formie osobnego urządzenia, które włącza się w połączenie sieciowe na zasadzie podobnej do wzmacniacza sygnału. W szczególności jako prosty osobny port monitorujący może zostać użyty zwykły koncentrator, którego dwa porty służą do utrzymania podsłuchiwanego połączenia, zaś trzeci port dostarcza dane do IDS. Oczywiście wydajność łącza jest wówczas zredukowana do trybu półdupleksu, natomiast jest to rozwiązanie zdecydowanie najtańsze.

Podobnym rozwiązaniem jest zastosowanie specjalizowanego urządzenia podsłuchującego (ang. *tap device*, *network tap*). Jest to urządzenie podłączane do konkretnego połączenia sieciowego podobnie jak osobny port monitorujący, z tym że jest to z reguły urządzenie pasywne, które nie zakłóca transferu danych w połączeniu nawet w przypadku braku zasilania. Ponadto w przypadku połączeń pracujących w pełnym dupleksie, czyli wykorzystujących dwa łącza fizyczne (np. dwie pary przewodów w łączu Ethernet) urządzenie podsłuchujące posiada dwa porty wyjściowe, na które kopiowany jest ruch z obu kierunków transmisji. Dzięki takiemu rozwiązaniu gwarantowana jest akwizycja danych bez strat przy zachowaniu pełnej wydajności monitorowanego łącza. Urządzenia te są stosunkowo kosztowne, dla przykładu cena najprostszych modeli przenośnych umożliwiających podsłuchiwanie jednego połączenia Ethernet 10/100 wynosi około 120-150 USD.

Wyjścia wielu urządzeń monitorujących mogą być podłączone do jednego IDS, o ile jego konstrukcja przewiduje taką możliwość, tj. jest on wyposażony w wiele portów. Należy natomiast wystrzegać się pokusy wykorzystania urządzenia łączącego (koncentratora, przełącznika) w celu zagregowania monitorowanych danych i przesłania ich do pojedynczego wejścia IDS. Poza zdegradowaniem wydajności rozwiązanie takie niesie ze sobą niebezpieczeństwo stworzenia niezaplanowanego połączenia pomiędzy monitorowanymi segmentami sieci.

Izolacja IDS

Sieciowy IDS powinien być całkowicie niewidoczny dla sieci, którą monitoruje; inaczej mówiąc, w sieci nie powinny pojawiać się żadne dane świadczące o jego istnieniu. W przypadku specjalistycznych urządzeń czy portów monitorujących połączenia oparte na technologii pełnego dupleksu jest to zrealizowane w ten sposób, że na łączu pomiędzy urządzenie monitorującym a IDS podłączona jest tylko linia (np. para przewodów w przypadku skrętki miedzianej) umożliwiająca transmisję w kierunku od urządzenia monitorującego do IDS. W przypadku monitorowania sieci opartej na technologii pracującej w trybie półdupleksu i wykorzystującej tylko jedną linię transmisyjną, to sam IDS musi działać tak, aby nie generować żadnych danych.

Oczywiście do efektywnego korzystania z IDS potrzebna jest metoda komunikacji z nim. Do tego celu IDS musi być wyposażony w port zarządzania, który umożliwia normalną wymianę danych między IDS a systemem, na którym pracuje administrator. Port taki powinien być całkowicie odseparowany od sieci, która podlega monitorowaniu. Podobne rozwiązania stosuje się w serwerach, i wówczas porty zarządzania wszystkich systemów łączy się osobnym segmentem sieci, tzw. siecią zarządzającą. Sieć taka jest niedostępna z innych segmentów, w szczególności z sieci Internet (powinna to być separacja na poziomie fizycznym), i można się nią posługiwać tylko pracując na dedykowanej stacji roboczej (konsoli zarządzania).

Programowe IDS

Problem izolacji

Przez programowe IDS rozumiemy IDS zrealizowane w formie oprogramowania zainstalowanego na komputerze i pracujące pod kontrolą systemu operacyjnego ogólnego przeznaczenia (np. GNU/Linux). Systemy te używają standardowych urządzeń sieciowych, transmitujących dane dwukierunkowo, i jeżeli urządzenia monitorujące, które dostarczają dane do IDS, nie zapewniają transmisji jednokierunkowej, to należy zadbać o jego prawidłową konfigurację tak, aby zachować izolację IDS od sieci monitorowanej. W przypadku systemu Linux interfejs sieciowy, do którego dostarczane są monitorowane dane, powinien nie mieć nadanego adresu logicznego (IP), natomiast, aby umożliwić odczytywanie przez interfejs monitorowanego ruchu, interfejs musi być przełączony w tryb promiskuistyczny (ang, *promiscuous mode*) zwany też trybem rozwiązłym. W trybie tym interfejs odbiera i przekazuje do warstw wyższych wszystkie ramki, bez względu na ich adres fizyczny.

Rozwiązania zintegrowane

W przypadku, gdy realizacja infrastruktury sieciowej ma się odbyć minimalnym kosztem, możliwe jest zintegrowanie w jednym systemie (serwerze) funkcjonalności rutera, zapory sieciowej i sieciowego IDS. W takim przypadku należy mieć świadomość, że nie jest spełniona zasada izolacji systemu IDS od monitorowanej sieci, a zatem sam IDS jest narażony na ewentualne ataki. Oczywiście w takim przypadku śledzony jest ruch przepływające przez interfejsy sieciowe systemu i nie ma konieczności przełączania interfejsów w tryb promiskuistyczny. Podobnie można rozważać zainstalowanie sieciowego IDS (w tym PIDS/APIDS) na serwerze świadczącym usługi sieciowe.

Notatki

Programowy IDS a programowa zapora sieciowa

W przypadku stosowania programowego IDS w systemie operacyjnym, gdzie działa programowa zapora sieciowa, należy uwzględnić możliwe interakcje pomiędzy oboma podsystemami.

W przypadku systemu Linux oprogramowanie IDS korzysta z biblioteki *libpcap* do przechwytywania śledzonych pakietów, podobnie jak sniffery takie jak *tcpdump* czy *wireshark.* Biblioteka ta przechwytuje pakiety na styku pomiędzy warstwą łącza danych (np. Ethernet) a warstwą sieciową (IP). Oznacza to, że w przypadku pakietów odbieranych przez interfejsy sieciowe są one przechwytywane przez bibliotekę *libpcap* PRZED przekazaniem ich do filtra pakietów *netfilter,* a zatem ustawienia filtra pakietów nie wpływają na możliwość analizowania ruchu przez IDS. Inaczej jest natomiast w przypadku pakietów przeznaczonych do wysłania; jeżeli zostaną one odrzucone przez filtr pakietów, nie będą przekazane do warstwy łącza danych, a więc nie będą podlegać śledzeniu przez IDS.

Należy pamiętać, że powyższe stwierdzenia dotyczą przypadku, gdy jako filtr pakietów używany jest moduł *netfilter*, a IDS przechwytuje pakiety korzystając z biblioteki *libpcap*. W innych przypadkach interakcja między IDS a filtrem pakietów może przebiegać inaczej.

3 Sieciowy IDS Snort

Snort jest specjalizowanym, sieciowym systemem wykrywania zdarzeń. Jest to system bardzo popularny i w chwili obecnej nie mający równorzędnej konkurencji wśród wolnego oprogramowania. Początkowo stosowany wyłącznie jako samodzielny system IDS, obecnie może być również wdrażany jako jeden z elementów zintegrowanego systemu wykrywania zdarzeń, jakim jest np. Prelude IDS.

Pobieranie i instalacja Snort

Oprogramowanie Snort nie znajduje się w standardowych repozytoriach oprogramowania dystrybucji CentOS, natomiast ze strony domowej projektu Snort, http://snort.org, można pobrać pakiety RPM przeznaczony dla systemów Red Hat Enterprise Linux 6. Pakiety te bez żadnych problemów instalują się w systemie CentOS 6.

Instalacja minimalna wymaga zainstalowania samego pakietu *snort* oraz pakietu biblioteki *daq* (Data AcQusition library). Ze względu na zależności oba pakiety należy zainstalować jednocześnie (w jednej transakcji RPM) albo poczynając od pakietu *daq*.

```
# rpm -ihv daq-2.0.4.centos7.x86_64.rpm snort-2.9.7.0-1.centos7.x86_64.rpm
Przygotowywanie...                ################################# [100%]
Aktualizowanie/instalowanie...
   1:daq-2.0.4-1                  ################################# [ 50%]
   2:snort-1:2.9.7.0-1            ################################# [100%]
```

Listując pliki konfiguracyjne znajdujące się w pakiecie łatwo zauważyć, że zakładany jest jest katalog */etc/snort/rules*, natomiast nie są w nim umieszczone żadne pliki:

```
# rpm -ql snort|grep etc
/etc/logrotate.d/snort
/etc/rc.d/init.d/snortd
/etc/snort
/etc/snort/classification.config
/etc/snort/gen-msg.map
/etc/snort/reference.config
/etc/snort/rules
/etc/snort/snort.conf
/etc/snort/threshold.conf
/etc/snort/unicode.map
/etc/sysconfig/snort
```

Katalog */etc/snort/rules* jest kluczowy dla działania *Snort,* ponieważ znajdują się w nim pliki reguł. Reguły definiują zdarzenia wykrywane przez *Snort* i określają sposób ich raportowania.

Próba uruchomienia usługi *snortd* zdefiniowanej przez skrypt */etc/rc.d/init.d/snortd*, pozornie kończy się sukcesem:

```
# service snortd start
Starting snortd (via systemctl):                           [  OK  ]
```

Informacja ta jest jednak myląca: *Snort* natychmiast po uruchomieniu kończy działanie, niestety zwracając jako kod wyjścia wartość 0 co sugeruje poprawne wykonanie. O przyczynie niepowodzenia można przekonać się sprawdzając status usługi:

```
# service snortd status
snortd.service - SYSV: snort is a lightweight network intrusion detection tool that
currently detects more than 1100 host and network vulnerabilities, portscans, backdoors,
and more.
   Loaded: loaded (/etc/rc.d/init.d/snortd)
   Active: active (exited) since pon 2015-04-06 23:58:20 CEST; 31s ago
  Process: 18631 ExecStart=/etc/rc.d/init.d/snortd start (code=exited, status=0/SUCCESS)

kwi 06 23:58:20 c7test snort[18636]: [ 2123 2152 3386 ]
kwi 06 23:58:20 c7test snort[18636]: kwi 06 23:58:20 c7test snort[18636]: Detection:
kwi 06 23:58:20 c7test snort[18636]: Search-Method = AC-Full-Q
```

Notatki

```
kwi 06 23:58:20 c7test snort[18636]: Split Any/Any group = enabled
kwi 06 23:58:20 c7test snort[18636]: Search-Method-Optimizations = enabled
kwi 06 23:58:20 c7test snort[18636]: Maximum pattern length = 20
kwi 06 23:58:20 c7test snort[18636]: FATAL ERROR: /etc/snort/snort.conf(253...y.
kwi 06 23:58:20 c7test snortd[18631]: Starting snort: [NIEUDANE]
```

Komunikat wyjaśniający przyczynę błędu można także odnaleźć w systemowym dzienniku zdarzeń:

```
# grep snort.conf /var/log/messages
Apr  6 23:58:18 c7test snort[18636]: Parsing Rules file "/etc/snort/snort.conf"
Apr  6 23:58:20 c7test snort[18636]: FATAL ERROR: /etc/snort/snort.conf(253) Could not stat
dynamic module path "/usr/local/lib/snort_dynamicrules": No such file or directory.
```

Można także użyć kombinacji opcji -c i -T, aby przetestować poprawność konfiguracji:

```
# snort -c /etc/snort/snort.conf -T
Running in Test mode
...
ERROR: /etc/snort/snort.conf(253) Could not stat dynamic module path
"/usr/local/lib/snort_dynamicrules": No such file or directory.
```

Pobieranie reguł Snort

Reguły definiujące działanie *Snort* jako sieciowego IDS dystrybuowane są niezależnie od samego oprogramowania. Związany jest z tym pewien model biznesowy umożliwiający pozyskiwanie przez twórców *Snort* środków przeznaczonych na dalszy rozwój tego projektu.

Reguły *Snort* dostępne są do pobrania z witryny projektu.

Użytkownicy, którzy wykupili płatną subskrypcję, otrzymują nowe zestawy reguł natychmiast po ich utworzeniu, co z kolei ma miejsce po zidentyfikowaniu nowego rodzaju zagrożenia. Tzw. czas reakcji jest tu dość krótki i z reguły nie przekracza kilku dni, zespół *Snort* podaje jako przykład zidentyfikowanie luki w działaniu systemu zdalnego wywoływania procedur (RPC) w Windows – w chwili pojawienia się bardzo „popularnego" wirusa Blaster wykorzystującego tę lukę subskrybenci reguł *Snort* mieli już możliwość jego wykrywania.

Zarejestrowani użytkownicy witryny mają możliwość pobierania reguł po 30 dniach od ich udostępnienia subskrybentom. Rejestracja w witrynie jest bezpłatna. Licencja na uzyskane w ten sposób zestawy reguł NIE zezwala na ich rozpowszechnianie, natomiast zezwala na ich stosowanie w dowolnej ilości wdrożeń *Snort* administrowanych przez danego użytkownika, włącznie z przypadkiem gdy *Snort* jest zainstalowany na systemie nie należącym do użytkownika, lecz do jego pracodawcy. Umożliwia to bezpłatne wykorzystywanie *Snort* do celów profesjonalnych.

Reguły są rozpowszechniane w postaci archiwum tar.gz:

```
# tar -xf ../snortrules-snapshot-2970.tar.gz
# ls
etc  preproc_rules  rules  so_rules
```

Katalog *etc* zawiera pliki zaktualizowane konfiguracyjne *Snort*, katalog *rules* – pliki reguł, katalog *so_rules* – kod źródłowy opcjonalnych modułów dynamicznych *Snort* oraz reguły korzystające z tych modułów, *preproc_rules* – reguły odwołujące się bezpośrednio do mechanizmów poszczególnych preprocesorów. Aby uruchomić usługę *snortd* należy co najmniej skopiować reguły z katalogu *rules* do katalogu */etc/snort/rules*. Nie należy natomiast kopiować plików z katalogu *etc,* ponieważ zawarta tam konfiguracja nie jest zgodna z binarną dystrybucją *Snort* dla systemu CentOS. Ponadto w pliku konfiguracyjnym *snort.conf* należy upewnić się, że */etc/snort/rules* został wskazany jako katalog reguł:

```
var RULE_PATH /etc/snort/rules
```

Reguły dynamiczne

Oprócz klasycznych reguł przetwarzanych przez silnik reguł dystrybuowane są reguły dynamiczne, dostarczane w postaci bibliotek binarnych. Reguły te są de facto rozszerzeniem standardowego silnika reguł, co pozwala na zawarcie w nich dowolnie skomplikowanej logiki. W szczególności w takiej postaci dostarczane są reguły zbudowane w oparciu o informacje pozyskane od firm współpracujących z SourceFire, zawierające informacje których nie można opublikować w formie jawnej reguły tekstowej. Zalecane jest aktywowanie tych reguł. Niestety jak dotąd nie są dystrybuowane reguły dynamiczne dla wersji RHEL 7 / CentOS7, stąd ich konfiguracja została pominięta w niniejszym opracowaniu. Na potrzeby uruchomienia *Snort* wystarczy deaktywować odwołanie do nieistniejącego katalogu reguł dynamicznych:

```
# grep dynamicrules /etc/snort/snort.conf
# dynamicdetection directory /usr/local/lib/snort_dynamicrules
```

Reguły preprocesorów

Reguły preprocesorów odwołują się do anomalii mogących wystąpić w ramach analizy danych pod katem danego protokołu obsługiwanego przez preprocesor. Są one dostarczone w archiwum reguł i aby ich użyć wystarczy odpowiednio aktywować i dostosować konfigurację Snort:

```
# ls /etc/snort/rules/preproc_rules/
decoder.rules  preprocessor.rules  sensitive-data.rules
# grep PREPROC /etc/snort/snort.conf
```

```
var PREPROC_RULE_PATH /etc/snort/rules/preproc_rules
include $PREPROC_RULE_PATH/preprocessor.rules
include $PREPROC_RULE_PATH/decoder.rules
include $PREPROC_RULE_PATH/sensitive-data.rules
```

Plik sid-msg.map

Archiwum reguł zawiera w podkatalogu *etc* plik *sid-msg.map*. Plik ten zawiera opisy zdarzeń generowanych przez poszczególne reguły i jest niezbędny dla generowania tekstowych zapisów zdarzeń. Plik należy skopiować do katalogu */etc/snort*.

Reguły reputacji

Nowością wprowadzoną w Snort 2.9 jest preprocesor `reputation` kwalifikujący ruch jako zdarzenie niepożądane w oparciu o reputację źródła (adresu źródłowego). Póki co nie są dla niego dystrybuowane żadne reguły. Zaleca się wyłączenie tego mechanizmu w pliku *snort.conf*:

```
# Reputation preprocessor. For more information see README.reputation
#preprocessor reputation: \
#   memcap 500, \
#   priority whitelist, \
#   nested_ip inner, \
#   whitelist $WHITE_LIST_PATH/white_list.rules, \
#   blacklist $BLACK_LIST_PATH/black_list.rules
```

Reguły społecznościowe

Oprócz reguł dystrybuowanych przez SourceFire, istnieją także reguły opracowywane przez społeczność użytkowników Snort. W szczególności należy wymienić dwa serwisy dystrybuujące społecznościowe zestawy reguł: Bleeding Snort (http://www.bleedingsnort.com) oraz Emerging Threats (http://www.emergingthreats.net).

Oinkmaster

Specyfika działania Snort powoduje, iż dalece ważniejsze, niż aktualizowanie samego silnika Snort, jest aktualizowanie zestawu reguł. Skrypt Oinkmaster (http://oinkmaster.sourceforge.net) działaniem przypomina program *yum* i pozwala na automatyczną aktualizację reguł pobieranych ze wskazanych w konfiguracji źródeł. Aby zapewnić regularne aktualizacje, należy uruchamianie Oinkmaster skonfigurować jako zadanie usługi *cron*.

Uruchamianie Snort w trybie sniffera

W najprostszej konfiguracji *Snort* może zostać uruchomiony w trybie sniffera sieciowego. Uruchomienie takie polega na wydaniu polecenia `snort` zgodnie z następującą składnią:

```
snort -v [opcje] [filtr]
snort [-v] -l katalog [-b] [opcje] [filtr]
snort -r [opcje] [filtr]
```

Użycie wyłącznie opcji `-v` oznacza, że polecenie `snort` będzie wypisywać na ekranie podsłuchane pakiety.

Użycie opcji `-l` z podaniem nazwy katalogu oznacza, że pakiety będą zapisywane do plików logów; pliki te są tworzone w podanym katalogu. Użycie opcji `-b` oznacza, że pakiety zostaną zapisane w postaci binarnej, co znacznie przyspiesza proces zapisu, ale wymaga ponownej interpretacji tak utworzonych plików.

Użycie opcji `-r` oznacza odczyt pakietów z pliku binarnego i wyświetlenie ich na ekranie w formie tekstowej.

Filtr jest wyrażeniem w języku BPF (ang. Berkeley Packet Filter) opisującym pakiety, które mają zostać zarejestrowane. Jest to ten sam język, którego używa sniffer `tcpdump`. Brak podanego filtra oznacza przechwytywanie wszystkich pakietów.

Pozostałe opcje używane przez polecenie `snort` w trybie sniffera prezentuje poniższa tabela.

Tabela 1: Opcje polecenia snort w trybie sniffera

Opcja	*Znaczenie*
`-e`	Powoduje wypisywanie informacji dotyczących warstwy łącza danych.

Notatki

Opcja	*Znaczenie*
`-d`	Powoduje wypisywanie także zawartości pakietów w formie znakowej i szesnastkowej.
`-C`	Użycie opcji `-C` powoduje wypisywanie tylko postaci znakowej.
`-i interfejs`	Określa interfejs, na którym prowadzony jest nasłuch.
`-n liczba`	Określa liczbę pakietów, po której `snort` kończy działanie. Bez podania tego parametru `snort` działa do momentu zakończenia poprzez wciśnięcie klawiszy Ctrl+C.
`-p`	Powoduje, że `snort` NIE przełącza interfejsu w tryb promiskuistyczny (przełączenie w ten tryb jest działaniem domyślnym).
`-F plik`	Wczytuje filtr pakietów z podanego pliku.
`-K tryb` `-h maska_podsieci`	Powoduje zapisywanie pakietów w wybranym trybie. Dostępne tryby to: `pcap`, który jest domyślny i oznacza zapisanie pakietów w formacie binarnym identycznym z formatem używanym przez inne sniffery, jak `tcpdump` czy `wireshark`, oraz tryb `ascii`, w którym informacje o pakiecie są zapisywane w postaci tekstowej. Opcja ma znaczenie tylko jeśli jest dokonywany zapis pakietów do plików (opcja `-l`). W przypadku trybu `pcap` tworzony jest jeden plik logów, a jego nazwa jest ustalana na podstawie czasu uruchomienia sniffera. W przypadku trybu `ascii` tworzone są podkatalogi reprezentujące adresy IP, a w tych podkatalogach są tworzone pliki logów; w przypadku protokołów TCP i UDP - dla danego strumienia danych (na podstawie numerów portów), w przypadku pozostałych protokołów nazwa logu jest nazwą protokołu. Użycie opcji `-h` powoduje, że podana podsieć jest traktowana jako lokalna (w oryginale „domowa") i to adresy z tej podsieci są brane jako podstawa do tworzenia nazw podkatalogów.
`-O` `-h maska_podsieci`	Opcja `-O` powoduje, że w wypisywanych na ekranie pakietach zostaną zamaskowane (ang. *obfuscated*) adresy IP. Jest to przydatne, gdy uzyskany wydruk ma zostać opublikowany, a administrator nie chce ujawniać adresu. Użycie opcji `-h` powoduje, że podana podsieć jest traktowana jako lokalna (w oryginale „domowa") i to adresy z tej podsieci zostaną zamaskowane. Opcje te działają wyłącznie w trybie odczytu pliku binarnego (`-r`).

? *Domyślnie snort, podobnie jak tcpdump,na czas swojego działania przestawia interfejs w tryb promiskuistyczny. Ponieważ jednak dotyczy to warstwy łącza danych (odbierane są ramki o dowolnym adresie fizycznym), nie zaś warstwy sieciowej (biblioteka libpcap przechwytuje pakiety zanim nastąpi dopasowanie ich adresów do adresów interfejsów sieciowych w systemie), tryb promiskuistyczny nie jest wykazywany przez polecenie ip link. Komunikat o przełączeniu jest natomiast wysyłany do logów systemowych.*

Przykłady:

Wypisanie nagłówków pakietów:

```
# snort -n 1 -v
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Snort BPF option: icmp
pcap DAQ configured to passive.
The DAQ version does not support reload.
Acquiring network traffic from "eth0".
Decoding Ethernet

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.2.2 IPv6 GRE (Build 121)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort-team
           Copyright (C) 1998-2012 Sourcefire, Inc., et al.
           Using libpcap version 1.0.0
           Using PCRE version: 7.8 2008-09-05
           Using ZLIB version: 1.2.3

Commencing packet processing (pid=2490)
04/09-12:54:36.248691 10.0.2.15 -> 212.77.100.101
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:47881   Seq:1  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

===============================================================================
Run time for packet processing was 20.508971 seconds
Snort processed 1 packets.
```

```
Snort ran for 0 days 0 hours 0 minutes 20 seconds
   Pkts/sec:           0
===============================================================================
Packet I/O Totals:
   Received:           1
   Analyzed:           1 (100.000%)
    Dropped:           0 (  0.000%)
   Filtered:           0 (  0.000%)
Outstanding:           0 (  0.000%)
   Injected:           0
===============================================================================
Breakdown by protocol (includes rebuilt packets):
        Eth:           1 (100.000%)
       VLAN:           0 (  0.000%)
        IP4:           1 (100.000%)
       Frag:           0 (  0.000%)
       ICMP:           1 (100.000%)
        UDP:           0 (  0.000%)
        TCP:           0 (  0.000%)
        IP6:           0 (  0.000%)
    IP6 Ext:           0 (  0.000%)
   IP6 Opts:           0 (  0.000%)
      Frag6:           0 (  0.000%)
      ICMP6:           0 (  0.000%)
       UDP6:           0 (  0.000%)
       TCP6:           0 (  0.000%)
     Teredo:           0 (  0.000%)
    ICMP-IP:           0 (  0.000%)
      EAPOL:           0 (  0.000%)
    IP4/IP4:           0 (  0.000%)
    IP4/IP6:           0 (  0.000%)
    IP6/IP4:           0 (  0.000%)
    IP6/IP6:           0 (  0.000%)
        GRE:           0 (  0.000%)
    GRE Eth:           0 (  0.000%)
   GRE VLAN:           0 (  0.000%)
    GRE IP4:           0 (  0.000%)
    GRE IP6:           0 (  0.000%)
GRE IP6 Ext:           0 (  0.000%)
   GRE PPTP:           0 (  0.000%)
    GRE ARP:           0 (  0.000%)
    GRE IPX:           0 (  0.000%)
   GRE Loop:           0 (  0.000%)
       MPLS:           0 (  0.000%)
        ARP:           0 (  0.000%)
        IPX:           0 (  0.000%)
   Eth Loop:           0 (  0.000%)
   Eth Disc:           0 (  0.000%)
   IP4 Disc:           0 (  0.000%)
   IP6 Disc:           0 (  0.000%)
   TCP Disc:           0 (  0.000%)
   UDP Disc:           0 (  0.000%)
  ICMP Disc:           0 (  0.000%)
All Discard:           0 (  0.000%)
      Other:           0 (  0.000%)
Bad Chk Sum:           0 (  0.000%)
    Bad TTL:           0 (  0.000%)
     S5 G 1:           0 (  0.000%)
     S5 G 2:           0 (  0.000%)
      Total:           1
===============================================================================
```

Notatki

```
Snort exiting
```

Zwracają uwagę szczegółowe statystyki po zakończeniu wykonywania (w kolejnych przykładach zostaną one pominięte).

Rozszerzone wypisanie nagłówków pakietów ICMP:

```
# snort -n 1 -e -v icmp
...
04/09-12:58:08.156622 08:00:27:02:4A:9E -> 52:54:00:12:35:02 type:0x800 len:0x62
10.0.2.15 -> 212.77.100.101 ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:49161   Seq:1  ECHO
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
...
```

Wypisanie pakietów UDP DNS wraz z zawartością w postaci tekstowej:

```
# snort -n 2 -dveC udp port 53
...
04/09-13:13:57.694606 08:00:27:02:4A:9E -> 52:54:00:12:35:02 type:0x800 len:0x4D
10.0.2.15:36856 -> 217.76.116.201:53 UDP TTL:64 TOS:0x0 ID:52710 IpLen:20 DgmLen:63 DF
Len: 35
.............www.zsk.p.lodz.pl.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

04/09-13:13:57.778907 52:54:00:12:35:02 -> 08:00:27:02:4A:9E type:0x800 len:0x70
217.76.116.201:53 -> 10.0.2.15:36856 UDP TTL:64 TOS:0x0 ID:33195 IpLen:20 DgmLen:98
Len: 70
.............www.zsk.p.lodz.pl..............<...zskl.../.......<
...3..
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
...
```

Zapisanie pakietów do pliku z późniejszym odczytem:

```
# snort -n 4 -l .
Running in packet logging mode
Log directory = .
...

===============================================================================
Snort exiting
# ls
snort.log.1333970182
# snort -vdeC -r snort.log.1197538673
Running in packet dump mode
...
04/09-13:16:30.576425 08:00:27:02:4A:9E -> 52:54:00:12:35:02 type:0x800 len:0x5A
10.0.2.15:123 -> 80.50.231.226:123 UDP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:76 DF
Len: 48
.............STEP.........-CL.....-CL...(.-C.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

04/09-13:16:30.584086 52:54:00:12:35:02 -> 08:00:27:02:4A:9E type:0x800 len:0x5A
80.50.231.226:123 -> 10.0.2.15:123 UDP TTL:64 TOS:0x0 ID:33203 IpLen:20 DgmLen:76
Len: 48
$..........6ATOM.-Cw...A.-C......-C...:O.-C.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

04/09-13:16:45.715541 08:00:27:02:4A:9E -> 52:54:00:12:35:02 type:0x800 len:0x49
10.0.2.15:56557 -> 217.76.116.201:53 UDP TTL:64 TOS:0x0 ID:24122 IpLen:20 DgmLen:59 DF
Len: 31
s7..........www.p.lodz.pl.....
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

04/09-13:16:45.785888 52:54:00:12:35:02 -> 08:00:27:02:4A:9E type:0x800 len:0x6C
217.76.116.201:53 -> 10.0.2.15:56557 UDP TTL:64 TOS:0x0 ID:33204 IpLen:20 DgmLen:94
Len: 66
s7..........www.p.lodz.pl.................web0...+..........3
.H
...
```

Odczyt z pliku z zamaskowaniem adresów IP:

```
# snort -vO -r snort.log.1197538673
...
04/09-13:16:30.576425 xxx.xxx.xxx.xxx:123 -> xxx.xxx.xxx.xxx:123
UDP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:76 DF
Len: 48
```

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
...
```

Plik binarny tworzony przez polecenie `snort` ma format identyczny z plikami tworzonymi przez polecenie `tcpdump`. Można więc dokonać analizy tego pliku poleceniem `tcpdump`:

```
# tcpdump -vvv -r snort.log.1333970182
reading from file snort.log.1333970182, link-type EN10MB (Ethernet)
13:16:30.576425 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 76)
    10.0.2.15.ntp > ntp1.tp.pl.ntp: [bad udp cksum 43f0!] NTPv4, length 48
       Client, Leap indicator: clock unsynchronized (192), Stratum 0 (unspecified), poll 6s,
precision -22
       Root Delay: 0.000000, Root dispersion: 0.002456, Reference-ID: (unspec)
         Reference Timestamp:  0.000000000
         Originator Timestamp: 3542958924.569942525 (2012/04/09 13:15:24)
         Receive Timestamp:    3542958924.592065939 (2012/04/09 13:15:24)
         Transmit Timestamp:   3542958990.576379702 (2012/04/09 13:16:30)
           Originator - Receive Timestamp:  +0.022123413
           Originator - Transmit Timestamp: +66.006437176
...
```

Zapis pakietów w trybie `ascii`:

```
# snort -l . -K ascii -i eth0 -n 10
Running in packet logging mode
...
# ls
10.0.2.15
# ls 10.0.2.15/
TCP:40471-80  UDP:58752-53
# cat 10.0.2.15/TCP\:40471-80
04/09-13:23:12.387868 10.0.2.15:40471 -> 23.21.248.121:80
TCP TTL:64 TOS:0x0 ID:48720 IpLen:20 DgmLen:60 DF
******S* Seq: 0xD5CAFAFB  Ack: 0x0  Win: 0x3908  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 476331 0 NOP WS: 5
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

04/09-13:23:12.503944 23.21.248.121:80 -> 10.0.2.15:40471
TCP TTL:64 TOS:0x0 ID:33466 IpLen:20 DgmLen:44
***A**S* Seq: 0x1ADB0001  Ack: 0xD5CAFAFC  Win: 0xFFFF  TcpLen: 24
TCP Options (1) => MSS: 1460
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
...
```

Notatki

4 Konfiguracja Snort w trybie IDS

Mechanizmy analizy danych

Dane sieciowe analizowane przez Snort przechodzą przez trzy zasadnicze mechanizmy analizy:

- dekoder
- preprocesory
- detektor

Dekoder jest mechanizmem składania pakietów warstwy sieciowej i z danych dostarczonych przez warstwę łącza danych.

Preprocesory dokonują wyspecjalizowanej dalszej analizy danych, z reguły pod kątem protokołu warstwy transportowej (np. składanie strumienia TCP) lub warstwy aplikacji (np. analiza protokołu HTTP). W odróżnieniu od dekodera preprocesory mogą korzystać z zapamiętanych wcześniej pakietów.

Detektor jest mechanizmem, który analizuje pakiety zgodnie z parametrami opisanymi w regułach *Snort*. Analiza ta jest zasadniczo dwojakiego rodzaju: bazująca na zawartości nagłówka (niekoniecznie warstwy sieciowej) lub bazująca na zawartości danych (ang. *content matching*).

Każdy z tych mechanizmów generuje dwa rodzaje danych: ostrzeżenia (*alert*) o zdarzeniach uznanych za potencjalne niebezpieczeństwo oraz logi pakietów, które wywołały ostrzeżenia (poza niektórymi przypadkami, kiedy nie można wyróżnić konkretnego pakietu świadczącego o zaistnieniu zdarzenia). Każdy z tych zapisów może podlegać oddzielnej konfiguracji. Zidentyfikowanie mechanizmu, który wygenerował ostrzeżenie jest możliwe, ponieważ ostrzeżeniu jest przypisany identyfikator generatora (GID, szczegóły w rozdziale poświęconym mechanizmom zapisywania ostrzeżeń).

Dostrajanie konfiguracji domyślnej

Po zainstalowaniu pakietu *Snort* i skopiowaniu pobranych reguł otrzymuje się domyślną konfigurację *Snort*, która w prostych zastosowaniach może być wystarczająca. Należy jednak pamiętać o właściwym dostrojeniu domyślnej konfiguracji, co uzyskuje się poprzez ustawienie wartości standardowych zmiennych w niej występujących.

? *Szczegółowy opis składni zmiennych zawarty jest w rozdziale poświęconym regułom.*

Tabela 2: Zmienne w konfiguracji domyślnej Snort

Zmienna	*Znaczenie*
`HOME_NET`	Określa podsieć IP, która jest traktowana jako sieć lokalna. Najprostsze postaci określania posieci to: - adres podsieci w formacie CIDR, np. `var HOME_NET 10.3.0.0/16` - adres podsieci przypisany do interfejsu, np. `var HOME_NET $eth0_ADDRESS` - dowolny adres: `var HOME_NET any`
`EXTERNAL_NET`	Określa podsieć IP, która jest traktowana jako sieć "zewnętrzna". Często wystarczająco dobre jest określenie: `var EXTERNAL_NET any`

`HTTP_SERVERS`
`SMTP_SERVERS`
`DNS_SERVERS`
`SQL_SERVERS`
`TELNET_SERVERS`
`SNMP_SERVERS`
	Zmienne te określają adresy serwerów odpowiednich usług. Ich prawidłowe określenie jest o tyle istotne, że pojawienie się np. żądania HTTP do serwera w sieci lokalnej nie znajdującego się na liście `HTTP_SERVERS` jest podstawą do wygenerowania ostrzeżenia.
	Zmienne te są tego samego formatu co `HOME_NET` i `EXTERNAL_NET`, jednak w ich przypadku bardziej przydatna jest możliwość wymienienia wielu adresów IP, co jest realizowane w postaci listy oddzielanej przecinkami, zawartej w nawiasach kwadratowych, np: `var HTTP_SERVERS [172.16.0.3,172.16.0.11]`

`HTTP_PORTS`	Określa porty, na których działają serwery HTTP w sieci lokalnej, najczęściej w postaci listy numerów portów, np. `portvar HTTP_PORTS [80,8080,8088]` **UWAGA:** Snort nie jest w stanie analizować danych transmitowanych protokołami szyfrowanymi. Z tego względu na powyższej liście nie należy uwzględniać portów, na których działa protokół HTTPS.
`RULE_PATH`	Określa katalog, w którym znajdują się pliki reguł. Domyślnie zmienna ta jest używana przez dyrektywę `include` i zdefiniowana następująco: `var RULE_PATH /etc/snort/rules`

Reguły dodatkowe

Nie wszystkie pliki reguł dostarczane w pakiecie standardowym są wykorzystywane w domyślnej konfiguracji. Dyrektywy `include` dla tych plików istnieją wprawdzie w domyślnej postaci pliku *snort.conf* (pod jego koniec), ale są

zdezaktywowane poprzez opatrzenie znakiem komentarza `#`. Aby użyć danego pliku reguł, wystarczy usunąć znak komentarza.

Użycie tych reguł pozostawiono do decyzji użytkownika, gdyż są one albo w fazie eksperymentalnej i mogą generować duże ilości nieuzasadnionych ostrzeżeń, albo wykonują skomplikowane przeszukiwania na dużej ilości danych. Tym niemniej często zawierają one bardzo ciekawą funkcjonalność, np. wykrywanie przesyłania siecią kodu bardziej znanych wirusów. Przed podjęciem decyzji o użyciu danego pliku reguł należy zapoznać się z dokumentacją zawartą w samym pliku.

Opcje uruchomienia i ogólne dyrektywy konfiguracyjne

Uruchomienie *Snort* w trybie IDS polega na wydaniu polecenia:

```
snort -c plik_konfiguracyjny [opcje] [filtr]
```

Wyróżnikiem tego rodzaju uruchomienia jest użycie opcji `-c`, wskazującej położenie pliku, w którym zapisana jest konfiguracja *Snort* jako systemu IDS (domyślnie */etc/snort/snort.conf*).

Konfiguracja *Snort* może zostać zapisana w wielu plikach, dzięki czemu staje się czytelniejsza i łatwiejsza w zarządzaniu (dotyczy to zwłaszcza plików zawierających reguły). Łączenie plików umożliwia dyrektywa `include`, która powoduje włączenie w miejscu jej wystąpienia podanego pliku.

Ogólne dyrektywy konfiguracyjne są określane w pliku konfiguracyjnym poprzez dyrektywę config o składni:

```
config nazwa_opcji [:wartość]
```

Przykłady:

```
config verbose
config set_uid: snort
config ignore_ports: 80 443
```

Większość dyrektyw ogólnych może być ustawiana także jako opcja uruchomienia polecenia `snort`.

W przypadku użycia jednocześnie opcji uruchomienia i dyrektywy konfiguracyjnej pierwszeństwo mają zawsze opcje uruchomienia.

Każda linia pliku konfiguracyjnego Snort i plików dołączanych powinna być pusta lub zawierać komentarz lub prawidłową dyrektywę konfiguracyjną / regułę. W szczególności linie zawierające tylko znaki odstępu powodują błąd naruszenia ochrony pamięci przy uruchamianiu Snort. W przypadku błędnych zapisów komunikaty o błędach wysyłane są do logów systemowych.

Tabela 3: Ważniejsze opcje i dyrektywy konfiguracyjne polecenia snort w trybie IDS

Dyrektywa i/lub opcja	*Znaczenie*
Ogólne	
`daemon / -D`	Powoduje uruchomienie procesu `snort` jako daemona. **UWAGA**: w tym trybie zarówno samo polecenie, jak i wszystkie katalogi i pliki ustawiane w konfiguracji lub za pomocą opcji muszą być podawane w postaci ścieżek bezwzględnych.
`-T`	Wraz z opcją `-c` pozwala na przetestowanie poprawności konfiguracji.
`chroot: katalog` `-t katalog`	Powoduje wykonanie operacji `chroot` do podanego katalogu. Wszelkie ścieżki ustawione w konfiguracji lub za pomocą innych opcji będą interpretowane względem tego katalogu.

`set_uid: użytkownik`
`-u użytkownik`
`set_gid: grupa`
`-g grupa`
	Powodują, że po zainicjowaniu proces `snort` zmieni swoje EUID / EGID na podanego użytkownika / grupę (inaczej mówiąc – zacznie działać z uprawnieniami danego użytkownika / grupy).

Akwizycja danych	
`no_promisc / -p`	Powoduje, że `snort` NIE przełącza interfejsu w tryb promiskuistyczny (przełączenie w ten tryb jest działaniem domyślnym).

Notatki

Dyrektywa i/lub opcja	*Znaczenie*
`interface: interfejs` `-i interfejs`	Określa interfejs, na którym prowadzony jest nasłuch.
`bpf_file: plik` `-F plik`	Wskazuje plik, w którym określony jest filtr wyznaczający pakiety poddawane analizie.
`pkt_count: ilość` `-n ilość`	Określa ilość pakietów, po przetworzeniu których *Snort* zakończy działanie.

`ignore_ports: protokół lista_portów`	Powoduje, że ignorowane są pakiety TCP lub UDP, które posiadają port źródłowy lub docelowy wymieniony na liście. Lista portów składa się z elementów oddzielanych spacjami, każdy element to pojedynczy port lub zakres portów wyrażony dwukropkiem.
	Przykład:
	`config ignore_ports: udp 53 67:68`
	`config ignore_ports: tcp 80 443 8080 8443`

Ostrzeżenia i logi	
`-A tryb`	Ustawia tryb zapisywania ostrzeżeń. Domyślny tryb to `full` będący odpowiednikiem dyrektywy `alert_full`. Inne często używane tryby to `fast` – odpowiednik dyrektywy `alert_fast` oraz `none` – wyłącza generowanie ostrzeżeń. **Zobacz też:** podrozdział „Zachowywanie ostrzeżeń i logów”.
`alert_with_interface_name` `-I`	Powoduje dołączenie do ostrzeżenia nazwy interfejsu, z którego odebrany został pakiet, który wywołał ostrzeżenie.
`-b` `-L plik`	Powoduje zapisywanie pakietów do logów w postaci binarnej zamiast tekstowej analogicznie jak w przypadku trybu sniffera. Jest to najbardziej wydajny sposób zapisu pakietów. Domyślnie nazwa pliku logów to *czas_utworzenia.snort.log*, gdzie *czas_utworzenia* to czas systemowy wyrażony w sekundach od 1 stycznia 1970. Nazwę pliku logów można ustawić za pomocą opcji `-L`.
`nolog / -N`	Wyłącza zapisywanie pakietów (ostrzeżenia są nadal zapisywane)
`logdir katalog / -l katalog`	Wskazuje katalog, w którym będą zapisywane pliki ostrzeżeń i pliki logów pakietów (domyślnie */var/log/snort*).
`alertfile: plik`	Określa (względem katalogu określonego opcją `logdir`) nazwę pliku, do którego będą zapisywane ostrzeżenia (domyślnie: *alert*).
`verbose / -v`	Powoduje, że pakiety oprócz zapisywania do logów są wypisywane na konsoli, analogicznie jak w przypadku trybu sniffera. Ze względu na niewielką wydajność wypisywania zaleca się co najwyżej używania tej opcji dla testowania działania *Snort*.

`dump_payload / -d`
`decode_data_link / -e`
`dump_chars_only / -C`
	Powodują umieszczanie w logach pakietów informacji o danych transportowanych przez pakiet i informacji z warstwy łącza danych, analogicznie jak w przypadku trybu sniffera. Stosując te opcje należy mieć na uwadze, że większa ilość danych dla każdego pakietu obniża wydajność zapisywania.

`obfuscate / -O`
`reference_net`
`maska_podsieci`
`-h maska_podsieci`
	Opcja `-O` powoduje, że w zapisywanych do logu pakietach zostaną zamaskowane (ang. *obfuscated*) adresy IP. Jest to przydatne, gdy uzyskany wydruk ma zostać opublikowany, a administrator nie chce ujawniać adresu. Użycie opcji `-h` lub dyrektywy `reference_net` powoduje, że podana podsieć jest traktowana jako lokalna (w oryginale „domowa”) i to adresy z tej podsieci zostaną zamaskowane.

Analiza pakietów	

`default_rule_state`
`order`
`--alert-before-pass`
`--process-all-events`
	Te dyrektywy i opcje określają sposób interpretowania reguł *Snort* przez detektor. Ich szczegółowy opis znajduje się w rozdziale poświęconym regułom.

Usługa snortd

Zainstalowanie *Snort* z pakietu RPM powoduje utworzenie skryptu */etc/rc.d/init.d/snortd*, który odpowiedzialny jest za realizację usługi *snortd.* Ciekawy jest sposób uruchamiania *Snort* przez skrypt usługi; bazuje on na dyrektywach zawartych w pliku */etc/sysconfig/snort*, na ich podstawie przekazując do polecenia `snort` odpowiednie opcje uruchomienia.

Powyższe oznacza, że konfiguracja zawarta w pliku /etc/sysconfig/snort MA PIERWSZEŃSTWO nad konfiguracją zawartą w pliku /etc/snort/snort.conf

? *Skrypt usługi snortd zawsze uruchamia polecenie snort w trybie daemona, stosując opcję -D.*

Tabela 4: Ważniejsze dyrektywy konfiguracyjne usługi snortd

Opcja	*Znaczenie*
`CONF=plik`	Określa położenie pliku konfiguracyjnego *Snort* (opcja `-c`)
`INTERFACE=lista`	Określa listę interfejsów, na których *Snort* ma prowadzić nasłuch. Lista może przyjąć jedną z następujących postaci: - pojedynczy interfejs, np. `INTERFACE=eth0` – podana nazwa interfejsu jest przekazywana do polecenia `snort` za pomocą opcji `-i`; - wiele interfejsów, np. `INTERFACE="eth0 eth1"` - polecenie `snort` jest uruchamiane z opcją `-i` osobno dla każdego interfejsu; - `INTERFACE=ALL` - polecenie snort jest uruchamiane z opcją `-i` osobno dla każdego interfejsu, lista interfejsów jest pobierana z pliku */proc/net/dev*, przy czym brane pod uwagę są tylko interfejsy zawierające w nazwie ciąg znaków „eth”.
`USER=nazwa` `GROUP=nazwa`	Powodują, że po zainicjowaniu proces `snort` zmieni swoje EUID / EGID na podanego użytkownika / grupę (opcje `-u` i `-g`). Jeżeli dyrektywa nie zostanie użyta, wartości domyślne to `snort` i `snort`.
`LOGDIR=katalog`	Wskazuje katalog, w którym będą zapisywane pliki ostrzeżeń i pliki logów pakietów (opcja `-l`)
`ALERTMODE=tryb`	Określa tryb zapisywania ostrzeżeń (opcja `-A`). **Zobacz też:** podrozdział „Zachowywanie ostrzeżeń i logów”.
`DUMP_APP=0 | 1`	Włącza / wyłącza zapisywanie segmentu danych pakietu (opcja `-d`)
`BINARY_LOG=0 | 1`	Włącza / wyłącza zapisywanie logów pakietów w postaci binarnej (opcja `-b`)
`NO_PACKET_LOG=0 | 1`	Włącza / wyłącza zapisywanie logów pakietów (opcja `-N`)
`PRINT_INTERFACE=0 | 1`	Włącza / wyłącza dopisywanie do ostrzeżeń nazwy interfejsu (opcja `-I`)
`BPF=filtr` lub `BPFFILE=plik`	Określa treść filtra / nazwę pliku zawierającego treść filtra (opcje używane zamiennie) kwalifikującego pakiety do analizy przez *Snort*.

! *Domyślną wartością zmiennej INTERFACE jest eth0, co wobec stosowanego od wersji 7 systemu nazewnictwa interfejsów sieciowych nigdy nie jest spełnione. W takiej sytuacji Snort nie uruchamia się poprawnie nie generując przy tym żadnych komunikatów.*

! *Skrypt usługi snortd nie przewiduje stosowania wszystkich opcji uruchomienia polecenia snort. Ponieważ jednak wartość opcji BPF jest przekazywana bezpośrednio jako parametr polecenia, można wstawić potrzebne opcje przed lub zamiast treści filtra pakietów, np. BPF="--alert-before-pass udp port 53"*

Przykład:

```
# grep INTERFACE /etc/sysconfig/snort
INTERFACE=enp0s9
...
# service snortd start
Starting snort:                                            [  OK  ]
# ps aux|grep snort
snort    21169  0.0 26.9 619300 274588 ?       Ssl  02:19   0:00 /usr/sbin/snort -A fast -b
-d -D -i enp0s9 -u snort -g snort -c /etc/snort/snort.conf -l /var/log/snort
...
# grep INTERFACE /etc/sysconfig/snort
INTERFACE="enp0s3 enp0s9"
...
# ps aux|grep snort
snort    21308  0.1 26.8 619184 273252 ?       Ssl  02:22   0:00 /usr/sbin/snort -A fast -b
-d -D -i enp0s3 -u snort -g snort -c /etc/snort/snort.conf -l /var/log/snort/enp0s3
snort    21323  0.5 26.9 619380 274288 ?       Ssl  02:23   0:00 /usr/sbin/snort -A fast -b
```

Notatki

```
-d -D -i enp0s9 -u snort -g snort -c /etc/snort/snort.conf -l /var/log/snort/enp0s9
```

Moduły dynamiczne

Architektura programowa *Snort* umożliwia dołączanie do niego podczas uruchamiania dynamicznych modułów rozszerzających jego funkcjonalność. Istnieją trzy rodzaje takich modułów:

- moduły preprocesorów – zawierające dodatkowe preprocesory. W standardowym pakiecie Snort zawartych jest kilka dynamicznych preprocesorów umieszczonych w katalogu */usr/lib64/snort-2.9.x.y_dynamicpreprocessor*, pozostałe używane preprocesory są wkompilowane w program `snort`;
- moduły detektora, w oryginale *engine modules* – zawierające dodatkowe funkcje detektora. W standardowym pakiecie Snort w katalogu */usr/lib64/snort-2.9.x.y_dynamicengine* znajduje się moduł *libsfengine.so*, będący częścią mechanizmu detektora;
- moduły reguł, zawierające reguły *Snort* zapisane nie w postaci jawnej, ale jako struktury danych wewnętrznie używane przez *Snort*. W standardowym pakiecie *Snort* nie ma reguł dynamicznych, ale mogą zostać doinstalowane.

Moduły dynamiczne, które mają zostać załadowane przy uruchamianiu programu `snort`, są wskazywane stosownymi dyrektywami lub opcjami uruchomienia:

Dyrektywa lub opcja	*Znaczenie*
`dynamicpreprocessor plik` `--dynamic-preprocessor-lib plik`	Wskazuje plik dynamicznego modułu preprocesora.
`dynamicpreprocessor directory katalog` `--dynamic-preprocessor-lib-dir katalog`	Wskazuje katalog, w którym znajdują się dynamiczne moduły preprocesorów (załadowane zostaną wszystkie moduły).
`dynamicengine plik` `--dynamic-engine-lib plik`	Jak powyżej, w odniesieniu do dynamicznych modułów detektora.
`dynamicengine directory katalog` `--dynamic-engine-lib-dir katalog`	
`dynamicdetection plik` `--dynamic-detection-lib plik`	Jak powyżej, w odniesieniu do dynamicznych modułów reguł.
`dynamicdetection directory katalog` `--dynamic-detection-lib-dir katalog`	

W architekturze Snort istnieją również wtyczki (ang. plugin), które jednakże nie są tym samym co moduły dynamiczne (wtyczki są włączane do Snort na etapie kompilacji).

Zachowywanie ostrzeżeń i logów

Kolejka zdarzeń dla pakietu

Pojedynczy pakiet, na skutek interpretacji przez wiele elementów architektury *Snort,* może spowodować wygenerowanie wielu ostrzeżeń o zdarzeniach. Dyrektywa

```
config event_queue: max_queue mq log ml events_order tryb
```

ustala maksymalną ilość ostrzeżeń, jakie będą zapamiętane dla danego pakietu (`mq`, domyślnie 8) oraz maksymalną ilość ostrzeżeń, jakie będą zapisane (`ml`, domyślnie 3). Ostrzeżenia do zapisania mogą być wybrane w jednym z `trybów`. Domyślny tryb `content_length` zakłada, że ostrzeżenia wygenerowane przez reguły mają pierwszeństwo nad tymi wygenerowanymi przez dekoder i preprocesory, zaś w ramach ostrzeżeń wygenerowanych przez reguły pierwszeństwo mają te, których reguły operują na większym zakresie danych pakietu (uznawane za dokładniejsze). Alternatywnym trybem jest `priority`, który szereguje ostrzeżenia na podstawie zapisanych w nich informacji o priorytecie ostrzeżenia.

Zapis do plików tekstowych

Standardowym sposobem zachowywania przez Snort ostrzeżeń oraz logów pakietów jest zapisywanie ich w plikach, standardowo leżących w katalogu */var/log/snort*. To właśnie do tej metody zachowywania odnoszą się opcje opisywane w rozdziałach poświęconych ogólnym opcjom konfiguracyjnym. Oprócz tego, do programu *Snort* włączonych jest wiele wtyczek umożliwiających zachowywanie ostrzeżeń i logów na inne sposoby; wtyczki te konfiguruje się używając odpowiednich dyrektyw konfiguracyjnych.

Ze względu na pierwszeństwo opcji uruchomienia nad dyrektywami konfiguracyjnymi, uruchomienie polecenia snort z opcją -A powoduje, że używana będzie tylko standardowa metoda zapisywania do plików, bez względu na dyrektywy konfiguracyjne.
W konsekwencji, ustawienie jakiejkolwiek wartości opcji ALERTMODE w pliku konfiguracyjnym usługi snortd (/etc/sysconfig/snort) powoduje, że działać będzie WYŁĄCZNIE zapisywanie do plików standardowych.

W przypadku stosowania standardowego zapisywania ostrzeżeń do plików *Snort* oferuje dwa tryby zapisywania ostrzeżeń: krótki i pełny.

Ostrzeżenia krótkie

Tryb ostrzeżeń krótkich jest włączany poprzez użycie dyrektywy konfiguracyjnej:

```
output alert_fast: plik
```

lub poprzez użycie opcji `-A fast` (lub ustawienie opcji `ALERTMODE=fast` w pliku konfiguracyjnym usługi `snortd`, wówczas użyta zostanie nazwa pliku `alert`) .

Ostrzeżenia w trybie krótkim są zapisywane w postaci jednej linii zawierającej podstawowe informacje o rodzaju zdarzenia i o pakiecie, który wywołał ostrzeżenie.

Przykład:

```
04/10-12:18:18.908906  [**] [3:17775:2] SHELLCODE Shikata Ga Nai x86 polymorphic shellcode
decoder detected [**] [Classification: Executable code was detected] [Priority: 1] {TCP}
11.0.0.11:48961 -> 11.0.0.1:80
```

Pierwsza informacja w linii to **data i czas wystąpienia zdarzenia**. Następnie podawany jest liczbowy **identyfikator**, na który składają się: identyfikator generatora GID, identyfikator zdarzenia SID i identyfikator wersji.

Identyfikator generatora wskazuje na mechanizm *Snort,* który wykrył zdarzenie. Wartości identyfikujące poszczególne mechanizmy zapisane są w pliku */etc/snort/gen-msg.map*; wartość 1 jest zarezerwowana dla zdarzeń wykrytych przez reguły (faza detektora), wartość 3 dla reguł dynamicznych, wartość 116 – dla fazy dekodera.

Identyfikator zdarzenia jest przypisywany zgodnie z numeracją obowiązującą w ramach danego mechanizmu, tak więc zdarzenie jest jednoznacznie identyfikowane przez parę GID/SID. W przypadku reguł SID jest nadawany wprost jako opcja reguły, zaś w przypadku preprocesorów SID jest nadawany przez preprocesor, wartości SID dla danego preprocesora należy poszukiwać w dokumentacji tego preprocesora lub w pliku :

```
# grep 122 /etc/snort/gen-msg.map
122 || 1 || portscan: TCP Portscan
...
122 || 5 || portscan: TCP Filtered Portscan
...
122 || 27 || portscan: Open Port
```

Identyfikator wersji jest używany, jeżeli reguła wykrywająca dane zdarzenie jest modyfikowana.

Kolejną informacją jest **ogólna klasyfikacja typu zdarzenia**. Informacja ta dotyczy zdarzeń wykrytych przez reguły i sposób jej określania jest opisany w rozdziale poświęconym regułom.

Priorytet zdarzenia jest ustalany przez mechanizm, który wykrył zdarzenie. W przypadku reguł priorytet może być ustalony jako parametr reguły lub może być wynikiem wspomnianej powyżej klasyfikacji. Najważniejsze zdarzenia mają priorytet 0, większy priorytet oznacza mniej ważne zdarzenie.

Ostatnią częścią ostrzeżenia są **informacje o pakiecie**, który spowodował wykrycie zdarzenia. W tym trybie podawany jest protokół, adres źródłowy i docelowy oraz ewentualnie porty protokołu TCP lub UDP.

Ostrzeżenia pełne

Tryb ostrzeżeń pełnych jest włączany poprzez użycie dyrektywy konfiguracyjnej:

```
output alert_full: plik
```

lub poprzez użycie opcji `-A full` (lub ustawienie opcji `ALERTMODE=full` w pliku konfiguracyjnym usługi `snortd`, wówczas użyta zostanie nazwa pliku `alert`).

W stosunku do trybu krótkiego ostrzeżenia są tu rozszerzone o szczegółowe informacje o nagłówku pakietu. Jeżeli ostrzeżenie jest wygenerowane przez regułę, która korzysta z mechanizmu referencji, ostrzeżenie zawiera również referencje do baz danych zawierających informacje o typach ataków.

Przykład:

```
[**] [1:17597:1] WEB-PHP TikiWiki jhot.php script file upload attempt [**]
[Classification: Attempted User Privilege Gain] [Priority: 1]
04/10-13:29:26.385469 11.0.0.11:57237 -> 11.0.0.1:80
TCP TTL:64 TOS:0x0 ID:61761 IpLen:20 DgmLen:1175 DF
***A**** Seq: 0x7AFC92BF  Ack: 0x4F1D72A5  Win: 0x4160  TcpLen: 32
[Xref => http://tikiwiki.org/tiki-read_article.php?articleid=136][Xref =>
http://www.securityfocus.com/bid/19819]
```

Notatki

Stosowanie ostrzeżeń pełnych jest wygodniejsze z punktu widzenia ich analizy, ale powoduje poważne obniżenie wydajności działania *Snort*. Wyjściem umożliwiającym zachowanie zarówno dokładnej analizy, jak i wydajności, jest zapisywanie ostrzeżeń do bazy danych lub pliku binarnego w tzw. formacie zunifikowanym i użycie wyspecjalizowanej aplikacji do późniejszej analizy.

? *Zobacz też: rozdział poświęcony analizie ostrzeżeń*

Binarne logowanie pakietów

Dane pakietów mogą, zamiast w postaci tekstowej, być zapisywane w postaci binarnej używanej przez sniffery takie jak `tcpdump` (tzw. format *pcap*). Jest to rozwiązanie znacznie zwiększające wydajność działania *Snort*.

W pliku konfiguracyjnym *Snort* włączenie binarnego zapisywania pakietów odbywa się poprzez użycie dyrektywy:

```
output log_tcpdump: plik
```

Analogiczny efekt daje użycie opcji uruchomienia polecenia `snort -b`.

Zapis binarny (zunifikowany)

Najbardziej wydajnym sposobem zachowywania zarówno ostrzeżeń *Snort*, jak i danych pakietów jest ich zapis w formacie binarnym, w oryginale nazywanym zunifikowanym (*unified*). Powstały w ten sposób plik ostrzeżeń nie może być odczytany bezpośrednio, a tylko poprzez użycie wyspecjalizowanej aplikacji; przykładem jest tu opisany w dalszej części rozdziału analizator *barnyard.*

Zapis binarny jest włączany w pliku konfiguracyjnym poprzez użycie dyrektyw:

```
output alert_unified2: plik [, limit]
output log_unified2: plik [, limit]
```

odpowiednio dla zapisu ostrzeżeń i logów pakietów. Opcjonalny limit podawany jest w megabajtach i wyznacza maksymalny rozmiar pliku. Do podanej nazwy pliku dołączany jest czas utworzenia.

Alternatywnie można użyć dyrektywy `output_unified2` aby zapisywać jednolity plik zawierający oba rodzaje informacji.

Zapis w formacie Prelude (IDMEF)

Ostrzeżenia mogą być zapisywane do bazy ostrzeżeń hybrydowego IDS *Prelude*, w formacie IDMEF. Do używania tego rodzaju zapisu niezbędne jest skonfigurowanie i uruchomienie *Prelude*. Zapis w tej postaci jest włączany w pliku konfiguracyjnym poprzez użycie dyrektywy:

```
output alert_prelude: profile=nazwa_profilu_prelude [info=liczba low=liczba medium=liczba high=liczba]
```

`Nazwa_profilu_prelude` jest nazwą profilu zdefiniowanego w konfiguracji *Prelude* do współpracy ze *Snort*. Opcjonalnie można ustalić, jakie wartości priorytetów liczbowych używane przez *Snort* odpowiadają jakim poziomom ważności w *Prelude*; domyślne powiązania to: `info=4`, `medium=3`, `low=2`, `high=1` i mniej.

! *Używanie przez Snort bazy danych wymaga uprzedniego przygotowania bazy, tzn. poza stworzeniem samej bazy - stworzenia użytkownika bazodanowego i nadania mu odpowiednich uprawnień, a także przygotowania schematu bazy danych. Przykład wykonania tych czynności dla bazy PostgreSQL został zaprezentowany w rozdziale poświęconym analizie ostrzeżeń.*

Ustalanie sposobu zapisu przez reguły

Odpowiednio konstruując własne typy reguł można stworzyć reguły, które zapisują ostrzeżenia i logi pakietów w sposób inny niż wynikający z ogólnej konfiguracji. Szczegółowe informacje znajdują sie w rozdziale poświęconym regułom.

Inne metody zachowywania danych

W standardowej wersji *Snort* wkompilowane są inne, rzadziej używane wtyczki realizujące zachowywanie ostrzeżeń i logów pakietów. Wymienić tu należy przede wszystkim wysyłanie informacji do logów systemowych poprzez usługę *syslog*, wysyłanie informacji bezpośrednio do konsoli oraz zapis w pliku tekstowym w postaci wartości oddzielanych przecinkami (plik CSV). Dokładne informacje znajdują się w dokumentacji *Snort*.

Ponadto preprocesory *Snort* często udostępniają możliwość zapisywania informacji w odrębnych plikach o dedykowanym formacie. Informacje na ten temat znajdują się w opisach poszczególnych preprocesorów.

Konfiguracja dekodera

Zadaniem dekodera w *Snort* jest wyekstrahowanie pakietu IP z danych dostarczonych z warstwy łącza danych. Wykonując tę czynność, dekoder sprawdza poprawność nagłówka pakietu IP i ewentualnie nagłówków warstwy transportowej (TCP, UDP). W przypadku wystąpienia nieprawidłowości dekoder generuje ostrzeżenia, używając GID 116.

Korzystając z przedstawionych poniżej dyrektyw konfiguracyjnych można wyłączyć generowanie ostrzeżeń w wybranych przypadkach.

`config disable_decode_alerts` - powoduje całkowite wyłączenie generowania ostrzeżeń przez dekoder.

`config disable_ipopt_alerts`, `config disable_tcpopt_alerts`, `config disable_ttcp_alerts` - powodują wyłączenie generowania ostrzeżeń związanych z nieprawidłowymi opcjami protokołów odpowiednio IP, TCP i T/TCP.

`config disable_tcpopt_obsolete_alerts`, `config disable_tcpopt_experimental_alerts` - powodują wyłączenie generowania ostrzeżeń związanych z przestarzałymi i eksperymentalnymi opcjami protokołu TCP.

`config enable_decode_oversized_alerts` - włącza sprawdzanie wartości długości segmentu danych, która jest zapisana w nagłówku pakietu, i porównywanie jej z wielkością całego pakietu. Jeżeli zadeklarowana opcja pakietu długość segmentu danych jest większa niż długość pakietu, następuje wygenerowanie ostrzeżenia.

`config checksum_mode lista_protokołów` - włącza sprawdzanie sumy kontrolnej dla protokołów wymienionych na liście (oddzielonych znakiem spacji). Dostępne wartości listy to: `ip`, `icmp`, `tcp`, `udp`, słowo kluczowe `all` oznaczające wszystkie znane dekoderowi protokoły, słowo kluczowe `none` oznaczające wyłączenie sprawdzania sum kontrolnych, oraz wartości `noip`, `noicmp`, `notcp` i `noudp` wyłączające sprawdzanie sum kontrolnych dla poszczególnych protokołów.

Przykład:

`config checksum_mode ip icmp tcp udp` - konfiguracja domyślna;

`config checksum_mode all noip noicmp` - sprawdzanie sum kontrolnych wszystkich protokołów z wyjątkiem IP i ICMP.

Zobacz też: dokumentacja w pliku /usr/share/doc/snort-2.9.x.y/README.decode

Konfiguracja preprocesorów

Preprocesory *Snort* to wyspecjalizowane moduły, dokonujące analizy ruchu sieciowego pod wybranym kątem. Analiza ta nie musi opierać się na danych zawartych w pojedynczym pakiecie, ponieważ preprocesor może przechowywać dane z wielu pakietów - dzięki temu możliwa jest np. analiza strumienia danych w TCP czy wykrywanie skanowania portów. W niniejszym rozdziale opisanych zostało kilka najważniejszych preprocesorów.

Anomalie wykrywane przez preprocesory

Preprocesory generują ostrzeżenia na skutek wykrycia anomalii w analizowanych danych. W niniejszym opracowaniu zrezygnowano ze szczegółowego wymieniania wszystkich anomalii dla poszczególnych preprocesorów, ponieważ znając identyfikator GID, jakim oznaczane są ostrzeżenia generowane przez dany preprocesor, można łatwo odszukać te informacje w pliku */etc/snort/gen-msg.map*.

Przykład: wypisanie anomalii wykrywanych przez preprocesor *frag3* (GID 123):

```
# grep 123 /etc/snort/gen-msg.map
123 || 1 || frag3: IP Options on fragmented packet
123 || 2 || frag3: Teardrop attack
123 || 3 || frag3: Short fragment, possible DoS attempt
123 || 4 || frag3: Fragment packet ends after defragmented packet
123 || 5 || frag3: Zero-byte fragment
123 || 6 || frag3: Bad fragment size, packet size is negative
123 || 7 || frag3: Bad fragment size, packet size is greater than 65536
123 || 8 || frag3: Fragmentation overlap
```

W uzyskanej liście pierwsze pole to GID preprocesora, drugie - SID anomalii, trzecie - opis anomalii.

Dla ułatwienia w tytułach podrozdziałów zawarto oprócz nazwy preprocesora przypisany mu numer GID.

Frag 3 (GID: 123)

Preprocesor ten składa pakiety z fragmentów przesyłanych w ramkach warstwy łącza danych. Różne systemy operacyjne posiadają różne implementacje składania pakietów, właściwości tych implementacji stanowią podstawę do

Notatki

przeprowadzania - za pomocą odpowiednio spreparowanych fragmentów - ataków mających na celu na ogół zakłócenie działania oprogramowania stosu TCP/IP w atakowanym systemie. Anomalie wykrywane przez preprocesor *frag3* świadczą o możliwej próbie przeprowadzania takiego ataku.

Preprocesor *frag3* włączany jest poprzez użycie dyrektywy:

```
preprocessor frag3_engine [bind_to lista_adresów] [opcje]
```

Użycie opcji `bind_to` umożliwia uruchomienie wielu instancji preprocesora *frag3* z różnymi opcjami i przydzielanie ruchu do tych instancji według docelowego adresu IP; jest to użyteczne przede wszystkim w połączeniu z opcją `policy` określającą sposób składania fragmentów. Aby to osiągnąć, należy użyć dyrektywy `frag3_engine` wielokrotnie z różnymi listami adresów; analizowany pakiet zostanie przekazany do pierwszej znalezionej instancji, do której listy adresów jest dopasowany. Lista adresów składa się z adresów IP lub adresów podsieci w formacie CIDR przedzielonych przecinkami.

Pozostałe opcje dyrektywy `frag3_engine` to:

`detect_anomalies`	Włącza wykrywanie anomalii.
`policy polityka`	Określa politykę składania pakietu z fragmentów. Różne systemy operacyjne używają różnych metod składania pakietów. Opcja ta, w połączeniu z opcją `bind_to`, pozwala na dopasowanie sposobu składania przez *frag3* do metody używanej przez chroniony system. Dostępne polityki to: `first`, `last`, `linux`, `windows`, `bsd`, `bsd-right` i `solaris`. W przypadku systemów z rodzin Linux, Windows, Solaris i BSD dobór polityki jest oczywisty. Polityka `last` używana jest przez urządzenia Cisco. Informacje o politykach stosowanych dla innych systemów zawarte są w dokumentacji preprocesora. Domyślną wartością jest `bsd`.
`ttl_limit liczba`	Określa maksymalną różnicę pomiędzy TTL pierwszego fragmentu pakietu a TTL kolejnych fragmentów (domyślnie 5).
`min_ttl liczba`	Określa minimalny TTL fragmentu (domyślnie 1).
`timeout czas`	Określa maksymalny czas w sekundach, przez jaki fragment jest przechowywany w oczekiwaniu na kolejne fragmenty (domyślnie 60).

Przykład:

```
preprocessor frag3_engine bind_to 10.3.15.0/24 172.16.0.63 policy linux detect_anomalies
preprocessor frag3_engine bind_to 10.3.0.1 policy last detect_anomalies
preprocessor frag3_engine policy windows detect_anomalies
```

Każda instancja preprocesora *frag3* rezerwuje pamięć na przechowywane fragmenty. Dyrektywa:

```
preprocessor frag3_global opcje
```

ustala ilość pamięci rezerwowanej przez każdą instancję. Dyrektywa ta może wystąpić tylko raz. Opcje dyrektywy `frag3_global` to:

`max_frags ilość` - określa maksymalną ilość jednocześnie przechowywanych fragmentów (domyślnie 8192)

`memcap rozmiar` - określa maksymalną ilość pamięci zajętej w celu przechowywania fragmentów; ilość ta podawana jest w bajtach (wartość domyślna odpowiada 4 MB).

Opcje te posiadają swoje odpowiedniki o nazwach `prealloc_frags` i `prealloc_memcap`, które mają to samo znaczenie, lecz powodują włączenie odmiennego algorytmu zarządzania danymi, który w specyficznych sytuacjach może okazać się bardziej wydajny.

Zobacz też: dokumentacja w pliku /usr/share/doc/snort-2.9.x.y/README.frag3

Stream 5 (GID: 129)

Preprocesor ten składa i śledzi sesje wymiany danych protokołów TCP i UDP. Podobnie jak *frag3*, preprocesor ten jest konfigurowany za pomocą dyrektyw konfiguracyjnych osobno dla strumieni TCP i UDP (`stream5_tcp` i `stream5_udp`) oraz dyrektywy konfiguracyjnej określającej globalną konfigurację wszystkich instancji preprocesora (`stream5_global`).

Sesje TCP

Ponieważ różne systemy operacyjne posiadają różne implementacje protokołu TCP, preprocesor *stream5* dla protokołu TCP można uruchomić w wielu instancjach, przypisując im różne adresy docelowe opcją `bind_to` oraz różne porty opcją `ports`, i ustawiając pozostałe opcje indywidualnie dla każdej instancji:

```
preprocessor stream5_tcp [bind_to lista_adresów] [ports lista_portów] [opcje]
```

Składnia i znaczenie opcji `bind_to` jest identyczna jak w przypadku preprocesora *frag3*. Lista portów opcji `ports` musi rozpoczynać się jednym ze słów kluczowych: `client`, `server` lub `both`, po którym musi być umieszczona lista numerów portów oddzielanych spacjami lub słowo kluczowe `all` oznaczajace wszystkie porty, np. `ports client all`, `ports server 80 443`.

Ważniejsze opcje dyrektywy `stream5_tcp` to:

`detect_anomalies`	Włącza wykrywanie anomalii.
`policy polityka`	Określa politykę śledzenia sesji TCP. Różne systemy operacyjne używają różnych implementacji protokołu TCP. Opcja ta, w połączeniu z opcjami `bind_to` i `ports`, pozwala na dopasowanie sposobu śledzenia przez *stream5* do implementacji używanej przez chroniony system. Najważniejsze polityki to: `linux` - systemy Linux z jądrem 2.4 i nowszym, `linux-old` - systemy Linux z jądrem 2.2 i starszym, `win2003` - system Windows 2003, `vista` - system Windows Vista, `windows` - pozostale systemy Windows, `macos` - systemy MacOS 10.3 i nowsze, `solaris` - systemy Solaris 9 i nowsze, `bsd` - systemy z rodziny BSD. Domyślną wartością jest `bsd`.
`overlap_limit liczba`	Jeżeli liczba pakietów, w których stwierdzono nakładanie się segmentów danych (*overlapped packets*) osiągnie limit, zostanie wygenerowane ostrzeżenie. Wartość 0 oznacza brak limitu i jest domyślna; maksymalna wartość to 255.
`min_ttl liczba`	Określa minimalny TTL pakietu (domyślnie 1).
`timeout czas`	Określa maksymalny czas w sekundach, po którym sesja TCP zostaje uznana za zerwaną. Wartość domyślna to 30, wartość maksymalna to 86400 (jedna doba).
`max_window liczba`	Określa maksymalną uznawaną za prawidłową wielkość okna. Wartość 0 oznacza brak limitu i jest domyślna. Wartość maksymalna to 1073725440 - odpowiada ona maksimum określonemu w RFC dla protokołu TCP. Ustawianie bardzo dużych wielkości okien jest jedną z technik ataków typu DoS, dlatego wielkość okna większa niż określoną tą opcją będzie traktowana jako anomalia.
`check_session_hijacking`	Włączenie tej opcji (domyślnie wyłączonej) powoduje, że adresy warstwy łącza danych w pakietach należących do sesji TCP są porównywane do tych samych adresów w pakietach nawiązujących sesję; niezgodność jest traktowana jako przejęcie (ang. *hijack*) sesji przez intruza.
`require_3whs [karencja]`	Domyślnie *Snort* traktuje wymianę pakietów TCP, nie poprzedzoną standardowym nawiązaniem sesji poprzez wymianę pakietów SYN, SYN ACK i ACK (ang. *3-way handshake*), jako kontynuację sesji nawiązanych przed uruchomieniem *Snort.* Użycie opcji `require_3whs` bez karencji lub z karencją równą 0 powoduje, że wszelkie takie wymiany są uznawane za anomalie. Dodatnia wartość karencji wyznacza czas w sekundach liczony od uruchomienia *Snort*, w jakim takie wymiany są uznawane za kontynuowane sesje; wartość maksymalna to 86400 (jedna doba).

Przykład:

```
preprocessor stream5_tcp bind_to 10.3.15.0/24 172.16.0.63 ports server all policy linux
preprocessor stream5_tcp policy windows
```

Sesje UDP

Ponieważ protokół UDP nie definiuje pojęcia sesji, śledzenie sesji UDP jest przez *Snort* realizowane poprzez zwykłe obserwowanie wymiany pakietów. Dlatego też śledzenie sesji UDP jest włączane pojedynczą dyrektywą:

```
preprocessor stream5_udp [timeout liczba] [ignore_any_rules]
```

Opcja `timeout` ma takie samo znaczenie jak w przypadku TCP. Użycie opcji `ignore_any_rules` ma wpływ na działanie mechanizmu detektora i powoduje, że ignorowane są reguły, które używają tylko kryteriów odnoszących się do pola danych pakietu, a nie określają zakresu portów (ani źródłowych, ani docelowych). Istnienie tej opcji jest podyktowane znacznym obciążeniem systemu przez analizę danych dużej ilości pakietów. Działanie tej opcji nie dotyczy reguł używających kryterium `flow` lub `flowbits`. Lista reguł ignorowanych na skutek działania tej opcji jest wypisywana podczas uruchamiania *Snort*.

Sesje ICMP

Notatki

Śledzenie sesji ICMP opiera się na tych samych założeniach co w przypadku UDP, jego implementacja jest jednakże na etapie eksperymentalnym i nie zaleca się jego stosowania w środowiskach produkcyjnych. Śledzenie sesji ICMP jest włączane pojedynczą dyrektywą:

```
preprocessor stream5_icmp [timeout liczba]
```

Opcja `timeout` ma takie samo znaczenie jak w przypadku TCP.

Konfiguracja ogólna

Dyrektywa:

```
preprocessor stream5_global opcje
```

ustala parametry ilościowe dla kazdej instancji preprocesora stream5. Opcje tej dyrektywy to:

`track_tcp yes/no`, `track_tcp yes/no`, `track_tcp yes/no` - włącza/wyłącza śledzenie sesji poszczególnych protokołów. Domyślnie śledzenie jest włączone dla wszystkich protokołów (pod warunkiem użycia odpowiedniej dyrektywy z wymienionych powyżej)

`max_tcp ilość`, `max_udp ilość`, `max_icmp ilość` - określa maksymalną ilość jednocześnie śledzonych sesji; maksymalna wartość dla każdego z protokołów to 1052672, wartości domyślne to 256000 - TCP, 128000 - UDP i 64000 - ICMP.

`memcap rozmiar` - określa maksymalną ilość pamięci zajętej w celu przechowywania pakietów przez każdą instancję preprocesora śledzącego sesje TCP. Ilość ta podawana jest w bajtach; wartość domyślna odpowiada 8 MB, wartość minimalna - 32 KB, maksymalna - 1 GB.

? *Preprocesor stream5 jest następcą używanych wcześniej preprocesorów stream4 i flow. Dla zachowania zgodności z poprzednimi wersjami oba te preprocesory są również wkompilowane w Snort i można je włączać, ale niemożliwe jest jednoczesne użycie stream5 i któregokolwiek z pary: stream4 i flow.*

? *Zobacz też: dokumentacja w pliku /usr/share/doc/snort-2.9.x.y/README.stream5*

SFPortscan (GID: 122)

Preprocesor ten wykrywa próby badania otwartych portów w danym systemie lub grupie systemów w przypadku protokołów TCP i UDP oraz próby wysyłania jednakowych pakietów do grupy systemów w celu zbadania ich odpowiedzi. Czynności te potocznie nazywane są skanowaniem portów albo skanowaniem systemów (ang. *port scanning*, *host scanning*) i stanowią z reguły wstępną fazę ataku (rozpoznanie, ang. *reconnaisance*). Ponieważ tego typu zdarzenia są ściśle powiązane z otwieraniem sesji protokołów TCP i UDP, użycie tego preprocesora wymaga użycia również preprocesora *stream5* lub pary: *stream4* i *flow*.

Rodzaje skanowania portów

Klasyczne skanowanie portów, zwane w terminologii angielskiej *portscan*, ma miejsce wtedy, gdy z jednego adresu (systemu) przeprowadzana jest próba łączenia się z wieloma portami systemu skanowanego. Prowadzi to do ustalenia, jakie usługi - stanowiące potencjalny cel ataku - są uruchomione na danym systemie. Z kolei testowanie portu (*portsweep*) ma miejsce wtedy, gdy atakujący próbuje połączyć się z tym samym portem na wielu systemach. Prowadzi to do ustalenia, które z badanych systemów mogą być podatne na konkretny atak, który chce przeprowadzić atakujący.

Skanowanie/testowanie maskowane (*decoy portscan/portsweep*) ma miejsce wtedy, gdy atakujący wysyła pakiety o zafałszowanym adresie źródłowym przemieszane z tymi o właściwym adresie źródłowym. Wysyłanie pakietów o fałszywym adresie nie przynosi atakującemu korzyści bezpośredniej (jako że odpowiedzi do niego nie docierają), ale jest próbą ukrycia swojej tożsamości i samego ataku przed IDS.

Skanowanie/testowanie rozproszone (*distributed portscan/portsweep*) ma miejsce wtedy, gdy atakujący przejmuje kontrolę nad grupą nie należących do niego systemów (tzw. *systemów-niewolników* lub *systemów-zombie*) i wykorzystuje te systemy do prowadzenia skanowania czy testowania portów. Umożliwia to atakującemu skuteczne ukrycie własnej tożsamości, jako że jego rzeczywisty adres nie jest używany do przeprowadzenia ataku. Skanowanie portów jednego systemu z wielu *systemów-zombie* może także stanowić atak DoS (tzw. *SYN Flood*).

Skanowanie portów a filtrowanie pakietów

Standardowo preprocesor *sfportscan* wykrywa skanowanie portów poprzez stwierdzenie odpowiedniej reakcji systemu skanowanego. Reakcją taką może być wysłanie pakietu TCP z włączoną flagą RST lub pakietu ICMP z komunikatem błędu. Jeżeli jednak atakowany system jest chroniony przez zaporę sieciową, to prawdopodobnie zapora ta porzuci niechciane pakiety bez odsyłania jakiejkolwiek informacji. W takim przypadku rozpoznanie skanowania portów musi być dokonane na podstawie aktywności systemu skanującego. Taka metoda rozpoznawania może jednakże wykazywać dużą ilość fałszywych alarmów w przypadku systemów, które z natury swojego działania nawiązują dużo sesji, np. serwerów proxy, serwerów DNS czy ruterów dokonujących translacji adresu źródłowego. Ostrzeżenia o zdarzeniach wykrytych przez *sfportscan* w taki sposób są oznaczane słowem *filtered*.

Konfiguracja *sfportscan*

Preprocesor *sfportscan* włączany jest poprzez użycie dyrektywy:

```
preprocessor sfportscan sense_level {poziom} proto {protokoły} scan_type {rodzaje} \
memcap {liczba} [opcje]
```

Opcja `sense_level` ustala poziom czułości wykrywania zdarzeń. Dostępne poziomy to `low`, `medium` i `high`. Poziom `low` oznacza wykrywanie skanowania tylko na podstawie reakcji skanowanego systemu. Poziom `medium` włącza również wykrywanie skanowania systemów chronionych zaporą sieciową (*filtered).* Poziom `high` stosuje ostrzejsze kryteria ilościowo-czasowe niż poziom medium.

Zastosowanie wyższego poziomu czułości oczywiście zwiększa skuteczność wykrywania zdarzeń, ale jednocześnie zwiększa ryzyko wystąpienia fałszywych ostrzeżeń. Zalecane jest korzystanie z pozostałych opcji dla obniżenia liczby fałszywych ostrzeżeń, i dopiero w ostateczności obniżanie poziomu czułości.

Opcja `proto` pozwala na ustalenie protokołu (lub listy protokołów oddzielanej spacjami) śledzonego przez preprocesor. Dostępne protokoły to `tcp`, `udp`, `icmp`, `ip` i słowo kluczowe `all` oznaczające wszystkie protokoły.

Opcja `scan_type` pozwala na ustalenie rodzaju wykrywanych skanowań (lub listy rodzajów oddzielanej spacjami). Dostępne rodzaje skanowań to `portscan`, `portsweep`, `decoy_portscan`, `distributed_portscan` i słowo kluczowe `all` oznaczające wszystkie rodzaje skanowań.

Opcja `memcap` ustala ilość pamięci rezerwowanej przez preprocesor do przechowywania danych o aktywności sieciowej systemów. Wartość ta jest podawana w bajtach i zaleca się, aby odpowiadała ona przynajmniej 1 MB.

Ważniejsze pozostałe opcje dyrektywy `preprocessor sfportscan` to:

`watch_ip {lista}`	Określa listę adresów, która dotyczy zarówno systemów skanujących jak i skanowanych; śledzone są tylko skanowania, w których pakiety mają adres źródłowy lub docelowy z tej listy. Lista adresów składa się z wartości przedzielonych przecinkami, bez odstępów. Każda z wartości może być adresem IP lub adresem podsieci w formacie CIDR; opcjonalnie do każdej wartości może być po dwukropku dołączony port lub zakres portów. **Przykład:** `watch_ip {172.16.0.63:1-1024,10.3.0.0/16}`
`ignore_scanners {lista}` `ignore_scanned {lista}`	Określa listę adresów, które będą ignorowane jako odpowiednio systemy skanujące i systemy skanowane. Na listy te należy wpisać adresy systemów, których normalna aktywność sieciowa powoduje generowanie fałszywych ostrzeżeń, takie jak serwery proxy, serwery DNS czy rutery dokonujące translacji adresu źródłowego. Lista adresów ma format identyczny z listą występującą w opcji `watch_ip` z tym, że nie może zawierać informacji o portach.
`logfile {plik}`	Włącza zapisywanie dodatkowych komunikatów preprocesora *sfportscan* do wymienionego pliku.

Logi pakietów preprocesora *sfportscan*

Charakterystyczną cechą preprocesora *sfportscan* jest to, że wykrywa on zdarzenia bazując nie na pojedynczym wystąpieniu pakietu, lecz grupy (często dość licznej) pakietów. Aby uniknąć zapisywania do logów pakietów zbyt dużej ilości danych, *sfportscan* zapisuje tam specjalnie wygenerowany fałszywy pakiet, którego wyróżnikiem jest numer protokołu warstwy transportowej wynoszący 255 oraz TTL równy 0. W segmencie danych tego pakietu są zapisane (w postaci tekstowej) dodatkowe informacje o zdarzeniu.

Przykład (zapis pakietów w formacie *pcap*, następnie wyświetlenie za pomocą polecenia `snort` w trybie sniffera):

```
# snort -vd -r /var/log/snort/tcpdump.log.1199291617
10/27-17:34:54.745230 192.168.0.202 -> 192.168.0.201
PROTO:255 TTL:0 TOS:0x0 ID:3480 IpLen:20 DgmLen:166
50 72 69 6F 72 69 74 79 20 43 6F 75 6E 74 3A 20  Priority Count:
30 0A 43 6F 6E 6E 65 63 74 69 6F 6E 20 43 6F 75  0.Connection Cou
6E 74 3A 20 32 30 30 0A 49 50 20 43 6F 75 6E 74  nt: 200.IP Count
3A 20 31 0A 53 63 61 6E 6E 65 72 20 49 50 20 52  : 1.Scanner IP R
61 6E 67 65 3A 20 31 39 32 2E 31 36 38 2E 30 2E  ange: 192.168.0.
```

Notatki

```
32 30 32 3A 31 39 32 2E 31 36 38 2E 30 2E 32 30  202:192.168.0.20
32 0A 50 6F 72 74 2F 50 72 6F 74 6F 20 43 6F 75  2.Port/Proto Cou
6E 74 3A 20 31 37 37 0A 50 6F 72 74 2F 50 72 6F  nt: 177.Port/Pro
74 6F 20 52 61 6E 67 65 3A 20 32 32 3A 31 30 30  to Range: 22:100
39 0A                                            9.
```

Własne ostrzeżenia preprocesora *sfportscan*

Jeżeli w dyrektywie `preprocessor sfportscan` użyto opcji `logfile`, to preprocesor zapisuje dodatkowe, rozszerzone informacje o wykrytych zdarzeniach do osobnego pliku. Poniżej przedstawiono przykład takiego komunikatu:

```
Time: 01/02-17:34:54.745230
event_id: 1
192.168.0.202 -> 192.168.0.201 (portscan) TCP Filtered Portscan
Priority Count: 0
Connection Count: 200
IP Count: 1
Scanner IP Range: 192.168.0.202:192.168.0.202
Port/Proto Count: 177
Port/Proto Range: 22:1009
```

Interpretacja tych informacji jest następująca: `event_id` to jednoznaczny identyfikator wystąpienia zdarzenia, dzięki któremu można powiązać z nim inne ostrzeżenia; `Priority Count` to ilość odnotowanych priorytetowych zajść świadczących o zaistnieniu zdarzenia (są to: wysłanie przez system skanowany pakietu TCP RST lub komunikatu błędu ICMP); `Connection Count` to ilość sesji między systemem skanującym a skanowanym, które są w fazie otwierania (duża różnica wartości `Connection Count` i `Priority Count` świadczy o tym, że system skanowany jest chroniony przez zaporę sieciową); `IP Count` to ilość adresów, z których przeprowadzano skanowanie (źródłowych); `Port Count` to ilość portów, które zostały zbadane (docelowych); `Scanner IP Range` to lista adresów, z których przeprowadzano skanowanie; w przypadku ataków typu *portsweep* zamiast tej wartości występuje `Scanned IP Range` - lista adresów, które były badane.

Oprócz ogólnej informacji o zdarzeniu preprocesor *sfportscan* zapisuje informacje o każdym otwartym (nasłuchującym) porcie, który został wykryty:

```
Time: 01/02-17:34:54.745231
event_ref: 1
192.168.0.202 -> 192.168.0.201 (portscan) Open Port
Open Port: 22

Time: 01/02-17:34:54.745232
event_ref: 1
192.168.0.202 -> 192.168.0.201 (portscan) Open Port
Open Port: 443

Time: 01/02-17:34:54.745233
event_ref: 1
192.168.0.202 -> 192.168.0.201 (portscan) Open Port
Open Port: 80
```

Wartość `event_ref` umożliwia powiązanie tej informacji z informacją ogólną poprzez porównanie `event_ref` z `event_id`.

Zobacz też: dokumentacja w pliku /usr/share/doc/snort-2.9.x.y/README.sfportscan

DNS (GID: 131)

Preprocesor ten analizuje zapytania DNS. Domyślnie nie generuje on ostrzeżeń, aby to osiągnąć należy użyć odpowiednich opcji konfiguracyjnych.

Preprocesor *dns* włączany jest poprzez użycie dyrektywy:

```
preprocessor dns [opcje]
```

Opcje preprocesora *dns* to:

`ports {lista}`	Określa listę (oddzielaną przecinkami) portów docelowych. Domyślnie brany pod uwagę jest ruch na port docelowy 53.
`enable_obsolete_types` `enable_experimental_types`	Włącza generowanie ostrzeżeń w przypadku stwierdzenia użycia odpowiednio przestarzałych/eksperymentalnych typów zapytań DNS, zgodnie z RFC 1035.
`enable_rdata_overflow`	Włącza generowanie ostrzeżeń w przypadku stwierdzenia zbyt długiego zapytania DNS (próba wywołania przepełnienia bufora serwera).

Zobacz też: dokumentacja w pliku /usr/share/doc/snort-2.9.x.y/README.dns

HTTP Inspect (GID: 123)

Preprocesor ten analizuje ruch pod kątem protokołu HTTP. Oprócz wykrywania zdarzeń - nieprawidłowości w informacjach wymienianych przez serwer i klienta HTTP preprocesor ten dokonuje tzw. normalizacji nagłówków żądań i odpowiedzi HTTP; przykładem normalizacji jest zamiana ścieżki */dir1//dir2/./dir3* na równoważną, znormalizowaną postać */dir1/di2/dir3*. Z normalizacji tej można skorzystać tworząc reguły operujące na danych protokołu HTTP, np. z użyciem kryterium `uricontent`.

Powyższe oznacza, że wyłączenie preprocesora http_inspect, a także ograniczenie zakresu jego działania, powoduje dezaktywowanie reguł korzystających z danych przygotowywanych przez ten preprocesor.

Preprocesor http_inspect NIE dekoduje danych transmitowanych szyfrowanym protokołem HTTPS.

Preprocesor *http_inspect* włączany jest poprzez użycie jednej z dyrektyw:

```
preprocessor http_inspect: global opcje_globalne
preprocessor http_inspect_server: server adres opcje_serwera
```

Pierwsza z dyrektyw może wystąpić jednokrotnie i określa opcje wspólne dla wszystkich instancji preprocesora. Druga dyrektywa może wystąpić wielokrotnie i określa szczegółowe opcje dla konkretnego serwera HTTP, określonego poprzez podanie adresu IP. W szczególnym przypadku jako adres może zostać podane słowo kluczowe `default` i oznacza to konfigurację dla wszystkich serwerów nie wymienionych wprost.

Poniżej zaprezentowano opcje globalne preprocesora *http_inspect*.

`detect_anomalous_servers`	Włącza wyszukiwanie danych protokołu HTTP w całym ruchu analizowanym przez *Snort*. Jeżeli dane takie zostaną i serwer nie ma dedykowanej konfiguracji preprocesora oraz port serwera nie jest wymieniony na liście domyślnej konfiguracji, wygenerowane zostanie ostrzeżenie.
`proxy_alert`	Włącza wyszukiwanie zapytań HTTP Proxy w całym ruchu analizowanym przez *Snort*. Jeżeli dane takie zostaną i serwer nie ma dedykowanej konfiguracji preprocesora (z użyciem opcji `allow_proxy_use`), generowane jest ostrzeżenie.
`iis_unicode_map plik [codemap strona_kodowa]`	Użycie tej opcji w konfiguracji globalnej jest obowiązkowe, aczkolwiek dotyczy ona tylko serwera Microsoft IIS. IIS używa dwubajtowego kodowania Unicode dla znaków spoza standardowego zbioru ASCII w nagłówku. Tak zakodowane znaki są przez serwer mapowane na znaki w stronie kodowej, w jakiej pracuje konkretny serwer. Aby preprocesor *http_inspect* mógł prawidłowo interpretować nagłówek, musi posiadać mapę translacji umożliwiającą wykonanie operacji mapowania. Opcja `iis_unicode_map` wskazuje lokalizację pliku zawierającego mapowanie znaków. Ponieważ plik taki może zawierać mapowania dla wielu stron kodowych, opcją `codemap` można wskazać konkretną stronę kodową do wykorzystania (jeżeli nie zostanie użyta - mapowanie dla konkretnego znaku zostanie wyszukane we wszystkich stronach kodowych). W standardowym pakiecie Snort dołączony jest plik *unicode.map*, zawierający mapowanie dla podstawowych stron kodowych, w tym CP 1250 i ISO-8859-2. Aby wygenerować własny plik mapowania, należałoby pobrać i skompilować specjalny program narzędziowy dystrybuowany wraz z kodem źródłowym preprocesora *http_inspect* i uruchomić go na wybranym serwerze. Opcję `iis_unicode_map` można także zastosować w konfiguracji konkretnego serwera.
`min_ttl liczba`	Określa minimalny TTL fragmentu (domyślnie 1).
`timeout czas`	Określa maksymalny czas w sekundach, przez jaki fragment jest przechowywany w oczekiwaniu na kolejne fragmenty (domyślnie 60).

Notatki

Konfiguracja dla konkretnego serwera zawiera bardzo wiele opcji dotyczących sposobu kodowania danych binarnych. Używanie tych opcji wymaga bardzo szczegółowej wiedzy na temat działania danego serwera HTTP, dlatego zalecane jest raczej używanie jednego z predefiniowanych profili, zawierających ustawienia dobrane pod kątem konkretnego serwera HTTP. Zgodnie z tym zaleceniem opcje kodowania zostały pominięte w poniższym opisie opcji konfiguracji konkretnego serwera.

`ports {lista_portów}`	Określa listę (oddzielanych spacjami) portów wykorzystywanych przez serwer. Domyślna lista zawiera tylko port 80.
`profile profil`	Włącza użycie danego profilu dla serwera. Opcja ta musi być podana jako pierwsza. Dostępne profile to: `apache` - serwer Apache (httpd), `iis` - Microsoft IIS oraz `all` - inne serwery HTTP. Opcja ta nie jest obowiązkowa; jeżeli nie zostanie użyta, zostaną użyte ustawienia domyślne, uwzględniające normalizację nagłówków, ale nie generujące ostrzeżeń w przypadku wystąpienia anomalii w kodowaniu danych.
`iis_unicode_map plik [codemap strona_kodowa]`	Znaczenie tej opcji jest identyczne jak w przypadku globalnych opcji preprocesora *http_inspect*. Opcja ta nie jest obowiązkowa; jeżeli nie zostanie użyta, zostaną użyte ustawienia globalne.
`allow_proxy_use`	Wskazuje, że dany serwer może być serwerem HTTP Proxy; opcja ta ma zastosowanie tylko wtedy, gdy użyto opcji globalnej `proxy_alert` i powoduje, że zapytania/odpowiedzi HTTP Proxy wymieniane z tym serwerem nie będą powodowały generowania ostrzeżeń. Dzięki użyciu opcji `proxy_alert` i `allow_proxy_use` można wykrywać dwa rodzaje niepożądanych zjawisk: - próby użycia przez użytkowników "wewnętrznych" serwera HTTP Proxy innego niż oficjalnie wymagany, prawdopodobnie w celu ominięcia restrykcji narzuconych na ten serwer proxy; - próba użycia przez uzytkowników "zewnętrznych" serwera HTTP Proxy w celu przeprowadzenia ataku na wybrany serwer HTTP (z użyciem adresu źródłowego serwera proxy);
`no_alert`	Całkowicie wyłącza generowanie przez preprocesor *http_inspect* ostrzeżeń dla danego serwera. Nie wyłącza to generowania ostrzeżeń przez reguły bazujace na inspekcji danych protokołu HTTP.
`flow_depth rozmiar`	Ogranicza przetwarzanie odpowiedzi HTTP do wymienionej liczby bajtów, licząc od początku odpowiedzi. Ograniczenie takie jest podyktowane faktem, że w zasadzie wszystkie możliwe anomalie wykrywane są w nagłówku odpowiedzi, a analizowanie całego zwracanego zasobu, niekiedy o bardzo dużym rozmiarze, może znacznie zmniejszyć wydajność *Snort.* Rozmiar podawany jest w bajtach i może przybrać wartości nie większe niż 1460, przy czym nagłówki odpowiedzi HTTP bardzo rzadko osiągają rozmiary przekraczające 300 bajtów i jest to wartość domyślna. Wartość -1 oznacza calkowite ignorowanie odpowiedzi serwerów HTTP. Wartość 0 oznacza analizę całych odpowiedzi serwerów HTTP.
`inspect_uri_only`	Powoduje, że w żądaniu HTTP analizie podlega tylko lokalizator zasobu - URI. Ponieważ ogromna większość ataków opiera się na manipulacji lokalizatorem, takie ograniczenie wbrew pozorom nie powoduje znacznego zmniejszenia skuteczności, natomiast radykalnie zwieksza jego wydajność. **UWAGA:** Należy pamiętać, że przy użyciu tej opcji, dla danego serwera nieaktywne będą reguły używające kryteriów odnoszących się do danych protokołu HTTP, z wyjątkiem `uricontent`.

Przykład:

```
preprocessor http_inspect detect_anomalous_servers proxy_alert iis_unicode_map unicode.map
preprocessor http_inspect_server server 172.16.0.3 profile apache ports {80}
preprocessor http_inspect_server server 172.16.0.5 profile iis ports {80}
preprocessor http_inspect_server server 172.16.0.63 profile all allow_proxy_use ports {3128 8080}
preprocessor http_inspect_server server default profile all ports {80}
```

Zobacz też: dokumentacja w pliku /usr/share/doc/snort-2.9.x.y README.http_inspect

Pozostałe

Oprócz wymienionych preprocesorów w standardowej dystrybucji *Snort* zawarte są inne preprocesory; niektóre z nich są rzadziej używane, inne mają status kodu eksperymentalnego.

Zobacz też: dokumentacja w katalogu /usr/share/doc/snort-2.9.x.y

Notatki

5 Reguły Snort

Ostatnią faza analizowania ruchu przez *Snort* jest zastosowanie reguł. Mechanizm reguł jest bardzo uniwersalny, umożliwia bowiem zastosowanie do pakietów rozmaitych kryteriów, opartych zarówno o nagłówki protokołów warstwy sieciowej i transportowej, jak i o dane przenoszone przez pakiet.

Definiowanie zmiennych

Reguły *Snort* (choć nie tylko one) mogą być oparte wartości zdefiniowanych wcześniej zmiennych. Umożliwia to stworzenie ogólnej konfiguracji, łatwej do przystosowania do konkretnego zastosowania, z czego szeroko korzysta konfiguracja domyślna *Snort* (zostało to przedstawione w rozdziale poświęconym dostosowywaniu konfiguracji domyślnej).

Wyróżniamy trzy rodzaje zmiennych:

- zmienne napisowe (ogólne)
- zmienne adresowe
- zmienne portowe

Zmienne napisowe definiowane są następująco:

```
var nazwa_zmiennej napis
```

Przykład:

```
var RULE_PATH /etc/snort/rules
```

Zmienne adresowe są definiowane również za pomocą dyrektywy `var`, istnieje wprawdzie dyrektywa `ipvar`, ale można jej używać tylko wtedy, gdy w *Snort* jest wkompilowana obsługa IPv6.

W najprostszej postaci zmienne adresowe są definiowane jako zwykłe adresy IP (nie ma możliwości stosowania nazw DNS) lub adresy podsieci w formacie CIDR:

```
var SIMPLE_HOST 10.0.0.1            var SIMPLE_NET 10.0.0.0/16
```

Wartość taka może zostać zanegowana poprzez użycie operatora negacji `!` i wtedy odczytujemy ją jako "wszystkie adresy z wyjątkiem...":

```
var NOT_HOST !10.0.0.1        var NOT_NET !10.0.0.0/16
```

Można także określić wartość zmiennej adresowej jako listę, na której mogą występować oba rodzaje adresów, wówczas cała lista odpowiada sumie (w sensie rachunku zbiorów) poszczególnych adresów. Lista musi być oddzielana przecinkami, zawarta w nawiasach kwadratowych i nie może zawierać znaków odstępu.

```
var SIMPLE_LIST [10.0.0.1,10.0.0.2,10.1.0.0/16]
```

Do takiej listy można także zastosować negację i znaczenie zanegowanej listy jest takie samo jak w przypadku pojedynczych wartości:

```
var SIMPLE_LIST ![10.0.0.1,10.0.0.2,10.1.0.0/16]
```

W ramach listy można używać wartości zwykłych i zanegowanych, przy czym ze względu na jednoznaczność zapisu nie zaleca się stosowania więcej niż jednego operatora negacji w jednej definicji zmiennej; jeżeli taka konieczność zaistnieje, należy posłużyć się zagnieżdżoną listą:

```
var COMPOUND_LIST [10.0.0.0/16,!10.0.0.1]
var MORE_COMPOUND_LIST [10.0.0.0/16,![10.0.0.1,10.1.1.0/24]]
```

Jako wartość zmiennej adresowej może także zostać użyte słowo kluczowe `any` oznaczające dowolny adres:

```
var ALL_ADDRESSES any
```

Do wartości `any` nie można zastosować negacji.

Zmienne portowe są definiowane za pomocą dyrektywy `portvar`. W najprostszej postaci zmienne portowe są definiowane jako zwykłe numery portów:

```
portvar SIMPLE_PORT 80
```

Zmienna portowa może także określać zakres portów, wyrażany za pomocą pary wartości oddzielonych dwukropkiem. Brak wartości po lewej stronie dwukropka jest automatycznie uzupełniany wartością 0, po prawej - 65535.

```
portvar PORT_RANGE 8000:9000  portvar LOW_PORTS :1023   portvar HIGH_PORTS 1024:
```

Wartość zmiennej portowej może zostać zanegowana i wtedy odczytujemy ją jako "wszystkie porty z wyjątkiem...":

```
portvar NOT_SSH_PORTS !22
```

Wartości portów mogą być łączone w listy i negowane w taki sam sposób, jak w przypadku zmiennych adresowych.

Jako wartość zmiennej portowej może także zostać użyte słowo kluczowe `any` oznaczające dowolny port:

```
portvar ALL_PORTS any
```

Do wartości `any` nie można zastosować negacji.

Odwołania do zmiennych

Odwołania do zmiennych mogą być trojakiego rodzaju: zwykłe, z komunikatem i z wartością domyślną.

Odwołania zwykłe zapisywane są następująco:

`$NAZWA_ZMIENNEJ` lub `$(NAZWA_ZMIENNEJ)`, np. `$EXTERNAL_NET`

Jeżeli wskazana zmienna nie jest zdefiniowana, podczas uruchamiania *Snort* zgłaszany jest błąd i uruchamianie zostaje przerwane.

Odwołania z komunikatem zapisywane są następująco:

`$(NAZWA_ZMIENNEJ:?komunikat)`, np. `$(EXTERNAL_NET:?Please define >EXTERNAL_NET<)`

Jeżeli wskazana zmienna nie jest zdefiniowana, podczas uruchamiania *Snort* zgłaszany jest błąd zawierający podany `komunikat` i uruchamianie zostaje przerwane.

Odwołania z wartością domyślną zapisywane są następująco:

`$(NAZWA_ZMIENNEJ:-wartość_domyślna)`, np. `$(EXTERNAL_NET:-any)`

Jeżeli wskazana zmienna nie jest zdefiniowana, użyta zostanie określona w odwołaniu wartość domyślna.

Negacja odwołań do zmiennych

Przed odwołaniem do zmiennej można użyć operatora negacji i w ten sposób uzyskać wartość zanegowaną. W przypadku zmiennych adresowych i portowych będących zdefiniowanymi jako listy negacja odwołania jest zaprzeczeniem całkowitej wartości listy, nie są stosowane znane z logiki reguły zaprzeczeń.

Stosując negacje odwołań do zmiennych należy pamiętać o potencjalnych pułapkach logicznych. W poniższym przykładzie zmienna `NOT_EXTERNAL_NET` nie pasuje do żadnego adresu IP, jakkolwiek formalnie jej definicja jest prawidłowa. Zastosowanie takiej zmiennej w budowie reguły spowoduje, że reguła nigdy nie będzie uwzględniana.

```
var EXTERNAL_NET any
var NOT_EXTERNAL_NET !$EXTERNAL_NET
```

Budowa reguły

Reguły są jednym z elementów pliku konfiguracyjnego Snort (domyślnie */etc/snort/snort.conf*), jednakże w praktyce umieszczane one są w odrębnych plikach konfiguracyjnych, włączanych w pliku głównym dyrektywą `include`. Takie rozwiązanie zwiększa czytelność konfiguracji i ułatwia zarządzanie regułami.

Reguła powinna być zapisywana w jednej linii, ale dla ułatwienia można rozbić ją na wiele linii używając znaku `\`.

Ogólna postać reguły jest następująca:

```
typ protokół adresacja (opcje)
```

Typy reguł

Typ reguły określa akcję, która zostanie wykonana na pakiecie, jeżeli będzie on pasował do reguły. Powiemy, że pakiet pasuje do reguły, jeżeli jest on odpowiedniego protokołu, jego adresy (źródłowy i docelowy) odpowiadają adresacji reguły, oraz spełnia on dodatkowe kryteria zawarte w opcjach reguły.

Podstawowe typy reguł Snort to: `alert`, `log` i `pass`.

Reguła typu `alert` powoduje wygenerowanie ostrzeżenia ORAZ zapisanie pakietu do logów, zgodnie z aktualną konfiguracją Snort.

Reguła typu `log` powoduje jedynie zapisanie pakietu do logów, bez generowania ostrzeżenia.

Reguła typu `pass` powoduje zignorowanie pakietu. Reguły tego typu stosuje się dla określenia ruchu, który ma zostać pominięty przy analizie, np. `pass ip any any <> 192.168.0.1 any`.

Pozostałe standardowe typy reguł stosowane są w szczególnych sytuacjach. Reguły typu `dynamic` i `activate` służą odpowiednio do definiowania i aktywowania zestawów reguł dynamicznych; funkcjonalność ta jest obecnie zastąpiona

Notatki

sekwencyjnym logowaniem pakietów i przeznaczona do wycofania w kolejnych wersjach *Snort*. Z kolei typy `drop`, `sdrop` i `reject` używane są przy działaniu *Snort* w trybie *inline*, nie omawianym w tym podręczniku.

Definiowanie własnych typów reguł

Oprócz stosowania typów standardowych istnieje również możliwość definiowania własnych typów reguł. Typ własny musi bazować na typie standardowym `log` lub `alert` (dyrektywa `type`) i umożliwia zdefiniowanie odrębnych metod zachowywania odpowiednio logów pakietów i ostrzeżeń dla reguł tego typu.

Przykład: typ reguły dokonującej zapisu ostrzeżenia w trybie pełnym do odrębnego pliku oraz reguła tego typu, generująca ostrzeżenie przy każdym wystąpieniu pakietu TCP o porcie docelowym 666.:

```
ruletype special
{
    type alert
    output alert_full: special-alerts.log
}

special tcp any any <> any 666
```

! *Nazwa nowego typu nie może zaczynać się od słowa będącego nazwą*

! *Niestety w wersji Snort dystrybuowanej w pakiecie RPM jest błąd uniemożliwiający uruchomienie Snort, jeżeli tworzona jest jakakolwiek reguła niestandardowego typu. Błąd objawia się komunikatem systemu o naruszeniu ochrony pamięci.*

Protokół i adresacja reguł

Określenie `protokołu` i `adresacja` są wstępnymi kryteriami określającymi, czy badany pakiet pasuje do danej reguły. Dopiero wtedy, gdy pakiet spełnia te warunki, zostaje sprawdzony pod kątem zgodności z kryteriami szczegółowymi, zapisanymi w opcjach reguły.

Protokół może być jedną z wartości: `tcp`, `udp`, `icmp` i `ip`.

Adresacja jest wyrażeniem określającym adres i port źródłowy oraz adres i port docelowy pakietu pasującego do reguły. Obie pary wartości są rozdzielone operatorem kierunkowości, który ma dwie postaci: `->` oraz `<>`. Pierwsza postać oznacza, że para po lewej stronie operatora to adres i port źródłowy, a po prawej – docelowy. Druga postać uwzględnia też sytuację odwrotną: wartości po lewej stronie operatora to adres i port docelowy, a po prawej – źródłowy. Określenie portu jest obowiązkowe, choć ma sens tylko w przypadku protokołów TCP i UDP, w pozostałych przypadkach należy użyć w tym miejscu wartości `any`.

Adresy i porty są wyrażane w takim samym formacie, jak w przypadku definicji zmiennych, można też stosować odwołania do zmiennych. Jest to powszechnie stosowane w standardowym zestawie reguł, gdzie znajdują się odwołania m. in. do zmiennych HOME_NET czy EXTERNAL_NET.

Przykłady:

`icmp 192.168.0.201 any -> 192.168.0.0/24 any` – tylko pakiety ICMP z adresem źródłowym 192.168.0.1 i docelowym z podsieci 192.168.0.0/24.

`tcp $HOME_NET $HTTP_PORTS <> !$HOME_NET any` – cały ruch (w obie strony) pomiędzy systemami w podsieci HOME_NET na porcie 80 a systemami spoza sieci HOME_NET na dowolnym porcie.

Kolejność przetwarzania reguł

! *Poniższy opis został utworzony na podstawie własnych eksperymentów z konfiguracją Snort. Rozbieżności z oficjalną dokumentacją wynikają prawdopodobnie z faktu, iż w wersji 2.8 Snort wprowadzono nowe zasady przetwarzania reguł, a nie uaktualniono dość dokładnie dokumentacji.*

Poszczególne typy reguł uszeregowane są w hierarchię ważności. Domyślna hierarchia to, poczynając od najważniejszej: `pass`, `alert`, `log`.

? *W poprzednich wersjach Snort stosowana była domyślna kolejność: alert, pass, log. Wobec nowego podejścia traci sens opcja -o polecenia snort, jak i opcja PASS_FIRST pliku konfiguracyjnego usługi snortd.*

Dla każdego pakietu najpierw ustalany jest zbiór reguł o typie najwyższego priorytetu. Jeżeli takie reguły istnieją, to są wykonywane i na tym analiza danego pakietu zostaje **zakończona**. Jeżeli takich reguł nie ma, to poszukiwane są reguły o typie kolejnego priorytetu i rozumowanie się powtarza. Bodaj najistotniejszą cechą tego algorytmu jest fakt, iż w razie istnienia reguł typu o wyższym priorytecie nie są wykonywane reguły typów o niższych priorytetach.

Przykład:

Przy poniższej regule, w razie pojawienia się jakiegokolwiek pakietu ICMP, pakiet zostanie zapisany do logów pakietów:

```
log icmp any any <> any any
```

Przy poniższym zestawie reguł:

```
log icmp any any <> any any
alert icmp any any <> any any (sid:10001; msg:"Regula test-alert";)
```

komunikat pojawi się w ostrzeżeniach, a pakiet pojawi się w logu pakietów, ale nie ze względu na regułę typu `log`, tylko na skutek działania reguły typu `alert`; zgodnie z przytoczoną powyżej zasadą reguła typu `log`, wobec obecności reguły typu `alert` o wyższym priorytecie, nie zostanie wykonana. Warto zauważyć, że kolejność zapisu reguł nie odgrywa tu roli.

Przy poniższym zestawie reguł:

```
log icmp any any <> any any
pass icmp any any <> any any
alert icmp any any <> any any (sid:10001; msg:"Regula test-alert";)
```

nie zostanie wykonana ani reguła typu `alert`, ani reguła typu `log`, i w konsekwencji żadne informacje nie zostaną zapisane.

Zmiana zasad przetwarzania reguł

Korzystając z dyrektyw konfiguracyjnych i opcji uruchomienia polecenia `snort` można wpływać na zasady – w szczególności kolejność – przetwarzania reguł.

Opcja `--alert-before-pass` zmienia domyślną hierarchię priorytetów na taką jaka była stosowana we wcześniejszych wersjach *Snort*, tzn. najwyższy priorytet mają reguły typu `alert`, a dalej `pass` i `log`. Opcja ta jest przydatna, jeżeli zapisana reguła typu `alert` nie daje żadnych efektów, wówczas używając opcji `--alert-before-pass` można łatwo sprawdzić, czy przyczyną nie jest występowanie reguły typu `pass` o pokrywających się kryteriach.

Opcja `--process-all-events` powoduje, że po wykonaniu reguł o wyższym priorytecie przetwarzanie pakietu nie jest zakończone, lecz następuje wykonanie reguł o niższym priorytecie, tak więc przy poniższym zestawie reguł:

```
log icmp any any <> any any
alert icmp any any <> any any (sid:10001; msg:"Regula test-alert";)
```

komunikat pojawi się w ostrzeżeniach, a pakiet pojawi się w logu pakietów dwukrotnie, najpierw na skutek działania reguły typu `alert`, a następnie na skutek działania reguły typu `log`.

Działanie opcji --process-all-events nie obejmuje reguł typu pass; wykonanie reguły tego typu zawsze kończy przetwarzanie pakietu.

Dyrektywa konfiguracyjna:

```
config order: typ1 typ2 typ3 ...
```

umożliwia określenie dowolnej hierarchii priorytetów typów reguł. Typy nie wymienione na liście są ustawiane w hierarchii za typami wymienionymi na liście, zgodnie ze swoimi domyślnymi priorytetami.

Przykład: `config order: log alert pass`

Dyrektywa:

```
config default_rule_state: enabled/disabled
```

powoduje ogólne włączenie lub wyłączenie przetwarzania reguł; domyślnie przetwarzanie reguł jest włączone.

Opcje reguły

Opcje reguły są jej ostatnim elementem, nie zawsze występującym. Opcje zapisywane są w postaci listy umieszczonej w nawiasach okrągłych; każda opcja jest postaci `nazwa:wartość;`.

Przykład (dwie opcje o nazwach: `sid` i `msg`):

```
(sid:10001; msg:"Regula test-alert";)
```

Ze względu na znaczenie opcje reguł dzielimy na parametry i kryteria szczegółowe.

Parametry reguły są wykorzystywane przede wszystkim przez reguły typu alert, gdzie ustalają informacje zawarte w ostrzeżeniu; obowiązkowe jest w tym przypadku określenie co najmniej identyfikatora SID ostrzeżenia.

Notatki

Kryteria szczegółowe reguły służą do precyzyjnego określenia, czy analizowany pakiet (wstępnie wyselekcjonowany poprzez adresację reguły) pasuje do tej reguły. Kryteria dzielą sie na dwie grupy: oparte o nagłówek pakietu (kryteria nagłówka) i oparte o zawartość segmentu danych pakietu (kryteria danych).

W przypadku zastosowania w regule wielu kryteriów mówimy, że pakiet pasuje do reguły, jeżeli spełnia WSZYSTKIE zdefiniowane kryteria.

Ze względu na bardzo dużą liczbę dostępnych opcji, w niniejszym opracowaniu zdecydowano się na wybranie tych, które są najczęściej wykorzystywane. Wybrane opcje stanowią ok. 75% ogólnej liczby. Opis wszystkich opcji znajduje się w dokumentacji Snort.

Parametry reguły

gid, sid, rev

Parametry te dotyczą reguł typu `alert` i określają odpowiednio identyfikator generatora GID, identyfikator zdarzenia SID i wersję reguły, z jakimi zostanie wygenerowane ostrzeżenie.

O ile ustalanie GID jest nieobowiązkowe (domyślny GID dla mechanizmu detektora ma wartość 1) i nawet nie zalecane, to dla każdej reguły typu alert obowiązkowe jest ustalenie wartości SID. W obu przypadkach zalecane jest używanie wartości większych niż 1000000 (milion), co gwarantuje, że nie nastąpi konflikt z wartościami używanymi przez mechanizmy *Snort* i standardowy zestaw reguł.

Przykłady:

```
gid:1000100;sid:1000101;      rev:2;
```

msg

Parametr ten ustala komunikat ostrzeżenia.

Przykład:

```
msg:"Komunikat ostrzezenia";
```

reference

Parametr ten wykorzystuje wbudowany w *Snort* mechanizm referencji do baz danych informacji o zdarzeniach. Referencje są definiowane poprzez użycie dyrektywy konfiguracyjnej:

```
config reference: nazwa URL
```

W standardowej konfiguracji referencje są zdefiniowane w pliku */etc/snort/references.config*. Poniżej prezentujemy wpis z tego pliku definiujący referencję do bazy *bugtraq*:

```
# grep bugtraq /etc/snort/reference.config
config reference: bugtraq   http://www.securityfocus.com/bid/
```

Parametr `reference` przyjmuje dwie wartości: nazwę bazy oraz identyfikator zdarzenia w tej bazie. Identyfikator jest dołączany na końcu URL bazy, i utworzony w ten sposób URL jest dołączany do wygenerowanego ostrzeżenia.

Przykład: Jeżeli reguła posiada następujący parametr:

```
reference: bugtraq,4088;
```

to w zależności od zastosowanego trybu zapisu ostrzeżenie będzie zawierało informację: `[Xref => http://www.securityfocus.com/bid/4088]` w trybie `full` lub `bugtraq,4088` w trybie `fast`.

classtype

Parametr ten wykorzystuje wbudowany w *Snort* mechanizm klasyfikacji zdarzeń, pozwalający na proste przypisywanie zdarzenia do danej klasy; z przynależności do klasy zdarzeń wynika priorytet zdarzenia, a opis jest dołączany do ostrzeżenia.

Klasy zdarzeń są definiowane poprzez użycie dyrektywy konfiguracyjnej:

```
config classification: nazwa,opis,priorytet
```

W standardowej konfiguracji referencje są zdefiniowane w pliku */etc/snort/classification.config*. Poniżej prezentujemy wpisy z tego pliku definiujące klasy ataków typu *Denial of Service*:

```
# grep dos /etc/snort/classification.config
config classification: attempted-dos,Attempted Denial of Service,2
config classification: successful-dos,Denial of Service,2
```

Parametr `classtype` przyjmuje wartość nazwy klasy.

Przykład: Jeżeli reguła posiada następujący parametr:

```
classtype:attempted-dos;
```

to ostrzeżenie będzie zawierało informację: `[Classification: Attempted Denial of Service]`, a zdarzenie będzie miało priorytet równy 2.

priority

Parametr ten ustala wartość priorytetu zdarzenia. Jeżeli użyty jest jednocześnie parametr `classtype` i `priority`, to ostateczną wartością priorytetu będzie ta ustalona przez parametr `priority`.

Przykład:

```
priority:10
```

Kryteria nagłówka

sameip

Proste kryterium nie posiadające wartości, spełnione jeśli adres źródłowy i adres docelowy pakietu są takie same.

Przykład:

```
sameip;
```

ttl

Kryterium to bada wartość pola TTL (*Time To Live*) pakietu.

Przykłady:

`ttl:64;` - spełnione, jeśli TTL pakietu jest równe 64.

`ttl:>1;` - spełnione, jeśli TTL pakietu jest większe niż 1.

`ttl:<=128;` - spełnione, jeśli TTL pakietu jest nie większe niż 128.

`ttl:64-128;` - spełnione, jeśli TTL pakietu jest z zakresu od 64 do 128 włącznie.

tos

Kryterium to bada wartość pola TOS (*Type Of Service*) pakietu.

Przykłady:

`tos:2;` - spełnione, jeśli TOS pakietu jest równe 2.

`tos:!2;` - spełnione, jeśli TOS pakietu jest różne od 2.

ip_proto

Kryterium to bada wartość pola protokołu pakietu. W przypadku protokołów TCP, UDP i ICMP można w tym celu użyć wartości protokołu dla reguły, zaś dla pozostałych protokołów należy ustalić dla reguły protokół IP i użyć kryterium `ip_proto`. Dodatkowo kryterium to pozwala na użycie operatora negacji oraz nierówności. Poszczególne protokoły mogą być wyrażane wartościami liczbowymi lub nazwami zgodnie z numeracją w pliku */etc/protocols*.

Przykłady:

`ip_proto:igmp;` - spełnione dla pakietów protokołu IGMP.

`ip_proto:!igmp;` - spełnione dla pakietów protokołów innych niż IGMP.

`ip_proto:>128;` - spełnione, jeśli numer protokołu jest większy niż 128.

`ip_proto:<128;` - spełnione, jeśli numer protokołu jest mniejszy niż 128.

Przykłady użycia:

`log ip any any -> any any (ip_proto:igmp;)` - zapis do logów pakietów protokołu IGMP.

`log ip any any -> any any (ip_proto:!tcp;)` - zapis do logów pakietów protokołów innych niż TCP.

dsize

Kryterium to bada długość segmentu danych pakietu (liczoną w bajtach). Badanie to jest użyteczne w połączeniu z innymi kryteriami pozwalającymi stwierdzić, że dane niesione przez pakiet stanowią komendę protokołu sieciowego, wówczas stwierdzając zbyt duży rozmiar segmentu danych można wnioskować, iż jest to próba ataku poprzez przepełnienie bufora serwera sieciowego.

Przykłady:

Notatki

`dsize:64;` - spełnione, jeśli segment danych pakietu ma długość 64.

`dsize:>400;` - spełnione, jeśli segment danych pakietu ma długość większą niż 400.

`dsize:<1024;` - spełnione, jeśli segment danych pakietu ma długość mniejszą niż 1024.

`dsize:512<>1024;` - spełnione, jeśli segment danych pakietu ma długość od 512 do 1024.

Kryterium to nie działa prawidłowo (jest zawsze nie spełnione) dla datagramów TCP składanych z fragmentów przesyłanych oddzielnymi pakietami IP.

id

Kryterium to bada wartość pola ID pakietu.

Przykład:

`id:31337;` (wartość często używana w narzędziach używanych przez atakujących).

fragoffset, fragbits

Kryteria te badają pola pakietu odpowiedzialne za obsługę dzielenia pakietów na fragmenty przesyłane przez warstwę łącza danych.

Kryterium `fragoffset` bada wartość pola przesunięcia fragmentu, wyznaczająca umiejscowienie fragmentu w całości pakietu. Oprócz przyrównania do podanej wartości obsługiwane są też operatory nierówności < i >.

Kryterium `fragbits` bada stan flag: M (*More Fragments*), D (*Don't Fragment*) i R (bit rezerwowy) pakietu. Wartością kryterium jest lista flag i opcjonalny modyfikator. Jeżeli nie użyto modyfikatora, to dla spełnienia kryterium flagi wymienione na liście mają być ustawione, a pozostałe wyzerowane; modyfikator `*` oznacza, że którakolwiek z flag z listy ma być ustawiona; modyfikator `+` oznacza, że oprócz flag z listy mogą być ustawione dowolne pozostałe flagi; modyfikator `!` oznacza, że żadna z flag z listy NIE ma być ustawiona.

Przykłady:

`fragoffset:0;` - spełnione, jeśli przesunięcie fragmentu jest równe 0 (oznacza to pierwszy fragment lub brak fragmentacji).

`fragoffset:>0;` - spełnione, jeśli przesunięcie fragmentu jest większe niż 0.

`fragbits:MD+;` - spełnione, jeśli ustawione są jednocześnie flagi *More Fragments* i *Don't Fragment* (sytuacja nieprawidłowa!).

`fragoffset:0; fragbits:M;` - para kryteriów jednoznacznie identyfikująca pierwszy fragment sfragmentowanego pakietu.

flags, seq, ack, window

Kryteria te odnoszą się do stanu pól nagłówka protokołu TCP i w sposób oczywisty mogą być spełnione tylko przez datagramy tego protokołu.

Kryterium `flags` to bada ustawienie flag TCP pakietu. Składnia tego kryterium jest następująca:

`flags:[modyfikator]flagi_sprawdzane[,flagi_ignorowane];`

Poszczególne flagi są reprezentowane na liście przez następujące identyfikatory: F - flaga FIN; S - SYN; R - RST; A - ACK; P - PSH; U - URG; 1, 2 - bity zarezerwowane; 0 - wartość specjalna oznaczająca pustą listę.

Jeżeli nie został zastosowany modyfikator, to kryterium jest spełnione, jeżeli wszystkie flagi sprawdzane są ustawione, a wszystkie pozostałe flagi, które nie znajdują się na liście flag ignorowanych, są wyzerowane. Bardzo często na liście flag ignorowanych znajdują się oba bity rezerwowe (wartość listy `12`); bity te są używane przez rozszerzenia protokołu TCP (np. jako flaga ECN - *Explicit Congestion Notification*) i z reguły nie są istotne dla wykrywania zdarzeń.

Przykłady:

`flags:0,12;` - spełnione, jeśli wszystkie flagi standardowe są wyzerowane (bity rezerwowe są ignorowane).

`flags:S;` - spełnione, jeśli flaga SYN jest ustawiona, a wszystkie pozostałe flagi, włącznie z bitami rezerwowymi, są wyzerowane.

Użycie modyfikatorów ma identyczny sens, jak w przypadku kryterium `fragbits`, tzn. modyfikator `*` oznacza, że którakolwiek z flag z listy ma być ustawiona; modyfikator `+` oznacza, że oprócz flag z listy mogą być ustawione dowolne pozostałe flagi (może być więc użyty zamiast wymieniania pozostałych flag na liście flag ignorowanych); modyfikator `!` oznacza, że żadna z flag z listy NIE ma być ustawiona.

Przykład:

`flags:+SF;` - spełnione, jeżeli ustawione są jednocześnie flagi SYN i FIN (sytuacja nieprawidłowa!) bez względu na stan pozostałych flag.

Kryteria `seq`, `ack` i `window` przyjmują wartości liczbowe i służą do badania odpowiednio: numeru sekwencji, numeru potwierdzenia i rozmiaru okna. W przypadku rozmiaru okna możliwe jest użycie operatora negacji.

Przykłady:

`seq:20;` - numer sekwencji równy 20.

`ack:1;` - numer potwierdzenia równy 1.

`window:!32768;` - rozmiar okna inny niż 32768.

flow, flowbits

Kryteria te odnoszą się do śledzenia sesji TCP i UDP i wymagają włączenia preprocesora śledzącego sesje (para *stream4/flow* lub *stream5*).

Składnia kryterium `flow` jest następująca:

```
flow [stan_sesji][,][ kierunek][,][ fragmentacja];
```

przy czym musi wystąpić przynajmniej jeden z warunków.

Warunek `stanu sesji` posiada dwie wartości: `established` - oznacza pakiety przesyłane w ramach ustanowionych sesji TCP/UDP oraz `stateless` - oznacza wszystkie pakiety, włącznie z tymi, które nie należą do żadnej ustanowionej sesji ani nie nawiązują nowej.

Warunek `kierunku` pozwala określić kierunek ruchu pakietu względem pary: klient-serwer. Jako klient traktowana jest strona aktywna, czyli ten system, który nawiązał sesję; jako serwer traktowana jest strona pasywna. Warunek ten posiada dwie wartości, każda jest reprezentowana przez dwa synonimy: `to_server` lub `from_client` oznacza ruch od klienta do serwera (w nomenklaturze oryginalnej *client request*), zaś `from_server` lub `to_client` - ruch od serwera do klienta (*server response*).

Warunek `fragmentacji` pozwala stwierdzić, czy datagram TCP/UDP jest składany z wielu pakietów IP, i posiada wartości: `no_stream` i `only_stream`. Kryterium z wartością `no_stream` powinien być stosowany, jeżeli ma być użyte kryterium `dsize`.

Przykłady:

`flow: from_client;` - cały ruch od klienta do serwera (włącznie z nawiązaniem sesji):

`flow: established,from_client;` - cały ruch od klienta do serwera, ale w ramach już ustanowionej sesji:

Kryterium `flowbits` pozwala na badanie i ustawianie stanu sesji, do której należy pakiet, względem preprocesora śledzącego sesje. Jego używanie wymaga dogłębnej znajomości działania preprocesora; zainteresowanych odsyłamy do dokumentacji *Snort.*

itype, icode, icmp_id, icmp_seq

Kryteria te odnoszą się do pakietów ICMP i w sposób oczywisty mogą być spełnione tylko przez pakiety tego protokołu. Kryteria `itype` i `icode` odnoszą się do typu i kodu ICMP, kryterium `icmp_id` - do pola ID protokołu ICMP, zaś kryterium `icmp_seq` - do numeru sekwencyjnego pakietu ICMP.

Każde z tych kryteriów przyjmuje wartość liczbową, z tym że w przypadku kryteriów `itype` i `icode` dodatkowo możliwe jest stosowanie operatorów nierówności < i > oraz zakresu <> identycznie, jak w przypadku kryterium `dsize`.

Kryteria danych

Kryteria danych są tak naprawdę najsilniejszym narzędziem w wykrywaniu zdarzeń. Zauważmy bowiem, że niepożądany ruch, który można wykryć na bazie zawartości nagłówków pakietów i/lub analizy stanu sesji protokołów warstwy transportowej, można równie łatwo zatrzymywać korzystając z filtra pakietów *netfilter,* tym samym usuwając potencjalne zagrożenie. Znacznie trudniej natomiast jest postępować w przypadku, gdy atakujący korzysta z usługi, która w zamierzeniu jest dla niego dostępna (np. publiczny serwer HTTP), natomiast atak polega na przesyłaniu do serwera specjalnie spreparowanych danych, mających spowodować pożądaną przez atakującego reakcję serwera. Ze względu na możliwość wystąpienia fałszywych alarmów (sytuacji, w których prawidłowa wymiana danych mogłaby być uznana za próbę ataku), rzadko podejmuje się decyzję o filtrowaniu ruchu na bazie zawartości segmentu danych, poprzestając jedynie na generowaniu ostrzeżeń.

Notatki

Kryteria danych w regułach *Snort* można generalnie podzielić na dwie grupy: kryteria tekstowe i kryteria binarne. W niniejszym opracowaniu przedstawiono kryteria tekstowe, które są najpowszechniej używane; kryteria binarne wymagają bardzo dogłębnej znajomości analizowanego protokołu i nie zawsze są użyteczne.

Oczywiście sama znajomość zasad działania kryteriów jest tylko techniczną podstawą do tworzenia własnych reguł bazujących na zawartości segmentu danych. Znacznie ważniejsza jest w tym przypadku znajomość rozmaitych typów ataków oraz wzorców danych charakterystycznych dla każdego z nich; jest to wiedza bardzo rozległa, wymagająca wąskiego wyspecjalizowania i z pewnością niemożliwa do przekazania w formie podręcznikowej. Z tego powodu codzienną praktyką administratora powinno być raczej bieżące uaktualnianie i ew. dostosowywanie reguł standardowych *Snort* niż tworzenie własnych reguł.

content

Kryterium `content` jest najpowszechniej występującym kryterium reguł *Snort.* Umożliwia ono przeszukanie segmentu danych pod kątem wystąpienia sekwencji danych, wyrażonej w postaci znakowej, binarnej lub mieszanej. Jeżeli sekwencja znajduje się wśród danych, kryterium jest spełnione.

! *Należy zwrócić uwagę na to, że dane analizowane przez kryterium content są wcześniej przetwarzane przez preprocesory, które często dokonują modyfikacji polegających zwłaszcza na normalizacji danych protokołu warstwy aplikacji (przykładem jest tu normalizacja dokonywana przez preprocesor http_inspect).*
Normalizacja na ogół ułatwia zapisanie reguły i duża część reguł standardowych bazuje na założeniu, że odpowiednie preprocesory będą uruchomione. Czasem jednak potrzebne jest przeszukanie danych nie przetworzonych przez preprocesory, należy wówczas posłużyć się modyfikatorem raw_bytes.

Kryterium `content` może wystąpić wielokrotnie, wówczas wymagane jest odnalezienie wszystkich wymienionych sekwencji.

Szereg wymienionych dalej w tym rozdziale kryteriów nie stanowi samodzielnych warunków, lecz stanowi modyfikatory działania kryterium `content`. Działanie modyfikatorów ograniczone jest do reguły, w której występują.

Wartością kryterium `content` jest wyszukiwana sekwencja znaków, w którą, w dowolnym miejscu, może być wpleciona sekwencja (lub wiele sekwencji) wartości binarnych, ujęta w parę znaków `|`.

Przykłady:

`content:"root";` - wyszukanie w segmencie danych słowa `root`.

`content:"root|00|toor";` - wyszukanie w segmencie danych słów `root` i `toor` przedzielonych bajtem o wartości 0.

`content:"root|08 0a 0d ff|";` - wyszukanie w segmencie danych słowa `root,` po którym następuje sekwencja bajtów o wartościach: 8, 10, 13 i 255.

Do wartości wyszukiwanej sekwencji można zastosować operator negacji, wówczas pakiet będzie spełniał kryterium, jeżeli sekwencja NIE zostanie odnaleziona w segmencie danych pakietu.

Przykłady:

```
content:!"root";
```

nocase

Jest to bezparametrowy modyfikator powodujący, że wyszukiwanie sekwencji tekstowych przez kryterium `content` odbywa się z ignorowaniem wielkości znaków.

Przykład:

```
content:"PASSWD"; nocase;
```

raw_bytes

Jest to bezparametrowy modyfikator powodujący, że wyszukiwanie sekwencji przez kryterium `content` odbywa się w oryginalnym segmencie danych pakietu, nie przetworzonym przez preprocesory.

Przykłady:

```
content:"/../"; raw_bytes;
```

http_client_body, http_uri, uricontent, urilen

Kryteria te odnoszą się do danych protokołu HTTP i wymagają działania preprocesora *http_inspect*.

`http_client_body` to bezparametrowy modyfikator powodujący, że wyszukiwanie sekwencji przez kryterium content odbywa się w ciele żądania klienta HTTP. Należy zauważyć, że przeszukiwanie nie odbywa się w ciele odpowiedzi serwera HTTP, inaczej mówiąc - modyfikator ten NIE powoduje wyszukiwania sekwencji w treści stron WWW.

`http_uri` to bezparametrowy modyfikator powodujący, że wyszukiwanie sekwencji przez kryterium `content` odbywa się w znormalizowanym przez preprocesor *http_inspect* URI żądania HTTP.

Kryterium `uricontent` jest niczym innym, jak skróconą wersją zapisu kryterium `content` z modyfikatorem `http_uri`.

Kryterium `urilen` przyjmuje wartość liczbową, która jest porównywana z długością znormalizowanego przez preprocesor *http_inspect* URI żądania HTTP. Dodatkowo możliwe jest stosowanie operatorów nierówności < i > oraz zakresu <> identycznie, jak w przypadku kryterium `dsize`.

Przykłady:

```
content:"pattern"; http_client_body;
content:"zsk.p.lodz.pl"; http_uri;
uricontent:"zsk.p.lodz.pl";
urilen:10<>20;
```

depth, offset

Modyfikatory te przyjmują wartości liczbowe i oznaczają odpowiednio: zakres liczony w bajtach od początku segmentu danych oraz przesunięcie liczone w bajtach od początku segmentu danych. Wartości te wyznaczają obszar, w którym kryterium `content` dokonuje wyszukiwania sekwencji.

Przykłady:

`content:"pattern"; depth:200;` - przeszukiwanie od początku segmentu danych do 200-go bajtu.

`content:"pattern"; offset:100;` - przeszukiwanie od 100-go bajtu do końca segmentu danych.

`content:"pattern"; depth:200; offeset:100` - przeszukiwanie od 100-go do 200-go bajtu segmentu danych.

distance, within

Modyfikatory te ustalają odpowiednio początek i koniec obszaru danych do przeszukiwania przez kryterium `content`. W odróżnieniu od modyfikatorów `depth` i `offset` obszar nie jest ustalony względem początku segmentu danych, ale względem końca sekwencji znalezionej przez poprzednie kryterium `content`. Wynika z tego, że modyfikatory te muszą być umieszczone pomiędzy dwoma kryteriami `content`; bazują one na kryterium poprzedzającym, a wpływają na kryterium następne.

Przykład:

```
content:"linux"; distance:10; within:20; content:"windows";
```

Powyższy zapis oznacza wyszukanie wzorca `linux`, i jeżeli zostanie on znaleziony, wyszukanie wzorca `windows` w obszarze od 10-go do 20-go bajtu licząc od końca znalezionej sekwencji `linux`.

pcre

Kryterium to dokonuje przeszukania zawartości segmentu danych podobnie jak `content` z tym, że wzorzec jest tu określany w formie wyrażenia regularnego zgodnego z PCRE (http://www.pcre.org). Do wzorca może zostać zastosowany operator negacji o znaczeniu identycznym, jak w przypadku kryterium `content`.

Określenie wzorca wyszukiwania w postaci wyrażenia regularnego z jednej strony umożliwia tworzenie bardzo wyrafinowanych kryteriów, z drugiej jednak - wymaga skomplikowanych algorytmów wyszukiwania, dużo mniej wydajnych w porównaniu z prostym wyszukiwaniem sekwencji. Dlatego też, o ile istnieje taka możliwość, należy używać raczej kryterium `content` do przeszukiwania segmentu danych.

Omawianie wyrażeń regularnych znacznie wykracza poza tematykę niniejszego opracowania, natomiast warto wspomnieć, iż w prostych przypadkach wyrażenie takie ma postać:

```
/wzorzec/opcje
```

Wśród opcji zawartych w standardzie PCRE najczęściej używana jest opcja `i`, powodująca, że wyszukiwanie odbywa się ze zignorowaniem wielkości znaków. Interpreter wyrażeń regularnych zawarty w *Snort* umożliwia stosowanie czterech dodatkowych opcji - `R`, `U`, `P` i `B` - nie należących do standardu PCRE. Opcja `R` oznacza, że wyszukiwanie odbędzie się od końca wzorca znalezionego przez poprzednie wystąpienie kryterium `pcre` (w ramach tej samej reguły); jest to odpowiednik modyfikatora `distance:0` dla kryterium `content`. Opcja `U` oznacza, że wyszukiwanie odbędzie się w URI żądania HTTP (odpowiednik `http_uri`). Opcja `P` oznacza, że wyszukiwanie odbędzie się w ciele żądania HTTP

Notatki

(odpowiednik `http_client_body`). Opcja `B` oznacza, że wyszukiwanie odbędzie się w oryginalnym segmencie danych pakietu, nie przetworzonym przez preprocesory (odpowiednik `raw_bytes`).

Przykład:

`pcre:"/linux/iB";` - wyszukanie wzorca `linux` w oryginalnym segmencie danych pakietu, ze zignorowaniem wielkości znaków.

Zapisywanie sekwencji pakietów

Niekiedy zachodzi potrzeba, aby po wykryciu zdarzenia zapisać do logów nie tylko samo zdarzenie, ale także dalszą historię wymiany pakietów. Zachowanie tych danych umożliwia następnie np. prześledzenie dalszej aktywności atakującego. *Snort* umożliwia taki zapis zarówno w ramach danej sesji TCP czy UDP, jak i całej wymiany pakietów pomiędzy dwoma systemami.

session

Parametr ten, użyty w regule, powoduje, że w przypadku wykonania reguły do logu ostrzeżeń zostanie zapisane nie tylko samo ostrzeżenie, ale także zawartość następnych pakietów w ramach tej samej sesji TCP/UDP. Stosowanie tego parametru wymaga użycia mechanizmu śledzenia sesji, tj. pary preprocesorów *stream4/flow* albo preprocesora *stream5*.

Parametr ten przyjmuje wartość `printable` - do logu zapisywane są tylko znaki drukowalne lub `all` – do logu są zapisywane wszystkie znaki z tym, że znaki niedrukowalne są zapisywane w postaci szesnastkowej.

Przykład: śledzenie sesji Telnet:

```
log tcp any any <> any 23 (session: printable;)
```

Warto zauważyć, że zapisywanie sesji powoduje dodatkowe obciążenie *Snort* i negatywnie wpływa na jego wydajność.

tag

Parametr ten użyty w regule typu `alert` powoduje, że po wykonaniu reguły analizowane są kolejne pakiety, i jeśli NIE spełniają kryteriów reguły, ale spełniają warunki określone przez parametr `tag`, to na podstawie tych pakietów tworzone są kolejne ostrzeżenia, a same pakiety zapisywane do logu pakietów. W przypadku zastosowania tego parametru do reguły typu `log` ma miejsce tylko zapisywanie kolejnych pakietów do logu pakietów.

Stosowanie tego parametru wymaga użycia mechanizmu śledzenia sesji, tj. pary preprocesorów *stream4/flow* albo preprocesora *stream5*.

Warto zauważyć, że ostrzeżenia stworzone na podstawie parametru `tag` mają odrębny typ (tzw. *tagged alerts*) i obsługa tego typu musi być uwzględniona w modułach realizujących zapis ostrzeżeń. Jak dotąd nie jest to zrealizowane w module umożliwiającym zapis ostrzeżeń do relacyjnych baz danych.

Parametr `tag` posiada następującą składnię:

```
tag: selektor,ilość,metryka[,źródło];
```

Selektor może przybrać jedną z wartości: `session` - wybór do zapisu następnych pakietów w ramach tej samej sesji TCP/UDP lub `host` – wybór na podstawie adresu źródłowego pakietu (a więc także np. ruch w innych sesjach pomiędzy systemami). W przypadku selektora `host` należy także określić źródło pakietów jako jedną z wartości: `src` - pakiety o tym samym adresie źródłowym co pakiet, który spowodował wykonanie reguły lub `dst` – pakiety o tym samym adresie docelowym co pakiet, który spowodował wykonanie reguły.

`Ilość` służy do ograniczenia liczby zapisywanych pakietów i jest wyrażona w jednostkach wskazanych metryką, która może mieć wartość `packets`, `seconds` lub `bytes`. Ograniczenie takie jest niezbędne, aby uniknąć ewentualnego ataku typu DoS wobec systemu IDS.

Niezależnie od określonej w regule ilości, jeżeli użyto metryki `seconds` albo `bytes`, uwzględniane jest dodatkowe ograniczenie w postaci dyrektywy:

```
config tagged_packet_limit: liczba
```

która ustala limit zapisywanych w ten sposób pakietów (wartość domyślna to 256). Zapis zostanie zakończony, jeżeli zostanie zapisana odpowiednia liczba pakietów LUB osiągnięta zostanie liczba bajtów lub czas zawarty w regule.

Przykład:

```
config tagged_packet_limit: 100;
log tcp any any <> any 23 (content:"root"; tag: session, 60,seconds;)
```

W powyższej konfiguracji, po wystąpieniu pakietu protokołu Telnet zawierającego słowo root, kolejne pakiety w tej sesji TCP będą zapisywane do logu pakietów przez 60 sekund chyba, że wcześniej zostanie wyczerpany limit 100 zapisanych pakietów.

6 Aplikacje pomocnicze

Istotnym aspektem posługiwania się systemami IDS jest możliwość przeprowadzania szczegółowych analiz danych wygenerowanych przez te systemy. Dlatego też oprócz implementacji samego systemu IDS należy rozważyć użycie aplikacji pomocniczych wspierających proces analizy.

W niniejszym rozdziale opisano dwie aplikacje wspomagające analizę danych generowanych przez sieciowy IDS *Snort:* analizator pakietów *barnyard* oraz kompleksową aplikację analizującą ostrzeżenia *BASE*.

Barnyard2

Barnyard2 jest specjalizowanym analizatorem logów umożliwiającym przeglądanie logów ostrzeżeń oraz pakietów zapisanych w formacie zunifikowanym (*unified).*

! *Analizator barnyard2 NIE odczytuje binarnego formatu zapisu pakietów zgodnego z tcpdump (pcap). Próba analizy takiego pliku kończy się komunikatem o naruszeniu ochrony pamięci.*

Analizator *barnyard2* pracuje w dwóch trybach: **wsadowym** (ang. *batch processing*), w którym dokonuje jednokrotnej analizy wskazanego pliku, oraz **ciągłym** (ang. *continual processing*), w którym monitoruje pliki znajdujące się we wskazanym katalogu i przetwarza nowo pojawiające się wpisy.

Instalacja

Analizator *barnyard2* można pobrać z repozytorium GitHub, gałąź *stable* zawierająca ostatnią stabilną wersje programu jest dostępna jako: *https://github.com/firnsy/barnyard2/tree/stable*.

Kompilacja samego programu wymaga pakietów: *libtool* oraz *libpcap-devel*. W przypadku obsługi bazy danych niezbędne są też biblioteki dla tej bazy, np. *postgresql-devel*.

Pierwszym krokiem po pobraniu gałęzi z repozytorium / rozpakowaniu archiwum jest przejście do katalogu głównego aplikacji i uruchomienie skryptu *autogen.sh*:

```
# cd barnyard2-stable
# ./autogen.sh
Found libtoolize
autoreconf: Entering directory `.'
...
autoreconf: Leaving directory `.'
You can now run "./configure" and then "make".
```

Następnie należy dokonać konfiguracji przed kompilacją. Na tym etapie następuje wybór funkcjonalności, które mają zostać wkompilowane w program *barnyard2* – w niniejszym przykładzie będzie to obsługa bazy PostgreSQL.

```
# ./configure --with-postgresql
...
checking pcap.h usability... yes
checking pcap.h presence... yes
checking for pcap.h... yes
checking for pcap_datalink in -lpcap... yes
checking for sparc... no
checking for postgresql... yes
...
```

Po skonfigurowaniu można przystąpić do zbudowania i zainstalowania programu.

```
# make
make  all-recursive
make[1]: Wejście do katalogu `/root/barnyard2-stable'
Making all in src
make[2]: Wejście do katalogu `/root/barnyard2-stable/src'
Making all in sfutil
```

Notatki

```
make[3]: Wejście do katalogu `/root/barnyard2-stable/src/sfutil'
gcc -DHAVE_CONFIG_H -I. -I../.. -I..  -DENABLE_POSTGRESQL  -g -O2 -fno-strict-aliasing
-Wall -c getopt_long.c
...


# make install
...
 /bin/mkdir -p '/usr/local/bin'
  /bin/sh ../libtool   --mode=install /bin/install -c barnyard2 '/usr/local/bin'
libtool: install: /bin/install -c barnyard2 /usr/local/bin/barnyard2
...
test -e /usr/local/etc/barnyard2.conf || install -m 600 ../etc/barnyard2.conf
/usr/local/etc
...
```

W procesie instalacji zainstalowany zostaje program `barnyard2` oraz jego plik konfiguracyjny */usr/local/etc/barnyard2.conf*.

Konfiguracja

Konfiguracja *barnyard2* polega przede wszystkim na określeniu metod zapisu wyników analizy. W dalszej części rozdziału opisano kilka podstawowych metod zapisu wraz z zaprezentowaniem ich efektów.

Dyrektywa `output alert_fast[:plik]` włącza tekstowy zapis ostrzeżeń do wskazanego pliku (domyślnie: *fast.alert*). Zapisy w tym pliku mają postać jak w poniższym przykładzie:

```
------------------------------------------------------------------------
12/18/07-10:46:35.909349 {ICMP} 212.77.100.101 -> 192.168.0.201
[**] [1:10001:0] NETBIOS SMB tapisrv ClientRequest WriteAndX andx object call
LSetAppPriority overflow attempt [**]
[Classification: Unknown] [Priority: 0]
[Xref => http://www.securityfocus.com/bid/14518]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2005-0058]
[Xref => http://www.microsoft.com/technet/Security/bulletin/ms05-040.mspx]
------------------------------------------------------------------------
```

Warto przy tym wspomnieć, że *barnyard2* korzysta z mapowań wartości GID i SID oraz systemów klasyfikacji i referencji *Snort*.

Dyrektywa `output alert_syslog` włącza tekstowy zapis ostrzeżeń do logów systemowych. Zapisy mają postać jak w poniższym przykładzie:

```
# tail /var/log/messages
Dec  18 12:51:00 desktop barnyard: [1:10001:0] NETBIOS SMB tapisrv ClientRequest WriteAndX
andx object call LSetAppPriority overflow attempt [Classification: Unknown] [Priority: 0]
{ICMP} 192.168.0.201 -> 212.77.100.101
```

Dyrektywa `output log_dump[:plik]` włącza tekstowy zapis ostrzeżeń wraz z danymi pakietów do wskazanego pliku (domyślnie: *dump.log*). Jeżeli analizie zostanie poddany tylko plik ostrzeżeń lub plik logu pakietów, to wynikiem będzie tylko sekwencja ostrzeżeń lub danych pakietów. Jeżeli analizie zostanie poddany jednocześnie plik ostrzeżeń i log pakietów, to *barnyard2* powiąże ze sobą ostrzeżenie oraz pakiet, który je wywołał, i zapisze obie informacja w formie jednego wpisu, jak w poniższym przykładzie:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

[**] [1:10001:0] NETBIOS SMB tapisrv ClientRequest WriteAndX andx object call
LSetAppPriority overflow attempt [**]
[Classification: Unknown] [Priority: 0]
[Xref => http://www.securityfocus.com/bid/14518]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=2005-0058]
[Xref => http://www.microsoft.com/technet/Security/bulletin/ms05-040.mspx]
Event ID: 20     Event Reference: 20
01/02/08-10:46:35.909349 212.77.100.101 -> 192.168.0.201
ICMP TTL:121 TOS:0x0 ID:40475 IpLen:20 DgmLen:84
Type:0  Code:0  ID:8574  Seq:2560  ECHO REPLY
8B 6B 7B 47 4D 92 0D 00 08 09 0A 0B 0C 0D 0E 0F  .k{GM...........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Dyrektywa `output log_pcap[:plik]` włącza binarny zapis pakietów do wskazanego pliku (domyślnie: *barnyard.pcap*). Nazwa pliku jest uzupełniana o datę i czas jego utworzenia. Zapisy w tym pliku mają postać zgodną z formatem *pcap* i w celu ich zinterpretowania należy posłużyć się dowolnym analizatorem obsługującym ten format; w poniższym przykładzie użyto polecenia `tcpdump`:

```
# tcpdump -qnt -r barnyard.pcap.2007-12-18\@12-51-00
reading from file barnyard.pcap.2007-12-18@12-51-00, link-type EN10MB (Ethernet)
IP 192.168.0.201 > 212.77.100.101: ICMP echo request, id 32289, seq 1, length 64
IP 212.77.100.101 > 192.168.0.201: ICMP echo reply, id 32289, seq 1, length 64
```

Warto dodać, że poszczególne metody zapisywania nie wykluczają się wzajemnie.

Zapis do relacyjnej bazy danych

Jedną z ciekawszych możliwości oferowanych przez *Barnyard2* jest zapisywanie ostrzeżeń i logów pakietów do relacyjnych baz danych. Podstawową zaletą takiego formatu zapisu jest łatwość przeszukiwania zarchiwizowanych danych zarówno według kryteriów tekstowych, jak i numerycznych czy wyszukiwania ciągów binarnych. Metody dostępu do relacyjnych baz są bardzo dobrze zestandaryzowane, dzięki czemu praktycznie w każdym współczesnym języku programowania można stosunkowo łatwo stworzyć aplikację, która będzie przeszukiwać i analizować dane zapisane przez *Snort* i prezentować wyniki w postaci czytelnej dla użytkownika.

Wybierane w czasie kompilacji wtyczki umożliwiające zapisywanie przez *Barnyard2* do relacyjnych baz danych obsługują pięć typów systemów zarządzania bazami danych: MySQL, PostgreSQL, Oracle, Microsoft SQL Server oraz uniwersalny sterownik baz danych ODBC.

Zapis do bazy danych jest włączany w pliku konfiguracyjnym poprzez użycie dyrektyw:

```
output database: alert, typ_bazy, nazwa_bazy [opcje]
output database: log, typ_bazy, nazwa_bazy [opcje]
```

odpowiednio dla zapisywania ostrzeżeń i logów pakietów.

`Typ_bazy` określa typ wykorzystywanego systemu zarządzania bazami danych; możliwe wartości to `mysql`, `postgresql`, `oracle`, `mssql` i `odbc`.

Stosowanie pozostałych opcji zależy od konfiguracji używanej bazy danych. Z reguły potrzebne są co najmniej opcje identyfikujące serwer bazy danych (`host=nazwa_lub_adres`, `port=numer_portu`) oraz użytkownika bazodanowego, z którego konta będą wykonywane operacje zapisu (`user=nazwa_użytkownika`, `password=hasło_użytkownika`). Inne opcje określają szczegóły formatu zapisu danych i najczęściej ich domyślne wartości są odpowiednie; ważniejsze z nich to:

`detail=full` lub `detail=fast` - opcja dotyczy zapisywania logów pakietów i określa ilość informacji o pakiecie zapisywanych do bazy danych; w trybie `fast` zapisywane są tylko informacje z nagłówka pakietu, w trybie `full` - także dane przenoszone przez pakiet.

`sensor_name=nazwa` - schemat bazy danych używany przez *Snort* zakłada, że wiele instancji *Snort* będzie zapisywało informacje do jednej bazy; poszczególne wpisy oznaczane są nazwą określaną przez opcję `sensor_name`, dzięki czemu można je rozróżnić. Ustalanie tej opcji nie jest obowiązkowe, gdyż w razie jej braku *Snort* generuje ją automatycznie.

Przykład:

```
output database: alert, postgresql, host=localhost port=5432 user=snort_user password=pleasechange dbname=snort_logs
output database: log, postgresql, host=localhost port=5432 user=snort_user password=pleasechange dbname=snort_logs
```

Uruchamianie barnyard – tryb wsadowy

Aby uruchomić analizator *barnyard2* w trybie wsadowym, należy użyć polecenia:

```
barnyard2 [opcje] -o plik [plik2 plik3 ...]
```

Użycie opcji `-o` wskazuje na korzystanie z trybu wsadowego. Inne opcje, umożliwiające modyfikację działania analizatora, przedstawia poniższa tabela.

Tabela 5: Opcje polecenia barnyard2 w trybie wsadowym

Opcja	*Znaczenie*
`-c plik`	Wskazuje położenie pliku konfiguracyjnego barnyard (domyślnie */etc/snort/barnyard.conf*)
`-L katalog`	Określa katalog, w którym mają zostać zapisane wyniki analizy (domyślnie są one zapisywane w katalogu bieżącym).

Notatki

Opcja	*Znaczenie*
`-s plik`	Wskazuje plik, w którym przechowywane jest mapowanie par wartości SID i GID na komunikaty ostrzeżeń (domyślnie */etc/snort/gen-msg.map*).
`-g plik`	Wskazuje plik, w którym przechowywane jest mapowanie wartości GID na nazwy mechanizmów generujących ostrzeżenia (domyślnie */etc/snort/gen-msg.map*).
`-p plik`	Wskazuje plik, w którym przechowywane są klasyfikacje ostrzeżeń (domyślnie */etc/snort/classification.config*).

`-v`
`-vv`
`-vvv`
	Określa szczegółowość komunikatów wyświetlanych przez *barnyard* na wyjściu standardowym. Użycie tej opcji NIE powoduje, że ostrzeżenia czy dane pakietów zostaną wypisane na ekranie.

Uruchamianie barnyard2 – tryb ciągły

W trybie ciągłym analizator *Barnyard2* monitoruje wskazany katalog, poszukując w nim plików o zadanym wzorcu nazwy. Nowe zdarzenia zapisywane zarówno w istniejących, jak i nowo powstających plikach w formacie *unified*, sa na bieżąco przetwarzane przez analizator zgodnie z jego konfiguracją. Informacje o ostatnio przetworzonych zdarzeniach są przechowywane w specjalnym rejestrze, tzw. pliku *waldo*.

Uruchomienie w trybie ciągłym następuje po uruchomieniu programu *Barnyard2* bez opcji `-o`.

Tabela 6: Dodatkowe opcje i dyrektywy polecenia barnyard2 w trybie wsadowym

Opcja / dyrektywa	*Znaczenie*
`-D` `config daemon`	Uruchomienie w trybie daemona. Domyślnie *Barnyard2* jest uruchamiany w pierwszym planie.
`-w plik` `config waldo_file`	Określa lokalizację pliku *waldo*
`-n` `config process_new_records_only`	Nakazuje analizatorowi przetwarzanie tylko nowych zdarzeń (późniejszych niż zapis w pliku *waldo*)
`-d katalog`	Wskazuje katalog, w którym znajdują się zapisane pliki zdarzeń w formacie *unified*.
`-f wzorzec`	Określa wzorzec (prefiks) nazwy plików zdarzeń w formacie *unified*.

Przykład: obsługa bazy danych PostgreSQL

Konfiguracja bazy danych (PostgreSQL)

W rozdziale tym opisano najprostszą konfigurację bazy danych PostgreSQL dla działania BASE przy założeniu, że baza, Snort i BASE działają w tym samym systemie. Należy pamiętać, że konfiguracja ta NIE zapewnia odpowiedniego poziomu bezpieczeństwa samej bazy. Zainteresowanych odsyłamy do zajęć poświęconych konfiguracji baz danych PostgreSQL.

Na konfigurację bazy danych składa się utworzenie bazy, utworzenie użytkownika bazodanowego korzystającego z tej bazy oraz zaimportowanie schematu bazy. Pierwsze z tych czynności wykonywane są z konta specjalnego użytkownika systemowego *postgres* przeznaczonego do zarządzania bazami danych PostgreSQL, za pomocą powłoki administracyjnej PostgreSQL – `psql`. Zakładamy, że system zarządzania bazami PostgreSQL został zainstalowany i uruchomiony zgodnie z informacjami zawartymi w dedykowanym rozdziale.

Pierwszy etap to utworzenie bazy, użytkownika i nadanie mu wszystkich praw do bazy:

```
# su - postgres
-bash-4.2$ psql
psql (9.2.10)
Wpisz "help" by uzyskać pomoc.

postgres=# create database snort_logs;
CREATE DATABASE
postgres=#  create user snort_user PASSWORD 'pleasechange';
CREATE ROLE
postgres=# grant ALL ON DATABASE snort_logs to snort_user;
GRANT
```

Kolejnym etapem jest umożliwienie nowo stworzonemu użytkownikowi bazodanowemu *snort_user* uwierzytelnienia w bazie *snort_logs* za pomocą hasła. W tym celu korzystając z konta administratora należy umieścić poniższą linię w pliku */var/lib/pgsql/data/pg_hba.conf*:

```
host    snort_logs      snort_user   127.0.0.1/32            password
```

Wpis ten oznacza, że użytkownik bazodanowy *snort_user* może uwierzytelniać się na podstawie hasła, jeżeli łączy się używając lokalnego adresu 127.0.0.1.

Po wczytaniu nowej konfiguracji do serwera PostgreSQL:

```
# systemctl reload postgresql
```

należy uruchomić powłokę `psql` i uwierzytelnić się do bazy jako stworzony użytkownik bazodanowy:

```
# psql -U snort_user -h 127.0.0.1 snort_logs
Hasło użytkownika snort_user:
psql (9.2.10)
Wpisz "help" by uzyskać pomoc.
```

Dzięki nadaniu praw dla użytkownika może on wczytać do bazy skrypt wprowadzający wymagane struktury. Skrypt ten znajduje się w katalogu programu *Barnyard2*, w podkatalogu *schemas*:

```
snort_logs=> \i /usr/src/barnyard2-stable/schemas/create_postgresql
...
snort_logs=> \d
                          Lista relacji
 Schemat |                Nazwa                |    Typ    | Właściciel
---------+-------------------------------------+-----------+------------
 public  | data                                | tabela    | snort_user
 public  | detail                              | tabela    | snort_user
 public  | encoding                            | tabela    | snort_user
 public  | event                               | tabela    | snort_user
 public  | icmphdr                             | tabela    | snort_user
 public  | iphdr                               | tabela    | snort_user
 public  | opt                                 | tabela    | snort_user
 public  | reference                           | tabela    | snort_user
 public  | reference_ref_id_seq                | sekwencja | snort_user
 public  | reference_system                    | tabela    | snort_user
 public  | reference_system_ref_system_id_seq  | sekwencja | snort_user
 public  | schema                              | tabela    | snort_user
 public  | sensor                              | tabela    | snort_user
 public  | sensor_sid_seq                      | sekwencja | snort_user
 public  | sig_class                           | tabela    | snort_user
 public  | sig_class_sig_class_id_seq          | sekwencja | snort_user
 public  | sig_reference                       | tabela    | snort_user
 public  | signature                           | tabela    | snort_user
 public  | signature_sig_id_seq                | sekwencja | snort_user
 public  | tcphdr                              | tabela    | snort_user
 public  | udphdr                              | tabela    | snort_user
(21 wierszy)
```

Konfiguracja Snort

Jak wspomniano wcześniej, począwszy od wersji 2.9.3 obsługa baz danych nie jest już dostępna bezpośrednio w *Snort*. Zamiast tego należy skonfigurować *Snort* do zapisywania zdarzeń w formacie *unified*, a nastepnie wykorzystać *Barnyard2* do przenoszenia tych danych do bazy danych.

Konfiguracja *Snort* do zapisywania zdarzeń w formacie *unified* wymaga dwóch czynności. Pierwszą z nich jest wprowadzenie odpowiedniej konfiguracji w pliku *snort.conf*:

```
# grep "output unified" /etc/snort/snort.conf
output unified2: filename merged.log, limit 128
```

Druga czynność to deaktywowanie opcji `ALERTMODE`, a także wyłączenie opcji `BINARY_LOG` w pliku konfiguracyjnym usługi *snortd*, */etc/sysconfig/snort*:

```
# grep ALERTMODE /etc/sysconfig/snort
# ALERTMODE=fast

# grep BINARY_LOG /etc/sysconfig/snort
BINARY_LOG=0
```

Potwierdzeniem poprawnej konfiguracji jest uruchomienie programu *snort* bez opcji `-A` oraz `-b`. Jest to niezbędne dla uzyskania efektu:

```
# ps aux|grep snort
snort    10828  0.0 28.3 631464 289120 ?       Ssl  14:44   0:00 /usr/sbin/snort -d -D -i
```

Notatki

```
enp0s9 -u snort -g snort -c /etc/snort/snort.conf -l /var/log/snort
# file /var/log/snort/merged.log.1428455395
  /var/log/snort/merged.log.1428455395: data
```

Próbne odczytanie pliku zdarzeń za pomocą narzędzia *u2spew*:

```
# u2spewfoo /var/log/snort/merged.log.1428455395 |head

(Event)
      sensor id: 0 event id: 1  event second: 1428455407  event microsecond: 93664
      sig id: 12   gen id: 129  revision: 1   classification: 3
      priority: 2  ip source: 192.168.2.2    ip destination: 192.168.2.1
      src port: 50696     dest port: 22protocol: 6  impact_flag: 0      blocked: 0

Packet
      sensor id: 0 event id: 1  event second: 1428455407
      packet second: 1428411260 packet microsecond: 93664
```

Próbna analiza narzędziem *Barnyard2* w trybie wsadowym:

```
# barnyard2 -c /usr/local/etc/barnyard2.conf -o /var/log/snort/merged.log.1428455395
Running in Batch mode

        --== Initializing Barnyard2 ==--
Initializing Input Plugins!
Initializing Output Plugins!
Parsing config file "/usr/local/etc/barnyard2.conf"
Log directory = /tmp

        --== Initialization Complete ==--

  ______   -*> Barnyard2 <*-
 / ,,_  \  Version 2.1.10 (Build 310)
 |o"  )~|  By Ian Firns (SecurixLive): http://www.securixlive.com/
 + ''''' +  (C) Copyright 2008-2012 Ian Firns <firnsy@securixlive.com>


    spool directory = /var/log/snort/
    spool filebase  = snort.log
    time_stamp      = 1428455532
    record_idx      = 0
Processing 1 files...
Opened spool file '/var/log/snort/merged.log.1428455395'
04/07-12:54:20.093664  [**] [129:12:1] stream5: TCP Small Segment Threshold Exceeded [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.2.2:50696 ->
192.168.2.1:22
...
04/07-12:55:34.067815  [**] [129:15:1] stream5: Reset outside window [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.2.2:38471 -> 192.168.2.1:80
Closing spool file '/var/log/snort/merged.log.1428410641'. Read 60 records
===============================================================================
Record Totals:
   Records:          60
    Events:          30 (50.000%)
   Packets:          30 (50.000%)
   Unknown:           0 (0.000%)
...
```

Konfiguracja Barnyard2

Ustawiono następujące dyrektywy pliku konfiguracyjnego:

```
config waldo_file: /tmp/waldo
config process_new_records_only
output database: alert, postgresql, user=snort_user dbname=snort_logs password=pleasechange
host=127.0.0.1 sensor_name=snort1
```

Uruchomienie *Barnyard2* w trybie ciągłym. Plik *waldo* nie istnieje, zaś w pliku *merged.log.1428455395* istnieją zapisane zdarzenia zademonstrowane powyżej. Baza danych nie przechowuje obecnie żadnych zdarzeń:

```
snort_logs=> select * from event;
 sid | cid | signature | timestamp
-----+-----+-----------+-----------
(0 wierszy)
```

Uruchomienie *Barnyard2* w trybie ciągłym:

```
# barnyard2 -v -c /usr/local/etc/barnyard2.conf -d /var/log/snort/ -f merged.log
Running in Continuous mode
```

```
        --== Initializing Barnyard2 ==--
Initializing Input Plugins!
Initializing Output Plugins!
Parsing config file "/usr/local/etc/barnyard2.conf"
Log directory = /tmp
WARNING database: Defaulting Reconnect/Transaction Error limit to 10
WARNING database: Defaulting Reconnect sleep time to 5 second
database: compiled support for (postgresql)
database: configured to use postgresql
database: schema version = 107
database:           host = 127.0.0.1
database:           user = snort_user
database:  database name = snort_logs
database:    sensor name = snort1
database:      sensor id = 1
database:     sensor cid = 1
database:  data encoding = hex
database:   detail level = full
database:     ignore_bpf = no
database: using the "alert" facility

        --== Initialization Complete ==--

  ______    -*> Barnyard2 <*-
 / ,,_  \   Version 2.1.10 (Build 310)
 |o"  )~|   By Ian Firns (SecurixLive): http://www.securixlive.com/
 + '''' +   (C) Copyright 2008-2012 Ian Firns <firnsy@securixlive.com>

WARNING: Unable to open waldo file '/tmp/waldo' (No such file or directory)
Opened spool file '/var/log/snort//merged.log.1428455395'
...
Waiting for new data
```

W wyniku działania *Barnyard2* zrealizowane zostały dwie czynności. Przede wszystkim zdarzenia zawarte w pliku zostały przeniesione do bazy danych:

```
snort_logs=> select * from event;
 sid | cid | signature |         timestamp
-----+-----+-----------+---------------------------
   1 |   1 |       507 | 2015-04-08 01:13:42.796+02
   1 |   2 |       507 | 2015-04-08 01:13:42.906+02
 ...
   1 |  30 |       510 | 2015-04-08 01:14:43.39+02
   1 |  31 |       510 | 2015-04-08 01:14:57.589+02
(31 wierszy)
```

Ponadto został stworzony binarny plik *waldo*:

```
# file /tmp/waldo
/tmp/waldo: data
```

Barnyard2 oczekuje teraz na pojawienie się nowych zdarzeń w pliku. Zjawisko to mimo użycia opcji -v nie jest raportowane na wyjściu programu, ale można zauważyć pojawienie się nowych krotek w bazie danych:

```
snort_logs=> select * from event;
 sid | cid | signature |         timestamp
-----+-----+-----------+---------------------------
   1 |   1 |       507 | 2015-04-08 01:13:42.796+02
   1 |   2 |       507 | 2015-04-08 01:13:42.906+02
   ...
   1 |  37 |       508 | 2015-04-08 01:26:45.708+02
   1 |  38 |       508 | 2015-04-08 01:26:47.598+02
(38 wierszy)
```

Notatki

Barnyard2 monitoruje także pojawienie się nowych plików, co może zdarzyć się na skutek ponownego uruchomienia *Snort* lub osiągnięcia przez dotychczasowy plik maksymalnego rozmiaru:

```
Closing spool file '/var/log/snort//merged.log.1428455395'. Read 124 records
Opened spool file '/var/log/snort//merged.log.1428456578'
Waiting for new data
```

Aplikacja BASE

*BASE (*ang. *Basic Analysis and Security Engine*) jest aplikacją służącą do kompleksowej analizy ostrzeżeń i logów pakietów wygenerowanych przez sieciowy IDS Snort. Aplikacja ta jest utworzona w języku PHP obsługiwana za pomocą przeglądarki WWW, zaś do jej uruchomienia niezbędne są: serwer HTTP z interpreterem języka PHP (np. serwer *Apache* z modułem obsługi PHP), baza danych, w której przechowywane są ostrzeżenia i dane pakietów oraz biblioteki pomocnicze.

Instalacja zależności i konfiguracja PHP

Aplikacja BASE wymaga do działania szeregu bibliotek. Przede wszystkim wymagany jest język PHP (pakiet *php*), ale także szereg jego bibliotek zawartych w pakietach *php-pgsql*, *php-pear*, *php-gd*, *php-xml*.

Kolejną grupę zależności stanowią biblioteki instalowane poprzez własny instalator PHP o nazwie *pear*:

```
# pear install Image_Graph-alpha Image_Canvas-alpha Image_Color
...
install ok: channel://pear.php.net/Image_Color-1.0.4
install ok: channel://pear.php.net/Image_Canvas-0.3.5
install ok: channel://pear.php.net/Image_Graph-0.8.0
```

Ostatnią zależnością jest biblioteka abstrakcji bazy danych ADODB. Biblioteka nie jest dostępna w repozytoriach, ale można skorzystać z pakietu dla wcześniejszej dystrybucji CentOS6:

```
# rpm -ihv php-adodb-4.95-1.a.el6.noarch.rpm
ostrzeżenie: php-adodb-4.95-1.a.el6.noarch.rpm: Nagłówek V3 RSA/SHA256 Signature,
identyfikator klucza 0608b895: NOKEY
Przygotowywanie...                ################################# [100%]
Aktualizowanie/instalowanie...
   1:php-adodb-4.95-1.a.el6       ################################# [100%]
```

Następnie należy w pliku konfiguracyjnym PHP, */etc/php.ini*, ustawić globalny poziom ostrzeżeń PHP na poniższą wartość:

```
error_reporting = E_ALL & ~E_NOTICE
```

Po wykonaniu tych czynności można uruchomić serwer Apache:

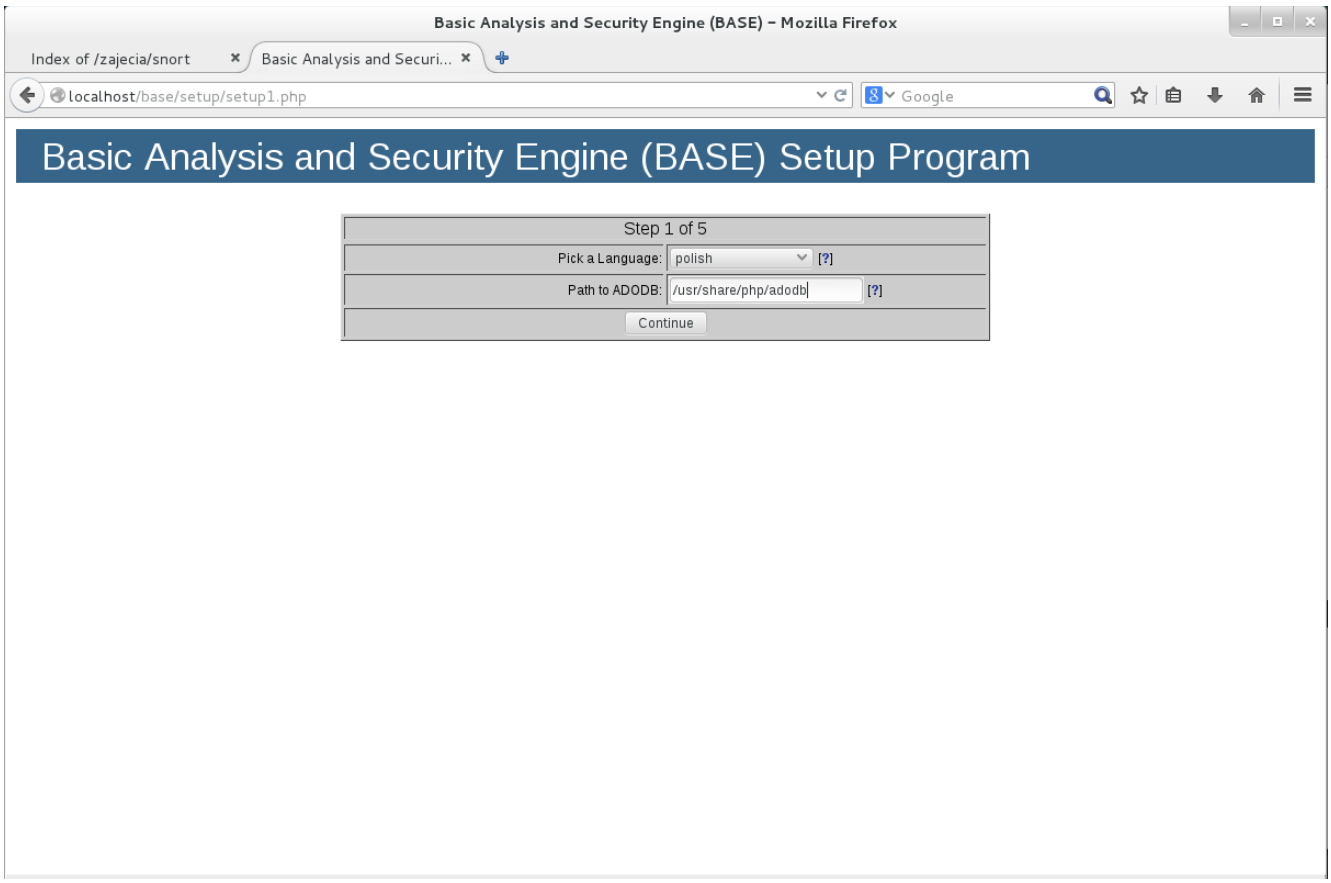
```
# systemctl start httpd
```

Instalacja i konfiguracja BASE

Aplikacja BASE dostępna jest do pobrania z witryny projektu, *http://base.professionallyevil.com/*. Aplikacja jest udostępniana w formie archiwum *.tar.gz*, którego zawartość należy wypakować do wybranego katalogu dostępnego dla serwera *Apache*; w tym przykładzie użyto katalogu */var/www/html/base*. Należy także odpowiednio ustalić właścicielstwo plików i katalogów, tak, aby serwer Apache miał do nich prawo zapisu; niezbędne to będzie dla procesu konfiguracji aplikacji.

```
# chown -R apache:apache /var/www/html/base
```

Następną czynnością jest uruchomienie aplikacji poprzez przeglądarkę WWW z adresem: *http://localhost/base* (w tym przykładzie przeglądarka jest uruchamiana na tym samym systemie, na którym działa aplikacja). Ponieważ aplikacja nie została skonfigurowana, uruchomiony zostanie proces konfiguracji, w którym w pierwszym etapie należy określić język, w którym ma wyświetlać komunikaty aplikacja (dostępny jest język polski), oraz położenie bibliotek *adodb*:

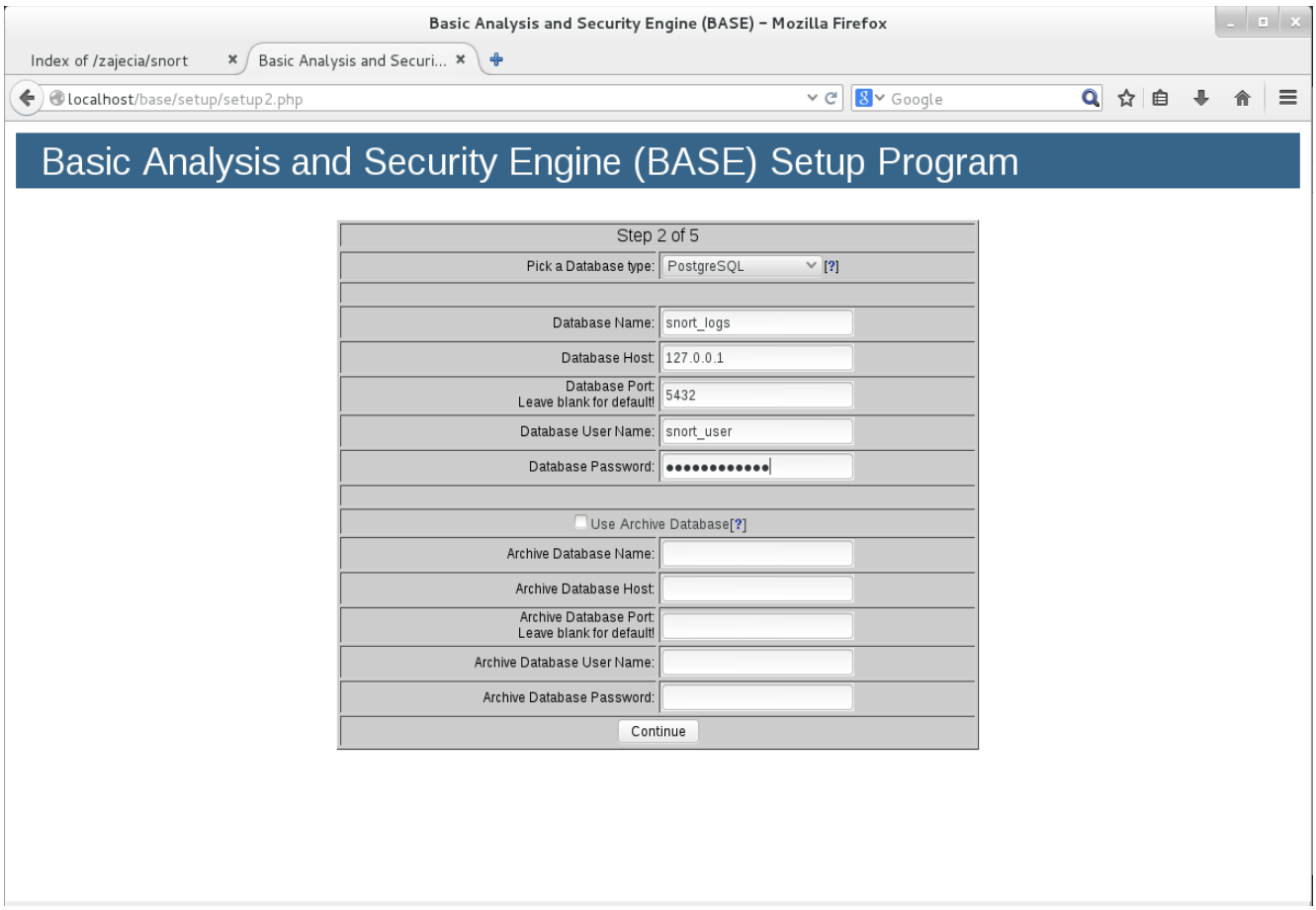


Korzystając z polecenia rpm łatwo ustalić położenie bibliotek *adodb* (*/usr/share/php/adodb*):

```
# rpm -ql php-adodb
...
/usr/share/php/adodb
...
```

W następnym kroku należy skonfigurować parametry używanej bazy danych:

Notatki



Kolejny etap to skonfigurowanie kontroli dostępu do aplikacji *BASE*. Z uwierzytelniania powinno się zrezygnować tylko wtedy, gdy dostęp do aplikacji będzie ograniczony tylko do tego samego systemu.

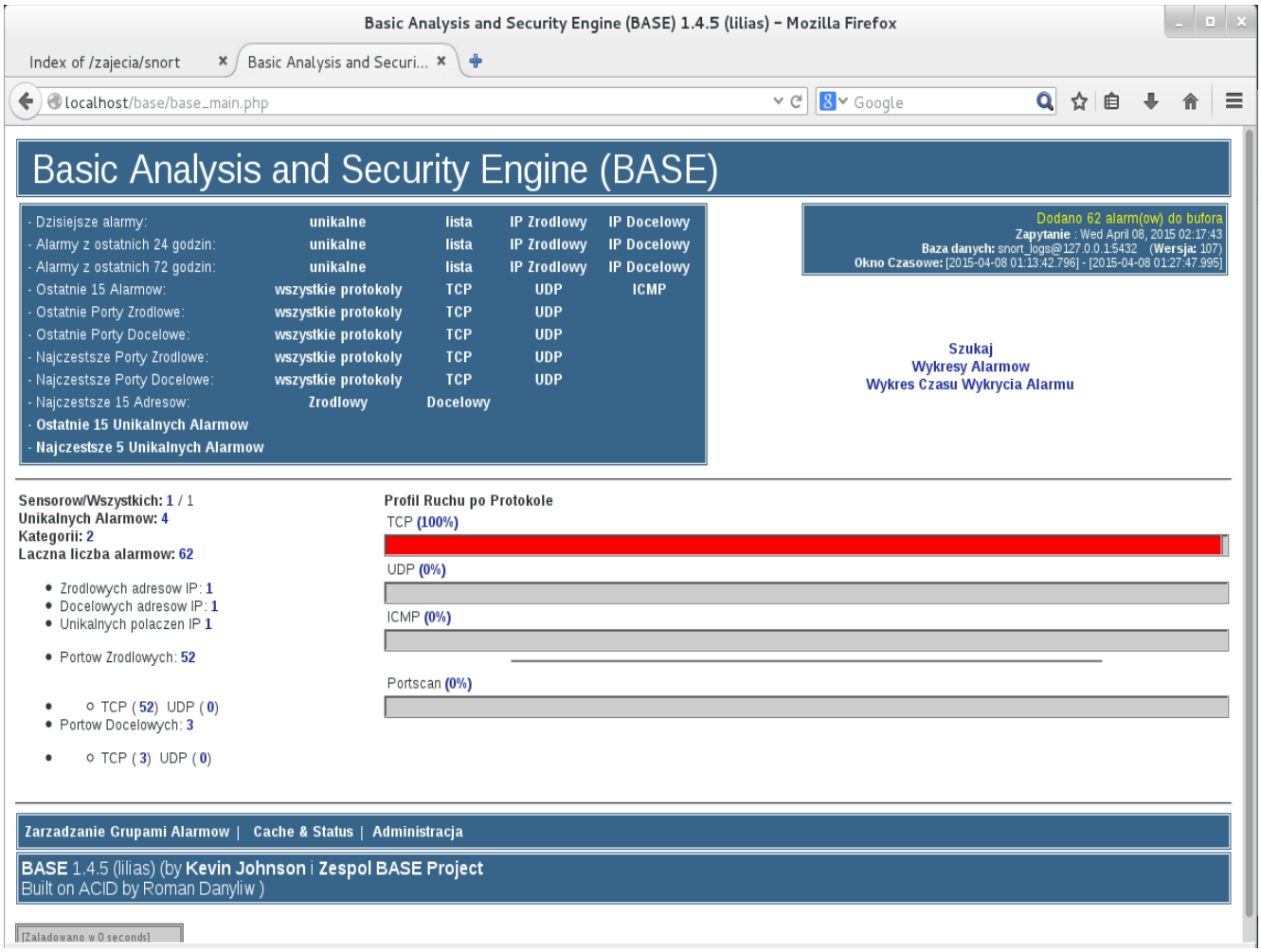
Ostatni krok konfiguracji to utworzenie rozszerzonego schematu bazy z dodatkowymi danymi używanymi przez aplikację *BASE*. W kroku tym nie podejmuje się żadnej decyzji.

? *Konfiguracja BASE zapisywana jest w katalogu aplikacji, w pliku base_conf.php. Usunięcie tego pliku spowoduje ponowne uruchomienie procesu konfiguracji.*

Uruchamianie BASE

Skonfigurowaną już aplikację *BASE* uruchamia się poprzez przeglądarkę WWW z adresem: *http://localhost/base*. Jeżeli skonfigurowana została kontrola dostępu do aplikacji, należy uwierzytelnić się podając ustaloną nazwę użytkownika i hasło.

Po prawidłowym uwierzytelnieniu otwierana jest strona główna aplikacji:



Konto utworzone w procesie konfiguracji jest kontem administratora; użytkownik ten może korzystać z pełnej funkcjonalności *BASE,* w tym - tworzyć inne konta użytkowników, którzy mogą tylko dokonywać analizy danych.

Funkcjonalność aplikacji *BASE* jest bardzo szeroka i jej opisywanie wykracza poza ramy tego opracowania, zresztą korzystanie z większości funkcji jest intuicyjne. Zainteresowanych szczegółami odsyłamy do dokumentacji znajdującej się na witrynie projektu.

Notatki

7 Pozostałe informacje

Filtrowanie ruchu przez Snort

Oprócz działania pasywnego, jakim jest generowanie ostrzeżeń o potencjalnych atakach, *Snort* może też działać aktywnie, tzn. próbować nie dopuścić do zaistnienia zdarzenia.

Jedną z takich możliwości jest praca Snort w tzw. trybie *inline*. W trybie tym Snort nie otrzymuje pakietów z interfejsu sieciowego, tylko od filtra pakietów *netfilter*. Wykorzystuje się do tego specjalny cel polecenia `iptables -` QUEUE. Snort działający w trybie *inline* obsługuje trzy dodatkowe, standardowe typy reguł: `drop`, `sdrop` i `reject`. Wykonanie reguły typu `drop` powoduje, że *netfilter* porzuci pakiet (odpowiednik celu DROP polecenia `iptables`), zaś *Snort* wygeneruje ostrzeżenie. Wykonanie reguły typu `sdrop` również powoduje porzucenie pakietu, ale bez wygenerowania ostrzeżenia. Wykonanie reguły typu `reject` powoduje, oprócz porzucenia pakietu i wygenerowania ostrzeżenia, także próbę uniemożliwienia atakującemu dalszych czynności: w przypadku protokołu TCP - poprzez wysłanie mu pakietu TCP z ustawioną flagą RST, w przypadku protokołu UDP - poprzez wysłanie mu pakietu ICMP z komunikatem *Destination Unreachable*.

Praca w trybie *inline* wymaga odpowiednio skompilowanego zarówno polecenia `snort`, jak i modułu *netfilter* i polecenia `iptables`, czego nie zapewniają standardowe pakiety w dystrybucji CentOS. Zainteresowanym proponujemy użycie specjalizowanej minidistrybucji Honeywall, dostępnej na witrynie http://honeynet.org.

Kolejną możliwością jest użycie systemu odpowiedzi (tzw. *flexible response*). Korzystając z parametru reguły `resp` można określić, że *Snort* zareaguje na wykonanie reguły wysłanie pakietu TCP/RST lub ICMP na adres źródłowy i/lub docelowy pakietu, który spowodował wykonanie reguły. Z kolei parametr `react` odnosi się do ruchu protokołu HTTP i umożliwia wysłanie do przeglądarki strony z komunikatem wygenerowanym przez *Snort* zamiast zawartości oczekiwanej przez przeglądarkę.

Należy pamiętać, że wykorzystanie w regule parametrów systemu odpowiedzi NIE powoduje porzucenia pakietu, który spowodował wykonanie reguły.

Podobnie jak w przypadku trybu *inline*, korzystanie z systemu odpowiedzi wymaga odpowiedniego skompilowania polecenia `snort`.

Mierzenie i poprawa wydajności

W przypadku analizy ruchu pochodzącego z szybkiego i intensywnie wykorzystywanego łącza dużego znaczenia nabiera zagadnienie obciążenia systemu przez działanie *Snort*. W szczególności, w przypadku wyczerpania zasobów systemowych, może dojść do wysoce niepożądanej sytuacji, kiedy nie wszystkie pakiety zostaną poddane analizie. Aby umożliwić administratorowi zapewnienie właściwych warunków pracy *Snort*, dostępne są narzędzia monitorujące wydajność preprocesorów (dyrektywa `config profile_preprocs`) oraz reguł (dyrektywa `config profile_rules`). Ponadto można ograniczyć ilość generowanych ostrzeżeń poprzez nałożenie limitów (ogólna dyrektywa `treshold` oraz parametr `treshold` dla konkretnej reguły) lub tymczasowe wyłączenie generowania ostrzeżeń o zadanych identyfikatorach GID i SID (dyrektywa konfiguracyjna `suppress`).

Odtwarzanie historii ruchu sieciowego

Podczas analizy wykrytych zdarzeń, jak również podczas przeprowadzania testów zabezpieczeń, z reguły przydatna jest możliwość odtworzenia historii ruchu. Polecenie `tcpreplay` (pakiet *tcpreplay* dostępny w repozytorium RPMForge) umożliwia ponowne wysłanie pakietów zapisanych w pliku w formacie *pcap*; plik taki może być wygenerowany za pomocą sniffera sieciowego (np. `tcpdump, wireshark`), może to być także plik logu Snort z zapisanymi pakietami, których wystąpienie spowodowało wygenerowanie ostrzeżenia.

! Uruchomienie polecenia tcpreplay na aktualnie zapisywanym przez Snort pliku logu doprowadzi do zapętlenia. Należy albo skopiować bieżący plik logu, albo zrestartować Snort tak aby został założony kolejny plik logu.

Uruchomienie polecenia `tcpreplay` przebiega następująco:

```
# tcpreplay -i interfejs_wyjściowy [opcje] plik_pcap
```

Domyślnie polecenie `tcpreplay` wysyła pakiety zapisane w pliku *pcap* zachowując wszystkie ich parametry oraz odstępy czasu pomiędzy nimi, czyli dokładnie odtwarza zapisany przebieg transmisji. Działanie to może być modyfikowane za pomocą opcji, z których najczęściej używane prezentuje poniższa tabela.

Tabela 7: Najczęściej używane opcje polecenia tcpreplay

Opcja	*Znaczenie*
`-K`	Powoduje, że plik *pcap* jest wczytywany w całości do pamięci przed rozpoczęciem odtwarzania. Dzięki temu ewentualne opóźnienia w odczycie pliku nie wpływają na odstępy między pomiędzy wysyłanymi pakietami.

Opcja	*Znaczenie*
`-l ilość`	Wskazuje, ile razy ma nastąpić odtworzenie historii ruchu (domyślnie: 1).
`-L ilość`	Wskazuje, ile pakietów zapisanych w pliku ma zostać wysłane; domyślnie wysyłane są wszystkie pakiety.
`-x mnożnik`	Ustala przyspieszenie lub spowolnienie odtworzenia w stosunku do pierwotnego przebiegu transmisji o podany mnożnik
`-p prędkość` `-M prędkość`	Ustala stałą prędkość, z jaką odtwarzana jest historia ruchu, odpowiednio w pakietach/s (`-p`) lub Mbit/s (`-M`)
`-t`	Powoduje odtworzenie historii z maksymalną możliwą prędkością
`-o`	Powoduje odtworzenie historii w trybie interaktywnym, w którym wysłanie każdego pakietu lub grupy pakietów jest potwierdzane

Przykład:

```
# tcpreplay -i eth0 /var/log/snort/attack.log
sending out eth0
processing file: /var/log/snort/attack.log
Actual: 6 packets (348 bytes) sent in 15.47 seconds
Rated: 22.5 bps, 0.00 Mbps, 0.39 pps
Statistics for network device: eth0
        Attempted packets:         6
        Successful packets:        6
        Failed packets:            0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
# tcpreplay -i eth0 -t /var/log/snort/attack.log
sending out eth0
processing file: /var/log/snort/attack.log
Actual: 6 packets (348 bytes) sent in 0.00 seconds
Rated: inf bps, inf Mbps, inf pps
Statistics for network device: eth0
        Attempted packets:         6
        Successful packets:        6
        Failed packets:            0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
# tcpreplay -i eth0 -o /var/log/snort/attack.log
sending out eth0
processing file: /var/log/snort/attack.log
**** Next packet #1 out eth0.  How many packets do you wish to send?
Sending packet 1 out: eth0
**** Next packet #2 out eth0.  How many packets do you wish to send?
Sending packet 2 out: eth0
**** Next packet #3 out eth0.  How many packets do you wish to send?
Sending packet 3 out: eth0
**** Next packet #4 out eth0.  How many packets do you wish to send? 3
Sending packet 4 out: eth0
Actual: 6 packets (348 bytes) sent in 9.10 seconds
Rated: 38.2 bps, 0.00 Mbps, 0.66 pps
Statistics for network device: eth0
        Attempted packets:         6
        Successful packets:        6
        Failed packets:            0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
```

Często spotyka się również potrzebę zmiany adresów w zapisanej historii ruchu. Jest to stosowane na przykład w przypadku, gdy chcemy przeprowadzić symulację wykrytego ataku na systemie testowym, lub też udostępnić historię

Notatki

ruchu osobie trzeciej do przeanalizowania, a zależy nam na nieujawnianiu adresu zaatakowanego systemu. W takiej sytuacji można posłużyć się poleceniem `tcprewrite`, należącym również do pakietu *tcpreplay*.

Uruchomienie polecenia `tcprewrite` przebiega następująco:

```
# tcpreplay –i plik_wejściowy –o plik_wyjściowy opcje
```

Polecenie `tcprewrite` kopiuje historię ruchu z pliku wejściowego do pliku wyjściowego dokonując zmian wyspecyfikowanych za pomocą opcji.

Tabela 8: Najczęściej używane opcje polecenia tcprewrite

Opcja	***Znaczenie***

`–N wzorzec`
`–D wzorzec`
`–S wzorzec`
	Powoduje zamianę adresów pakietu zgodnie z podanym wzorcem. Zamiana dotyczy dowolnego adresu (`–N`), tylko adresu docelowego (`–D`) lub tylko adresu źródłowego (`–S`).
	Wzorzec jest określony jako oddzielana przecinkami sekwencja par; każda para składa się z dwóch przedzielonych dwukropkiem adresów podsieci w notacji CIDR lub adresów hostów. Pierwszy adres w parze jest adresem pierwotnym, zaś drugi – adresem, który ma zostać użyty w pliku wynikowym.
	Przykład: `–N 234.56.78.90:127.0.0.1,212.51.220.0/128:172.16.0.0/128`

`–p wzorzec`	Powoduje zmianę portu TCP lub UDP zgodnie z podanym wzorcem. Wzorzec jest określony jako oddzielana przecinkami sekwencja par; każda para składa się z dwóch przedzielonych dwukropkiem numerów portów. Pierwszy numer w parze jest numerem pierwotnym, zaś drugi – numerem, który ma zostać użyty w pliku wynikowym. Przykład: `–p 8080:80,8443:443`
`–e adres1:adres2`	Zamienia adresy pakietów tak, aby historia wynikowa odpowiadała konwersacjom między hostami o adresach `adres1` i `adres2`.

Przykład:

```
# tcprewrite –i attack.log –o attack-local.log –D 217.76.116.201:127.0.0.1
```

Polecenie tcprewrite posiada rozległe możliwości modyfikacji parametrów kopiowanych pakietów.
Zobacz też: # man tcprewrite