Projekt HoneySpy – sieć serwerów Honeypot

Bezpieczeństwo Systemów Operacyjnych

Robert Nowotniak

Michał Wysokiński

Pomysł projektu

Celem projektu jest stworzenie oprogramowania do budowy sieci serwerów typu Honeypot. Narzędzia tego typu są wykorzystywane do wczesnego wykrywania incydentów bezpieczeństwa w sieciach komputerowych, prób uzyskania nieuprawnionego dostępu do zasobów lub usług oraz analizy nowych, powszechnie niejawnych zagrożeń. Działają więc na zasadzie ,,pułapki'', imitując prawdziwe usługi w sieci.

Serwer, który należy napisać, powinien działać w trybie demona w jednym z kilku trybów. W przypadku działania z uprawnieniami zwykłego użytkownika proces nasłuchuje na wysokich portach TCP i UDP w celu przekazania połączeń do różnych modułów odpowiedzialnych za symulowanie fałszywych usług. W przypadku posiadania odpowiednich uprawnień przez proces serwera (cap_net_bind_service, cap_net_raw, cap_net_admin) lub działania z prawami superużytkownika, serwer powinien móc przypisywać dodatkowe aliasy IP do interfejsów sieciowych, stwarzając złudzenie dostępności dodatkowych hostów w sieci. Pożądane jest, aby nie było możliwości wykrycia działania honeypota poprzez porównywanie adresów warstwy liniowej z innego punktu w danym segmencie sieci. W tym celu serwer powinien fałszować adresy MAC dla każdego z wykorzystywanych adresów IP. Można to zrealizować np. poprzez symulowanie bridge'a i dodawanie odpowiednich wpisów do tabeli przekształceń arptables i ebtables, których regułami powinien zarządzać honeypot.

Serwer (działający w trybie rozszerzonym) powinien mieć także możliwość otwarcia surowego gniazda sieciowego (raw socket) w celu wyczekiwania na pakiety zgodne ze wzorcami określonymi przez osobę zarządzającą honeypotem. Opis pakietów, które muszą być wykrywane, powinien być określany w składni interfejsu PCAP.

Demon powinien wykorzystywać narzędzia do pasywnej analizy odcisku stosu TCP/IP w celu wykrywania systemu operacyjnego na hostach łączących się z honeypotem. Opcje takiej analizy (brane pod uwagę pakiety, dopasowywanie przybliżone) powinno być można konfigurować. Z drugiej jednak strony, serwer powinien fałszować charakterystykę swojego stosu TCP/IP dla każdego z aliasów IP w ten sposób, aby samemu oszukiwać tego typu narzędzia (nmap, xprobe, p0f, amap). W tym celu należy wykorzystać odpowiednie łaty na jądro (dostępne dla niektórych systemów operacyjnych).

Każdy z serwerów powinien móc logować zdarzenia do serwera centralnego, lub móc pobierać od niego opcje konfiguracyjne. Komunikacja powinna być szyfrowana za pomocą SSL z wykorzystaniem certyfikatów.

Przykładowe imitowane usługi, które mógłby realizować serwer:

- Fałszywy serwer open-relay
- Serwer ssh z imitacją dostępu do prostej powłoki
- Imitacja open proxy (tzw. sugarcage)
- Nieaktualna wersja serwera www

Głównym zadaniem honeypota jest logowanie prób dostępu do symulowanych usług, wysyłanych do niego zapytań, a w szczególności prób wykorzystania błędów przepełnienia bufora, łańcuchów formatujących, zapytań zawierających shellcode'y itp.

Siecią serwerów powinno dać się zarządzać z poziomu aplikacji webowej pracującej w modelu AJAX, co pozwoli uniknąć ciągłego przeładowywania strony (np. przy podglądzie nadchodzących pakietów).

Serwer powinien być napisany z myślą o bezpieczeństwie, więc powinien wykorzystywać mechanizmy typu chroot(2), tryb skażenia Perla, pozbywanie się uprzywilejowanych identyfikatorów, gdy tylko jest to możliwe, a docelowo powinien być chroniony przez model obowiązkowej kontroli dostępu MAC, np. mechanizm RBAC (należy przygotować odpowiedni zestew reguł).

Projekt HoneySpy jest realizowany w większości w języku Perl.

Podsystemy Honeypota

HoneySpy jest narzędziem wykorzystującym różne gotowe rozwiązania (ippersonality, p0f, ebtables) i narzędzia sieciowe, pozwalającym na łatwe zarządzanie siecią wielu takich serwerów działających w różnych punktach sieci.

Komunikacja pomiędzy serwerami Honeypot – podsystem RPC

Komunikacja pomiędzy serwerami podrzędnymi (sensorami), a serwerem nadzorującym służy dwóm celom:

- Przesyłanie logów zdarzeń na serwer centralny
- Wysyłanie rozkazów z serwera do sensorów

Komunikacja musi się odbywać poprzez szyfrowane połączenie SSL, więc nie ma możliwości wykorzystania gotowych rozwiązań typu XMLRPC. Każde wywołanie zdalnej funkcji wiązałoby się z negocjacją szyfrowanego połączenia, co byłoby bardzo obciążające.

Zamiast tego serwery utrzymują ciągłe połączenia między sobą (podobnie jak serwery IRC). Serwer, pracujący w trybie sensora, po wystartowaniu łączy się z serwerem centralnym. Serwer centralny może wydawać rozkazy serwerowi podrzędnemu poprzez wysyłanie mu nazwy funkcji, którą ma wykonać, kontekstu oraz zserializowanej listy argumentów (przypomina to funkcje w języku Lisp).

Jednoczesna obsługa wielu połączeń jest realizowana za pomocą modułu obiektowego interfejsu dla wywołania systemowego select(2) – IO::Select. Każdy węzeł posiada listę uchwytów (r_set, w_set, e_set), które są obserwowane na okoliczność wystąpienia na nich możliwości nieblokującego odczytu, zapisu lub zainstnienia wyjątku.

Każdy węzeł posiada także hash z referencjami handlerów, które należy wywołać w przypadku wystąpienia zdarzenia na którymś z obserwowanych uchwytów. Handlery te są dodawane w odpowiednich funkcjach na zasadzie domknięć (ang. *closure*).

Metoda odpowiedzialnymi za odebranie i wykonanie funkcji do wykonania są Node::process_command oraz Node::_call_function. Metody te są wywoływane wtedy, gdy jest pewność, że dostępne są dane do odczytu z gniazda i odczyt nie spowoduje zablokowania.

```
sub process_command {
   ...
      my $buf;
      sysread($sock, $buf, 4);
```

```
        my $len = unpack('N', $buf);
        sysread($sock, $buf, $len);
        my ($function, $arrayctx, @args);
        eval {
           ($function, $arrayctx, @args) = @{thaw($buf)};
        };
        for ($@) {
           if (/Magic number checking on storable string failed/) {
              $logger->error("Wrong data received from client.");
              return;
           }
        }

        local $" = ',';
        $logger->debug("Running $function(@args) in "
           . ($arrayctx?'list':'scalar') . ' context');

        my @result = $self->_callFunction($function, $arrayctx, @args);
        $logger->debug("Function result: @result");

        $self->addfh($sock, 'w');
        $self->{'w_handlers'}{$sock} = sub {
           sendToPeer($sock, @result);
           $self->removefh($sock, 'w');
        };

     }
  }
```

Rozserializowanie przysłanych danych

Handler odsyłający wartość zwracaną

Metoda wywołująca funkcję wraz z argumentami przysłanymi z rozkazem:

```
  sub _callFunction {
     my ($self, $function, $arrayctx, @args) = @_;
     my @result;

     eval {
        no strict 'refs';
        unshift(@args, $self);
        if ($arrayctx) {
           @result = @{[&{*{$function}}(@args)]};
        }
        else {
           @result = (scalar &{*{$function}}(@args));
        }
     };
     for ($@) {
        last unless ($@);

        if (/Undefined subroutine/) {
           $result[0] = "No such function ($function) on remote side";
           last;
        }
        else {
           $result[0] = "Error $_ during excecution of remote called function";
        }

        $logger->error($result[0]);
     }
     return @result;
  }
```

Wywołanie w odpowiednim konteście

Zarządzanie siecią serwerów

Konsola tekstowa

Do testowania sieci została wstępnia przygotowana konsola tekstowa (korzystająca z biblioteki ReadLine), za pomocą której można wywoływać funkcje na węzłach sieci.

```
$ ./console.pl 127.0.0.1 9000
Enter PEM pass phrase:
************************************************************
***            HoneySpy experimental console             ***
************************************************************

Connection established
2005/11/07 22:33:35 konstruktor
> getName
2005/11/07 22:33:38 Odpowiedz sensora:
2005/11/07 22:33:38 main
> getSensors
2005/11/07 22:33:40 Odpowiedz sensora:
2005/11/07 22:33:40 main
->sensor1
> runOnNode sensor1 kill
2005/11/07 22:34:00 Odpowiedz sensora:
2005/11/07 22:34:00 0
> getSensors
2005/11/07 22:34:04 Odpowiedz sensora:
2005/11/07 22:34:04 main
> _
```

Podanie frazy kudującej dla klucza prywatnego administratora

Zaprojektowany model klas oraz system zdalnego wywoływania procedur pozwala zrealizować taką konsolę administracyjną w następujący, bardzo prosty sposób:

```
my $master = IO::Socket::SSL->new( PeerAddr => $ARGV[0],
   PeerPort => $ARGV[1],
   Proto    => 'tcp',
   SSL_use_cert => 1,

   SSL_key_file => '../certs/admin-key.enc',
   SSL_cert_file => '../certs/admin-cert.pem',
   SSL_ca_file => '../certs/master-cert.pem',
   SSL_verify_mode => 0x01);

my $s = new Sensor({
   name => 'main',
   socket => $master,
});

...
print $prompt;
while (defined($_ = $term->readline($prompt))) {
   $term->addhistory($_) if /\S/;
   my ($cmd, @args) = ($_);
   if (/^(.*?)\s(.*)/) {
      ($cmd, @args) = ($1, split(/\s/, $2));
   }

   $s->sendToPeer($cmd, 1, @args);
   my @res = $s->recvFromPeer();

   print $prompt;
```

Druga strona musi miec certyfikat podpisany przez mastera

```
}
```

Interfejs www

Interfejs www do zarządzania siecią serwerów powinien umożliwiać obejrzenie stanu serwerów, ich aktualnej listy, działających na nich usług oraz trybów, w których pracują serwery (czy jest aktywne fałszowanie stosu TCP/IP, zmiana adresów MAC, ...). Interfejs powinien także umożliwiać przeglądanie dziennika zdarzeń z serwerów, zmiany ich konfiguracji oraz wyłączenie serwera podrzędnego. Do zrealizowania interfejsu www może posłużyć np. system szablonów Perla Template::Toolkit.

Kod realizujący interfejs www będzie równie prosty jak kod konsoli tekstowej – dzięki klasie Sensor.

Dzięki korzystaniu z metody AUTOLOAD w klasie Sensor każde wywołanie metody, która nie została w niej zdefiniowana będzie obsłużone jako rozkaz wywołania tej metody zdalnie.

Przykładowo więc wywołanie postaci:

```
my $result = $sensors{'sensor1'}->setFingerprint('213.76.144.125', 'Tru64');
```

spowoduje wysłanie do sensora1 rozkazu wywołania na nim funkcji setFingerprint (bo w klasie Sensor nie jest ona zdefiniowana) w kontekście skalarnym.

Diagram wdrożenia

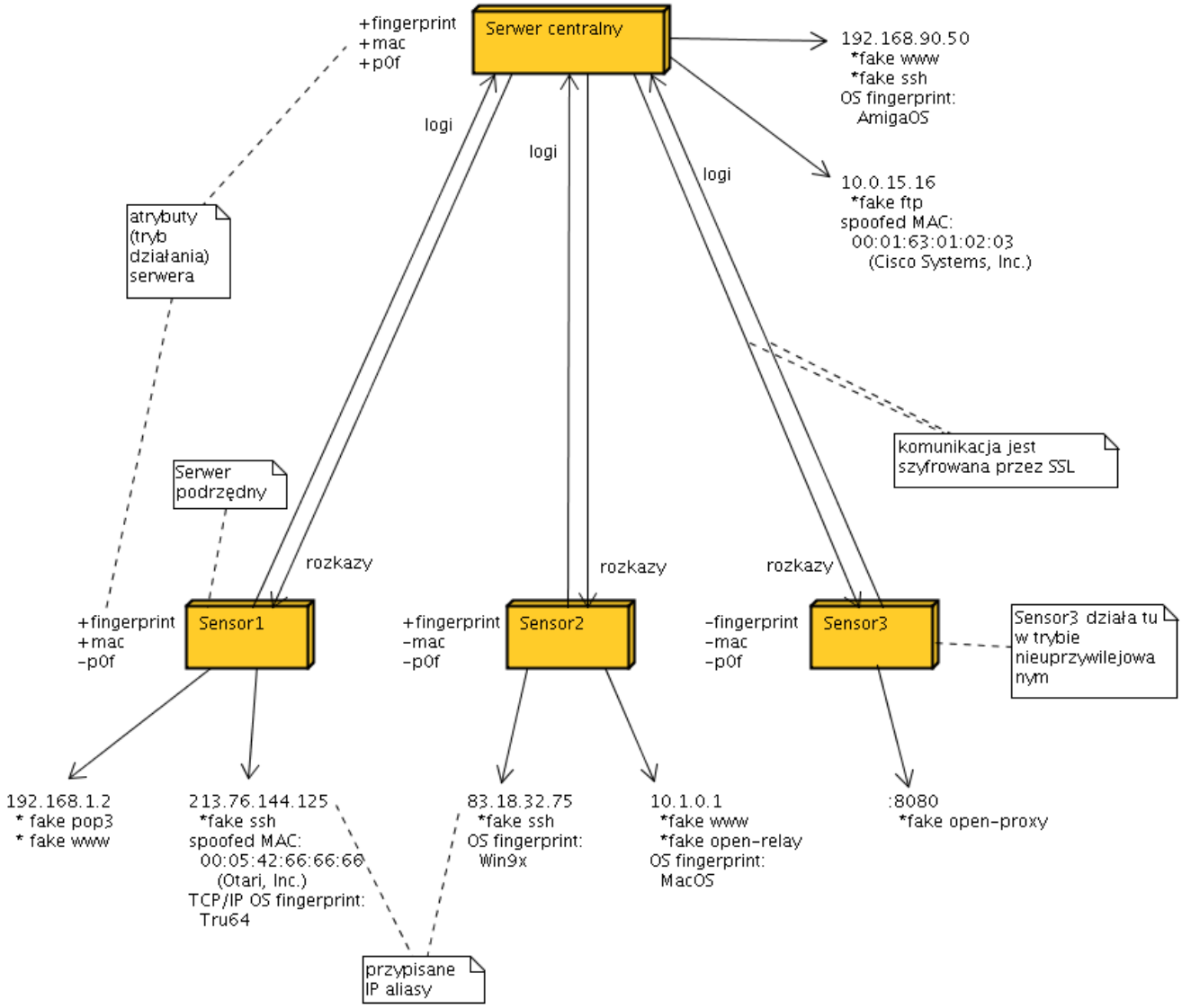
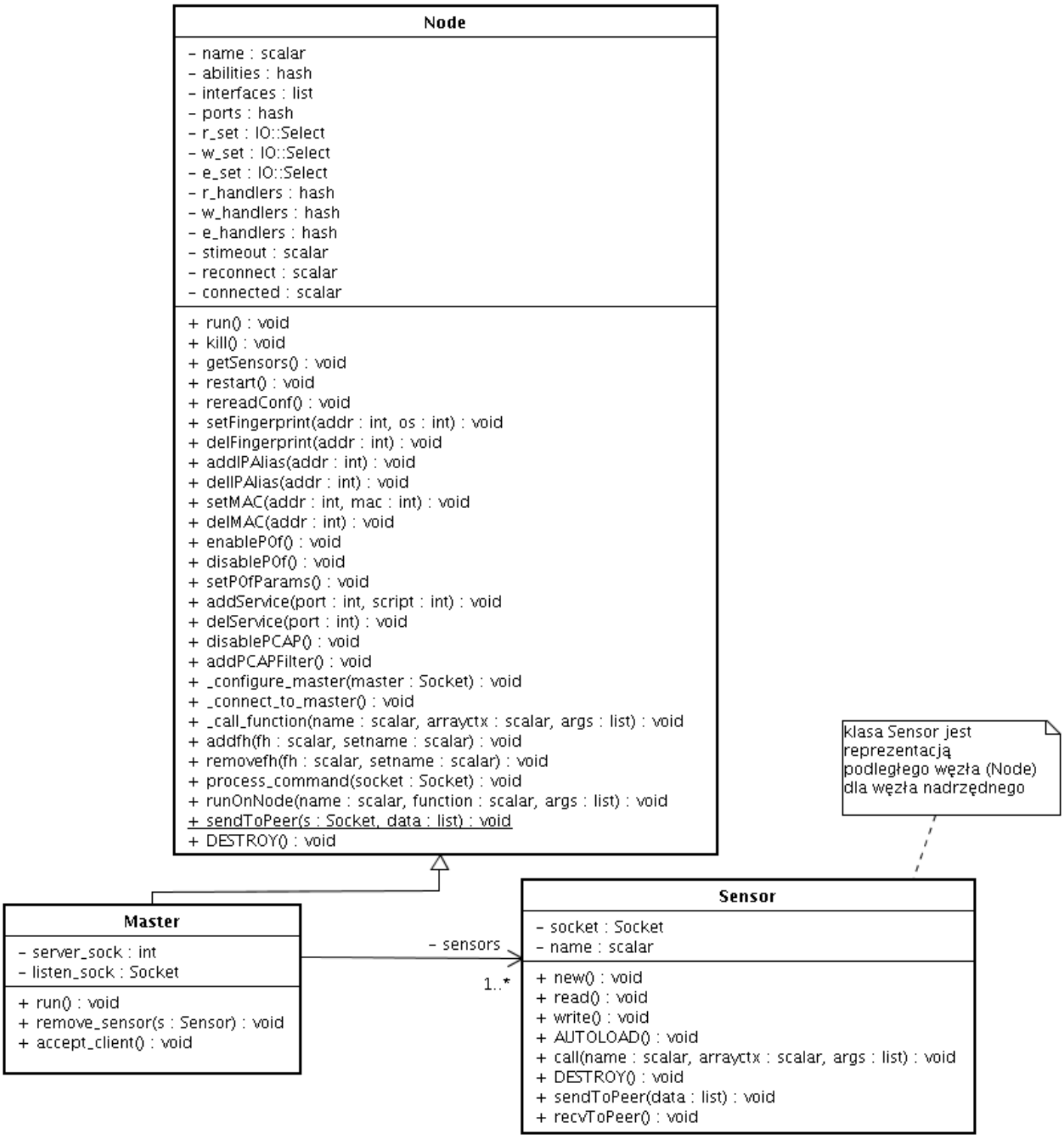


Diagram klas



Uwierzytelnianie za pomocą certyfikatów

Każdy węzeł w sieci dysponuje własną parą kluczy, własnym certyfikatem oraz zna certyfikat serwera centralnego. Połączenia mogą być ustanawiane tylko wówczas, gdy druga strona przedstawia certyfikat podpisany certyfikatem serwera centralnego.

Dla każdego nowego węzła sieci musi zostać wygenerowana para kluczy oraz certyfikat podpisany certyfikatem serwera głównego.

Do zarządzania certyfikatami służy narzędzie certs/generate.sh

```
$ ./generate.sh sensor
Tworzenie kluczy i certyfikatu klienta (sensor) sieci

Nazwa klienta:
sensor2
Generating RSA private key, 1024 bit long modulus
..++++++
.............................................++++++
e is 65537 (0x10001)
Enter pass phrase for sensor2-key.enc:
Verifying - Enter pass phrase for sensor2-key.enc:
Enter pass phrase for sensor2-key.enc:
writing RSA key
Using configuration from /etc/ssl/openssl.cnf
DEBUG[load_index]: unique_subject = "yes"
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 3 (0x3)
        Validity
            Not Before: Nov  7 21:20:11 2005 GMT
            Not After : Nov  7 21:20:11 2006 GMT
        Subject:
            countryName               = PL
            organizationName          = HoneySpy network
            organizationalUnitName    = sensor
            commonName                = sensor2
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                40:21:38:EE:BE:3A:95:45:14:FD:C8:7F:48:9D:DA:7F:85:8D:20:47
            X509v3 Authority Key Identifier:
                keyid:5E:E6:37:0F:7D:95:95:3A:15:02:9D:E5:06:9C:F4:59:11:35:13:A2
                DirName:/C=PL/O=HoneySpy network/OU=Master Server/CN=Master
                serial:90:D0:FC:BB:23:2B:7C:8F

Certificate is to be certified until Nov  7 21:20:11 2006 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

Pokazać certyfikat? [t/n] n
```

Specjalnym rodzajem klienta sieci jest administrator, dla którego także musi zostać wygenerowana para kluczy (./generate.sh admin) oraz certyfikat podpisany certyfikatem serwera centralnego. Ponadto klucz prywatny administratora jest zaszyfrowany frazą kodującą.

Połączenie administratora z siecią serwerów jest więc możliwe tylko w przypadku posiadania frazy kodujące klucza prywatnego.

Nasłuchiwanie ruchu sieciowego (PCAP)

Serwer Honeypot może otworzyć surowe gniazdo sieciowe (korzystanie z Net::PCAP) w celu nasłuchiwania na ruch sieciowy o określonych wzorcach. Składnia służąca do określania pasujących pakietów powinna być zgodna z interfejsem PCAP – wykorzystywanym w narzędziach typu tcpdump, snort, ethereal.

Każdy Honeypot posiada listę reguł, których suma logiczna jest kompilowana do postaci reprezentacji wewnętrznej w chwili włączenia nasłuchiwania przez interfejs PCAP.

Do zarządzania interfejsem PCAP służą metody klasy Node: addFilter, replaceFilters, delFilter, getFilters, getFilter, enablePcap. Metody te mogą być wywoływane zdalnie, za pośrednictwem opracowanego systemu RPC.

Załóżmy, że z testowej konsoli administracyjnej wydane zostały następujące rozkazy:

```
> addFilter tcp[tcpflags] & tcp-syn != 0
...
> addFilter icmp
...
> getFilters
2005/11/09 20:53:02 Odpowiedz sensora:
2005/11/09 20:53:02 tcp[tcpflags] & tcp-syn != 0
->icmp
> enablePcap
```

Powyższe reguły powodują wychwytywanie pakietów TCP z ustawioną flagą SYN oraz wszystkich pakietów ICMP. enablePcap aktywuje nasłuchiwanie przez PCAP – deskryptor interfejsu jest po prostu dodawany do listy uchwytów r_set obserwowanych przez IO::Select na okoliczność wystąpienia możliwości nieblokującego odczytania danych (czyli nadejście pasującego pakietu).

```
sub _compileFilter {
   my ($self) = @_;
   my $compiled;

   return unless @{$self->{'pcap_filters'}};

   # sprawdzic kazda regule po kolei
   foreach (@{$self->{'pcap_filters'}}) {
      my $err = Net::Pcap::compile($self->{'pcap'}, \$compiled, $_, 1, 0);
      if ($err) {
         $err = "Error in rule: $_";
         $logger->error($err);
         return $err;
      }
   }

   if (@{$self->{'pcap_filters'}} > 1) {
      # zrobic alternatywe logiczna wszystkich regul
      my @filters = @{$self->{'pcap_filters'}};
      my $sum = $filters[0];
      $sum = "($sum) or ($_)" foreach @filters[1..$#filters];
      my $err = '';
      $err = Net::Pcap::compile($self->{'pcap'}, \$compiled, $sum, 1, 0);
```

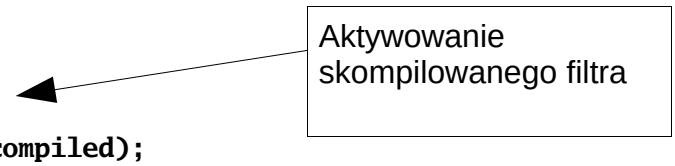
Sprawdzenie poprawnosci reguł po kolei

Kompilacja sumy logicznej wszystkich reguł

```
      if ($err) {
         $err = "Error in rules sum: $sum. " . Net::Pcap::geterr($self->{'pcap'});
         $logger->error($err);
         return $err;
      }
   }

   Net::Pcap::setfilter($self->{'pcap'}, $compiled);

   return 0;
}
```




```
Handler obslugujący możliwość odczytu pakietu z interfejsu PCAP wywołuje funkcję loop
na tym interfejsie, a w wywołaniu tej funkcji podawana jest referencja na funkcję
_pcapPacket, która obsługuje wychwycony pakiet, czyli loguje informację o nim.

sub _pcapPacket {
   my ($user_data, $hdr, $pkt) = @_;

   my $eth_obj = NetPacket::Ethernet->decode($pkt);
   my $msg = "Packet matched PCAP rule.";
   $msg .= " src mac: " . $eth_obj->{'src_mac'};
   $msg .= " dst mac: " . $eth_obj->{'dst_mac'}
      if defined($eth_obj->{'dst_mac'});

   #my $ip_obj = NetPacket::IP->decode(eth_strip($pkt));
   my $ip_obj = NetPacket::IP->decode(substr($eth_obj->{'data'}, 2));
   $msg .= " | ";
   $msg .= 'src:' . $ip_obj->{'src_ip'};
   $msg .= ' dst:' . $ip_obj->{'dest_ip'};
   $msg .= ' ipver:' . $ip_obj->{'ver'};
   $msg .= ' tos:' . $ip_obj->{'tos'};
   $msg .= ' len:' . $ip_obj->{'len'};
   $msg .= ' id:' . $ip_obj->{'id'};
   $msg .= ' proto:' . $ip_obj->{'proto'};
   $msg .= ' flags:' . $ip_obj->{'flags'};
   $logger->info($msg);
}
```

W przypadku reguł filtra wprowadzonych powyżej otrzymamy przykładowo następujące wpisy w logach Honeypota:

- **zalogowany pakiet icmp**

```
2005/11/09 21:11:47 Packet matched PCAP rule. src mac: 000000806028 | src:192.168.66.6
dst:212.77.100.101 ipver:4 tos:0 len:84 id:0 proto:1 flags:2
2005/11/09 21:11:47 Packet matched PCAP rule. src mac: 00024450bc99 |
src:212.77.100.101 dst:192.168.66.6 ipver:4 tos:0 len:84 id:23594 proto:1 flags:0
```

- **zalogowany pakiet TCP z flagą SYN**

```
2005/11/09 21:13:26 Packet matched PCAP rule. src mac: 000000806028 | src:192.168.66.6
dst:213.180.130.200 ipver:4 tos:0 len:60 id:59803 proto:6 flags:2
2005/11/09 21:13:28 Packet matched PCAP rule. src mac: 00024450bc99 |
src:213.180.130.200 dst:192.168.66.6 ipver:4 tos:0 len:60 id:0 proto:6 flags:2
```

Rozpoznawanie zdalnego systemu operacyjnego

HoneySpy wykorzystuje narzędzie p0f w celu rozpoznawania zdalnych systemów operacyjnych na łączących się hostach. Za pomoca p0f system może także zbierać informacje o czasie uptime na zdalnych hostach lub o znajdujących się za nimi sieciami NAT. Opcje konfiguracyjne dla p0f powinno być można konfigurować z poziomu interfejsu www.

Fałszowanie charakterystyki stosu TCP/IP

HoneySpy jest pisany z myślą o systemach GNU/Linux oraz FreeBSD. Dla obydwu tych systemów są dostępne łaty na jądro pozwalające na fałszowanie charakterystyki stosu TCP/IP:

- IP Personality (dla GNU/Linux)
- FingerPrintFucker (dla FreeBSD)

Przykładowo do zmiany charakterystyki stosu na parametry wczytane z /etc/pers/amigaos.conf (udawanie AmagaOS) przy uzycie IP Personality mogą być użyte następujące reguly iptables:

```
echo 20480 > /proc/sys/net/ipv4/ip_conntrack_max
iptables -t mangle -I PREROUTING 1 -d 192.168.66.2 -j honeyspy
iptables -t mangle -A OUTPUT -s 192.168.66.2 -j honeyspy
iptables -t mangle -A honeyspy -d 192.168.66.2 \
   -j PERS --tweak dst --local --conf /etc/pers/amigaos.conf
iptables -t mangle -A honeyspy -s 192.168.66.2 \
   -j PERS --tweak src --local --conf /etc/pers/amigaos.conf
```

W celu wykorzystania łat IPPersonality (ten projekt został zarzucony), należy je dostosować do aktualnej wersji jądra (drobne zmiany).

Fałszowanie adresów warstwy liniowej

Jednym z zadań HoneySpy jest możliwość zmiany adresu MAC w zależności od adresów, które zostały przypisane do honeypota. Serwer realizuje to poprzez zarządzanie konfiguracją software'owego bridge'a oraz wprowadzanie wpisów do tabel ebtables i arptables.

W celu fałszowania adresów MAC przez HoneySpy musi zostać utworzony interfejs bridge'a (nazwijmy go honeyspy), do którego portów zostaje podłączony rzeczywisty interfejs sieciowy (na przykład eth0) oraz atrapy (ang. *dummy*) interfejsów, którym są przypisywane adresy IP symulowanych hostów.

```
arptables -t mangle -A OUTPUT  --h-length 6 -o honeyspy -s 192.168.66.6 \
        -j mangle --mangle-mac-s 9e:2d:3c:22:94:15
arptables -t mangle -A OUTPUT  --h-length 6 -o honeyspy -s <ip_honey_spy> \
        -j mangle --mangle-mac-s <spoof_mac>

ebtables -t nat -A PREROUTING -d 00:00:00:00:00:01  -j redirect
ebtables -t nat -A PREROUTING -d <spoof_mac>  -j redirect
```

Moduły imitacji usług

Moduły z fałszywymi usługami powinny być uruchamiane w sposób podobny do działania super-serwera (inetd, xinetd). Umożliwi to bardzo łatwe dodawanie nowych usług, przenoszenie modułów z innych narzędzi typu Honeypot (np. honeyd) lub wykorzystanie modułów w innych narzędziach. Demon Honeypot powinien nasłuchiwać na wybranych portach TCP i UDP, oraz utrzymywać w pamięci odwzorowanie postaci <nr_port>->skrypt. W momencie otrzymania połączenia demon tworzy nowy proces, w którym duplikuje standardowe deskryptory I/O za pomocą deskryptora gniazda sieciowego, a następnie uruchamia skrypt odpowiedzialny za uruchomienie usługi.

Moduł odpowiedzialny za nasłuchiwanie i wywoływanie skryptów z usługami, powinien także mieć możliwość działania z uwzględnieniem przypisanych ograniczeń, np. odmawiać przyjmowania więcej niż danej liczby połączeń do danego moduły w określonym czasie, aby zapogać zagrożeniom typu Denial-Of-Service.

Skrypty z usługami mogą więc być dowolnymi programami, które odczytują dane wejściowe i piszą na standardowe wyjście.

Uruchamianie nowej imitacji usługi na węźle sieci

Metoda addService służy do otwarcia nowego portu i oczekiwania na połączenia, których obsługa będzie przekazana do odpowiedniego skryptu imitacji usługi.

```
#
# Przypisanie usługi na danym porcie
#
sub addService {
   my ($self, $addr, $proto, $port, $script, @args) = @_;
   $logger->info("Adding service on $addr:$port ($proto)");

   my $socket = new IO::Socket::INET(
      LocalAddr => $addr,
      LocalPort => $port,
      Proto => $proto,
      Listen => 5,
      Reuse => 1
   );
   if (!$socket) {
      my $msg = "Couldn't open socket: $!";
      $logger->error($msg);
      return $msg;
   }

   $self->addfh($socket, 'r');
   $self->{'r_handlers'}{$socket} = sub {
      my $client = $socket->accept();
      if (! $client) {
         $logger->error("Couldn't accept connection ($!)");
         return 1;
      }
      $logger->debug("Connection to service $script from " . $client->peerhost);
      $SIG{'CHLD'} = 'IGNORE';
      my $pid = fork();
      if (! $pid) {
         setsid();
         open(STDIN, "<&=".fileno($client));
         open(STDOUT, ">&=".fileno($client));
         { exec($script, @args); }
         $logger->error("Couldn't run script ($!)");
         return 1;
      }
   };

   return 0;
}
```

Dodanie gniazda do zbioru obserwowanych uchwytów

Przypisanie handlera (na zasadzie domknięcia)

Utworzenie podprocesu ze skryptem imitacji usługi

Główna pętla obsługi zdarzeń

```
sub run {
   my $self = shift;
   $logger->info("Starting node " . $self->{'name'});

   if ($self->{'mode'} eq 'sensor') {
      while (!($self->_connect_to_master())) {
         my $delay = $self->{'reconnect'};
         $logger->info("Retrying in $delay seconds");
         select(undef, undef, undef, $delay);
      }
   }

   local $| = 1;

   $logger->debug("Entering main loop - node " . $self->{'name'});
   for (;;) {
      $logger->debug("Waiting on select(2) syscall...");

      $logger->debug("Write watched handles: " , $self->{'w_set'}->handles);
      my ($r_ready, $w_ready, $e_ready) =
         IO::Select->select(
            $self->{'r_set'}, $self->{'w_set'}, $self->{'e_set'},
            $self->{'stimeout'});

      if (!defined($r_ready)) {
         #
         # Na zadnym uchwycie nie bylo zdarzenia
         #
         $logger->debug("Timeout");
         if ($self->{'mode'} eq 'sensor') {
            if (! $self->{'connected'}) {
               $logger->debug("Trying to reconnect to my master");
               $self->_connect_to_master();
            }
         }
         next;
      }

      foreach my $fh (@$r_ready) {
         $logger->debug("Something ($fh) in read ready set");

         $self->{'r_handlers'}{$fh}()
            if exists($self->{'r_handlers'}{$fh});
      }
      foreach my $fh (@$w_ready) {
         $logger->debug("Something ($fh) in write ready set");

         $self->{'w_handlers'}{$fh}()
            if exists($self->{'w_handlers'}{$fh});
      }
   }
}
```

Próba połączenia się z masterem

Oczekiwanie na zdarzenie

Wywołanie odpowiedniego handlera dla każdego aktywnego uchwytu pliku

Przykładowa, bardzo prosta usługa

Imitacja serwera finger mogłaby więc wyglądać w następujący sposób:

```
#!/usr/bin/perl -w

use strict;

my $username = <STDIN>;
chomp $username;

print <<EOF;

Welcome to Linux version 2.6.12.5-mipscvs-20050711-ip30r10k+ at serwer.honeyspy !
 12:53:22 up  1:48, 10 users,  load average: 0.20, 0.10, 0.08

EOF

print <<EOF;
Login: $username                            Name: (null)
Directory: /home/$username                  Shell: /bin/bash
On since Sun Nov  6 11:05 (CET) on tty1    1 hour 47 minutes idle
     (messages off)
On since Sun Nov  6 12:30 (CET) on pts/0   4 minutes 50 seconds idle
No mail.
EOF
```

Działanie węzłów sieci

Serwer centralny

Zdarzenia w sieci są logowane za pomocą modułu Log::Log4per. Log z przykładową sesją działania wstępnej wersji serwera przedstawiony jest poniżej.

Uruchomienie serwera:

```
2005/11/07 22:43:36 konstruktor Mastera
2005/11/07 22:43:36 konstruktor Node
Master=HASH(0x84edda0)
2005/11/07 22:43:36 Starting master server main
2005/11/07 22:43:36 Starting node main
2005/11/07 22:43:36 Entering main loop - node main
2005/11/07 22:43:36 Waiting on select(2) syscall...
```

Nadejście połączenia i włączenie węzła do sieci:

```
2005/11/07 22:43:48 Something (IO::Socket::SSL=GLOB(0x84eddd0)) in read ready set
2005/11/07 22:43:48 Client connected from 127.0.0.1
2005/11/07 22:43:48 Sensor sensor1 joined the network
2005/11/07 22:43:48 konstruktor
2005/11/07 22:43:48 Added IO::Socket::SSL=GLOB(0x8505fec) to select sets
2005/11/07 22:43:48 Waiting on select(2) syscall...
```

Wysłanie próbnej funkcji (info()) do wykonania na sensorze, który się właśnie połączył:

```
2005/11/07 22:43:48 Write watched handles: IO::Socket::SSL=GLOB(0x8505fec)
2005/11/07 22:43:48 Something (IO::Socket::SSL=GLOB(0x8505fec)) in write ready set
2005/11/07 22:43:48 Writing data to sensor
2005/11/07 22:43:48 Waiting on select(2) syscall...
```

```
2005/11/07 22:43:48 Write watched handles:
2005/11/07 22:43:48 Something (IO::Socket::SSL=GLOB(0x8505fec)) in read ready set
2005/11/07 22:43:48 Reading data from sensor
2005/11/07 22:43:48 Odpowiedz sensora:
2005/11/07 22:43:48 1
2005/11/07 22:43:48 Sensor replied: 1
2005/11/07 22:43:48 Waiting on select(2) syscall...
```

Przyjęcie kolejnego połączenia (administrator):

```
2005/11/07 22:44:01 Something (IO::Socket::SSL=GLOB(0x84eddd0)) in read ready set
2005/11/07 22:44:01 Client connected from 127.0.0.1
2005/11/07 22:44:01 Administrator connected from 127.0.0.1
```

Obsluga rozkazu otrzymanego z węzła-administratora i odesłanie odpowiedzi:

```
2005/11/07 22:44:08 Something (IO::Socket::SSL=GLOB(0x81e5ba0)) in read ready set
2005/11/07 22:44:08 Processing data from server.
2005/11/07 22:44:08 Running getName() in list context
2005/11/07 22:44:08 Function result: main
2005/11/07 22:44:08 Waiting on select(2) syscall...
2005/11/07 22:44:08 Write watched handles: IO::Socket::SSL=GLOB(0x81e5ba0)
2005/11/07 22:44:08 Something (IO::Socket::SSL=GLOB(0x81e5ba0)) in write ready set
2005/11/07 22:44:36 Something (IO::Socket::SSL=GLOB(0x81e5ba0)) in read ready set
2005/11/07 22:44:36 Processing data from server.
2005/11/07 22:44:36 Running
runOnNode(sensor1,addService,0.0.0.0,tcp,6070,services/finger) in list context
2005/11/07 22:44:36 Odpowiedz sensora:
2005/11/07 22:44:36 0
2005/11/07 22:44:36 Function result: 0
2005/11/07 22:44:36 Waiting on select(2) syscall...
2005/11/07 22:44:36 Write watched handles: IO::Socket::SSL=GLOB(0x81e5ba0)
2005/11/07 22:44:36 Something (IO::Socket::SSL=GLOB(0x81e5ba0)) in write ready set
2005/11/07 22:44:36 Waiting on select(2) syscall...
2005/11/07 22:44:36 Write watched handles:
```

Obsługa zamnknięcia połączenia przez konsolę administracyjną:

```
2005/11/07 22:44:40 Something (IO::Socket::SSL=GLOB(0x81e5ba0)) in read ready set
2005/11/07 22:44:40 Processing data from server.
2005/11/07 22:44:40 My master closed connection.
```

Obsługa zamknięcia połączenia przez sensor:

```
2005/11/07 22:44:48 Something (IO::Socket::SSL=GLOB(0x8505fec)) in read ready set
2005/11/07 22:44:48 Reading data from sensor
2005/11/07 22:44:48 Sensor closed connection
2005/11/07 22:44:48 Removing sensor sensor1
2005/11/07 22:44:48 Destruktor Sensora sensor1
2005/11/07 22:44:48 Waiting on select(2) syscall...
```

Obsługa połączenia, w którym nie został przedstawiony przez klienta certyfikat podpisany certyfikatem serwera centralnego:

```
2005/11/07 22:49:40 Something (IO::Socket::SSL=GLOB(0x84eddd0)) in read ready set
2005/11/07 22:49:40 Client connected from 127.0.0.1
2005/11/07 22:49:40 Unauthorized connection dropped
2005/11/07 22:49:40 Waiting on select(2) syscall...
```

Zakończenie działania serwera po otrzymaniu sygnału SIGINT:

```
2005/11/07 22:49:40 Write watched handles:
2005/11/07 22:49:44 Caught SIGINT.
```

```
2005/11/07 22:49:44 Closed main socket.
```

Węzeł podrzędny

Uruchomienie węzła podrzędnego i oczekiwanie w pętli na możliwość ustanowienia połączenia z serwerem centralnym:

```
2005/11/07 23:06:22 konstruktor Node
Node=HASH(0x8288910)
2005/11/07 23:06:22 Starting node sensor1
2005/11/07 23:06:22 Connecting to 127.0.0.1:9000...
2005/11/07 23:06:22 unable to create socket: IO::Socket::INET configuration failed
2005/11/07 23:06:22 Retrying in 4 seconds
2005/11/07 23:06:26 Connecting to 127.0.0.1:9000...
2005/11/07 23:06:26 unable to create socket: IO::Socket::INET configuration failed
2005/11/07 23:06:26 Retrying in 4 seconds
```

Połączenie z serwerem centralnym (rozpoznanie podpisu certyfikatu serwera głównego na certyfikacie przedstawionym przez drugą stronę):

```
2005/11/07 23:06:30 Connecting to 127.0.0.1:9000...
2005/11/07 23:06:30 Certificate's subject: /C=PL/O=HoneySpy network/OU=Master
Server/CN=Master
2005/11/07 23:06:30 Certificate's issuer: /C=PL/O=HoneySpy network/OU=Master
Server/CN=Master
2005/11/07 23:06:30 Certificate recognized.
2005/11/07 23:06:30 Using cipher: AES256-SHA
2005/11/07 23:06:30 Entering main loop - node sensor1
2005/11/07 23:06:30 Waiting on select(2) syscall...
```

Wykonanie rozkazu otrzymanego od serwera nadrzędnego:

```
2005/11/07 23:06:30 Something (IO::Socket::SSL=GLOB(0x84ede58)) in read ready set
2005/11/07 23:06:30 Processing data from server.
2005/11/07 23:06:30 Running info() in scalar context
2005/11/07 23:06:30 Jestem wezel sensor1
2005/11/07 23:06:30 Function result: 1
2005/11/07 23:06:30 Waiting on select(2) syscall...
```

Wykorzystywane moduły Perla

Do prawidłowego działania HoneySpy zalecane są następujące moduły Perla:

- Net::Pcap
- Net::PcapUtils
- Socket?
- Curses
- Net::Server
- Net::p0f
- Log::log4perl
- IPTables?
- IP::Country
- Net::IFconfig

- Net::MAC::Vendor?
- Net::MacMap::vendor?
- IP::Country
- Geography::Countries
- IO::Socket::INET
- IO::Socket::SSL
- Params::Validate
- Template::Toolkit
- CGI::Session?
- Template::Plugin::Session?

Inne tego typu narzędzia

Projekt HoneySpy może się wydawać podobny do innych tego typu narzędzi, na przykład do znanego honeyd, jednak zaplanowaliśmy inną architekturę rdzenia systemu, jak również nie chcemy powielać modułów usług, które sa już napisane dla honeyd. Namiastki usług powinny być po prostu skryptami wywoływanymi w taki sposób jak ma to miejsce w super-serwerze (inetd, xinetd itp). Dzięki takiemu podejściu skrypty napisane dla innych narzędzi Honeypot będą mogły być podłączone do HoneySpy, jak równiez stworzone przez nas moduły będą mogły być przeniesione na przykład do honeyd.

Honeyd w celu fałszowania adresów MAC oraz charakterystyki stosu, zawiera pełną implementację stosu TCP/IP w przestrzeni użytkownika i wysyła przez gniazdo pełne ramki w warstwie liniowej.

HoneySpy wykorzystuje różne gotowe i sprawdzone narzędzia w celu uzyskania takiego samego rezultatu, ale w prostszy sposób.

Nad projektem pracują

- Robert Nowotniak
- Michał Wysokiński

Adresy związane z projektem

- Repozytorium SVN projektu
  http://svn.berlios.de/viewcvs/honeyspy/
- Strona projektu na serwerze Berlios
  http://honeyspy.berlios.de/
- Informacje o projekcie z serwera Berlios
  http://developer.berlios.de/projects/honeyspy/
- Aktualna dokumentacja projektu
  http://robert.nowotniak.com/honeyspy/