T*omasz GRUDZIECKI*[1]

WSPÓŁCZESNE ZAGROŻENIA W SIECI INTERNET ORAZ SPOSOBY WALKI Z NIMI PRZY POMOCY SYSTEMÓW-PUŁAPEK

Streszczenie

Kilka lat temu Internet był pełen zagrożeń propagujących się w sposób aktywny poprzez skanowanie. Obecnie większość komputerowych przestępców przerzuciło się na atakowanie z wykorzystaniem aplikacji klienckich, w szczególności przeglądarek internetowych. Jedną z efektywniejszych metod wykrywania nowych oraz śledzenia zagrożeń są pułapki zwane honeypotami.

Abstract

A few years ago Internet was full of threats that propagate actively through scanning. Nowadays most computer criminals are focused on client-side threats, especially attacks against, or involving the use of, web browsers. A very effective method of detecting new threats (both: server-side and client-side) are honeypots – traps designed to help detect malware and fight botnets.

1 WPROWADZENIE

Analizując historię zagrożeń propagujących się przez sieć Internet zauważyć można ewolucję zarówno celu, w jakim wykorzystywano złośliwe oprogramowanie tzw. malware (ang. *malicious software),* jak i sposobów infekcji. Celem stały się pieniądze, chęć zysku. Natomiast sposobem, by ten cel osiągnąć, stały się botnety – sieci przejętych komputerów. Aby przejąć jak największą liczbę maszyn infekując je, wykorzystuje się podatności w aplikacjach bądź usługach. Nośnikiem tzw. exploita, czyli kodu wykorzystującego daną podatność mogą być zarówno robaki - samopropagujące się przez sieć złośliwe aplikacje, jak i złośliwe strony internetowe czy pliki PDF.

Wszystkie te zagrożenia są coraz bardziej wyrafinowane, coraz groźniejsze i jest ich coraz więcej. Aby móc je wykrywać, śledzić oraz z nimi walczyć, należy posłużyć się

[1] Mgr inż. Tomasz Grudziecki jest specjalistą od bezpieczeństwa sieciowego w Zespole Projektów Bezpieczeństwa w CERT Polska (Computer Emergency Response Team) działającego w ramach instytutu badawczego Naukowa i Akademicka Sieć Komputerowa. Ponadto wykłada w Warszawskiej Wyższej Szkole Informatyki.

specjalnie do tego celu stworzonymi narzędziami. Jednym z nich są systemy pułapek nazywane honeypotami (ang. *honeypots*).

2 ATAKI NA APLIKACJE SERWEROWE

Początkowo schemat ataku był prosty: należało znaleźć potencjalną ofiarę, a następnie zaatakować ją wykorzystując jedną z podatności obecną w jej systemie. Podatność taka, zwana także luką (ang. *vulnerability*), mogła istnieć zarówno w samym systemie operacyjnym (np. w implementacji stosu TCP/IP), jak i w aplikacji świadczącej usługi serwerowe, czyli oczekującej na połączenie od klienta (np. serwer MS SQL).

2.1 Usługa serwerowa pod obstrzałem

Usługa serwerowa jest stosunkowo łatwa do wykrycia. Spowodowane jest to jej podstawową funkcją – serwowania usługi klientowi. Funkcja ta wymaga czekania (słuchania) na połączenie inicjalizowane przez klienta. Skoro to klient nawiązuje połączenie do usługi serwerowej, musi znać jej adres IP (warstwa trzecia, sieciowa modelu ISO/OSI) oraz port (warstwa czwarta, transportowa modelu ISO/OSI). Aby ułatwić komunikację w sieciach opartych na protokołach TCP(UDP)/IP, stworzono listę odzwierciedlającą numery portów TCP i UDP na popularne usługi, które powinny słuchać na tych portach (jest to lista tzw. *well-known ports*). W związku z tym atakujący może próbować „na ślepo" połączyć się na port, na którym powinna działać podatna usługa. Atakować w ten sposób można losowe bądź kolejne adresy IP.

Aby zoptymalizować proces znalezienia podatnego serwera i wykorzystania luki, właściwy atak poprzedzony jest prostym z technicznego punktu widzenia skanowaniem. Skanowanie najczęściej polega na próbach łączenia się na losowe bądź kolejne, czyli zwiększane o jeden adresy IP na konkretny port bądź listę portów. W ten sposób pozyskać można zbiór adresów IP, pod którymi działają serwery mające otwarty port, na którym może działać podatna usługa. Dopiero na adresy IP z tej listy może zostać wysłany exploit, który umożliwi dalszą penetrację systemu aż do przejęcia kontroli nad fragmentem lub całością serwera.

Przeniesienie działania usługi na inny nieoczywisty port na niewiele się zda, gdyż skanowanie, czyli próby połączeń, może odbywać się także na inne (losowe bądź kolejne) porty. Z czasem moc obliczeniowo komputerów rosła w szybkim tempie, tak samo jak przepustowość łączy internetowych. W wyniku tego skanowania oraz ślepe próby wykorzystania luk były efektywnym sposobem infekcji systemów i samo propagacji złośliwego oprogramowania nazwanego robakami internetowymi (ang. *Internet worms*).

2.2 Wykrywanie ataków na aplikacje serwerowe

Propagacja złośliwego oprogramowania poprzez ślepe skanowanie jest efektywna, ale także efektowna. Zakrojone na szeroką skalę skanowania oraz mające szybkie tempo propagacja robaków jest łatwo zauważalna przez wszelkiego rodzaju narzędzia monitorujące sieć. W takim przypadku nie może być mowy jednocześnie o cichym oraz masowym wykorzystaniu jakiejś podatności.

W szczególnych przypadkach, gdy usługa serwowana nie jest dla klientów z dowolnego miejsca z Internetu, lecz dla wybranej grupy (np. lokalnej sieci bądź ograniczonej puli adresów publicznych), łatwo ruch będący próbami połączeń z niedozwolonych adresów odfiltrować na firewallu. Inne narzędzia potrafią rozpoznawać żądania (ang. *requests*) klienta do serwera (analiza protokołów wyższych warstw, np. WAF[2]), bądź rozpoznawać fragmenty exploitów (systemy IDS[3]/IPS[4]). Istnieją także systemy wykrywające anomalie w ruchu, a także hybrydy łączące kilka funkcji w celu poprawy skuteczności ochrony.

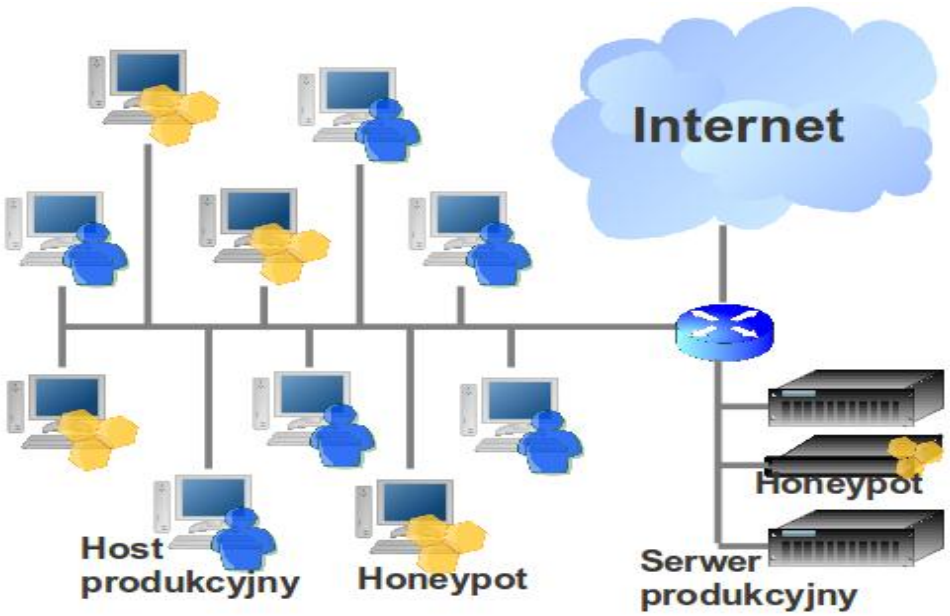
Wymienione wyżej rozwiązania, choć nie zapewnią w pełni bezpieczeństwa aplikacjom serwerowym, mogą w znaczącym stopniu korzystnie wpłynąć na bezpieczeństwo sięci i będących w niej systemów.

2.3 Pułapki w postaci „garnków miodu”

W celu walki z zagrożeniami propagującymi się przez sieć w sposób aktywny stworzono pojęcie pułapek, tzw. z ang. honeypots (w dosłownym tłumaczeniu „garnek miodu”). Pułapka ma za zadanie symulować podatny system/aplikację (bądź cały segment sieci składający się z podatnych serwisów) i czekać na atak. W tym celu do honeypotów przypisane są nieużywane adresy IP. Ponieważ do takiego systemu nie powinien dotrzeć żaden legalny ruch, każda próba połączenia do pułapki z definicji uznana jest za co najmniej podejrzaną. Jeżeli pułapka wejdzie w interakcję z atakującym, to dodatkowo nie tylko pozyskane zostaną informację o celu i formie ataku, ale także exploit na konkretną lukę. Specyfika działania honeypotów pozwala na wykrywanie – oprócz znanych zagrożeń – także zupełnie nowych, w skrajnym przypadku również tzw. ataków 0-day[5]. Jeżeli pułapki zostaną rozmieszczone logicznie pomiędzy adresami IP produkcyjnymi (wymieszane), mogą służyć do wykrywania infekcji wewnątrz takiej sieci – zarażony samopropagującym się złośliwym oprogramowaniem host najprawdopodobniej będzie próbował

[2] WAF – ang. *Web Application Firewall*
[3] IDS – ang. *Intrusion Detection System* – system wykrywania włamań
[4] IPS – ang. *Intrusion Prevention System* – system zapobiegania włamaniom
[5] Atak 0-day – atak wykorzystujący nieznaną wcześniej podatność

infekować w pierwszej kolejności inne hosty w bezpośrednim sąsiedztwie sieciowym. Z tego powodu pułapki typu honeypot stały się efektywnym narzędziem nie tylko do wykrywania nowych oraz śledzenia istniejących zagrożeń, ale także do ochrony sieci od wewnątrz. Jeżeli infrastruktura sieciowa jest podzielona na segmenty, pułapki powinny znajdować się w każdym z nich i odzwierciedlać usługi charakterystyczne dla danej podsieci (patrz rys.1). Przykładem wykorzystania honeypotów do aktywnej ochrony infrastruktury sieciowej jest projekt ARAKIS [1].



Rys 1. Przykładowe rozmieszczenie honeypotów serwerowych w infrastrukturze sieciowej podzielonej na segmenty

2.3.1 Serwerowe honeypoty niskointeraktywne

Ze względu na złożoność interakcji honeypoty dzielą się na wysokointeraktywne i niskointeraktywne. Honeypoty niskointeraktywne symulują mniej lub bardziej udanie podatne usługi lub aplikacje. W szczególności potrafią udawać także cały system operacyjny z typowymi dla niego usługami sieciowymi, np. współdzielenie zasobów w systemach z rodziny Microsoft Windows.

Aplikacje udające prawdziwe usługi nie mogą tego robić w sposób idealny. W szczególności nie sposób symulować nieznane podatności. Z drugiej strony programy symulu-

jące usługi mają mniejsze zapotrzebowanie na zasoby systemowe niż rzeczywiste programy. Ponadto są bezpieczniejsze, gdyż nie mogą zostać skompromitowane przy pomocy exploitów, które przechwytują. Takie usługi można także łatwiej skalować oraz rozmieszczać je w różnych lokalizacjach sieciowych.

Jednymi z najczęściej wykorzystywanych darmowych „serwerowych" honeypotów niskointeraktywnych są honeyd[6] oraz dionaea[7].

2.3.2 Serwerowe honeypoty wysokointeraktywne

Honeypoty wysokointeraktywne często są rzeczywistymi systemami, na które działają rzeczywiste podatne usługi sieciowe. Poprzez odizolowanie takiego systemu i prowadzenie pełnej obserwacji jego zachowania (np. monitorowanie operacji zapisów i odczytów dyskowych, pamięci, uruchomionych procesów i aktywności sieciowej) można wykrywać ataki na daną usługę. Zebrane w ten sposób dane będą kompletne i bardziej konkretnie – w porównaniu do honeypotów niskointeraktywnych – będzie można zidentyfikować cel (konkretną lukę) oraz sposób przeprowadzenia ataku (exploit wraz z informacjami, co się dzieje z systemem po infekcji).

Jednakże takie rozwiązanie jest bardziej skomplikowane i wymaga większych zasobów zarówno systemowych, jak i sprzętowych. Ponadto wymagane jest posiadanie oprogramowania w odpowiedniej wersji. Wynika to z faktu, że exploit dedykowany jest konkretnej podatności, a każda wersja danej aplikacji zawiera inny zestaw luk i słabości. W przypadku całego systemu operacyjnego jako takiego, sytuacja ta zmienia się z każdą aktualizacją. Poziom „aktualności" danego programu lub systemu nazywane jest z ang. *patch-level*, co można przetłumaczyć na język polski jako „poziom załatania".

3 EWOLUCJA ZAGROŻEŃ W STRONĘ APLIKACJI KLIENCKICH

Internet w ostatnich latach stał się ogólnodostępnym i popularnym dobrem. Szczególną ciągle rosnącą popularnością cieszą się serwisy oparte na stronach internetowych (WWW – ang. *World Wide Web*). Wraz z tą popularnością rosły też znaczenie i możliwości przeglądarek internetowych, które służą do prezentowania końcowemu użytkownikowi strony internetowej. By zaspokoić potrzebę użycia nowych technologii w tworzeniu serwisów internetowych pojawiło się wiele rozszerzeń oraz wtyczek (ang. *extension* oraz *plug-in*) do przeglądarek dostarczanych przez trzecich producentów niezwiązanych

[6] Honeyd – niskointeraktywny honeypot tworzący wirtualne hosty w sieci komputerowej: http://www.-honeyd.org/faq.php#what

[7] Dionaea – najnowszej generacji niskointeraktywny honeypot: http://dionaea.carnivore.it/

z twórcami przeglądarek. Przykładami takimi są: wtyczka do obsługi animacji flash, do wyświetlania w przeglądarce dokumentów w formacie PDF (obie dostarczane przez firmę Adobe), do odtwarzania filmów w formacie QuickTime (dostarcza Apple), czy do obsługi platformy Java (dostarcza Oracle). Przeglądarki internetowe mogą zastępować inne aplikacje oferując w połączeniu z odpowiednimi serwisami internetowymi funkcjonalność wypieranych programów, np. odtwarzanie filmów i telewizji, czy tworzenie i edycję dokumentów.

W konsekwencji przeglądarki internetowe stały się jedną z najważniejszych aplikacji w systemie operacyjnym komputera biurkowego (ang. *desktop computer*). Ta cecha nie została niezauważona przez przestępców komputerowych. W roku 2007 rozpoczęła się zmiana wektora ataku z wykorzystywania podatności w usługach sieciowych systemów operacyjnych, na wykorzystywanie luk w aplikacjach klienckich (w szczególności przeglądarek internetowych i ich komponentów) [2]. Obecne trendy [3] wskazują na to, że chociaż zagrożenia propagujące się w sposób aktywny (przez skanowanie) nie zniknęły, to większość infekcji odbywa się za pośrednictwem aplikacji klienckich, w szczególności z wykorzystaniem techniki drive-by download opisanej w dalszej części artykułu.

3.1 Techniki wykorzystywane w atakach przez aplikacje klienckie

Główną cechą zagrożeń propagujących się przez przeglądarki internetowe różniącą je od zagrożeń samopropagujących się przez usługi sieciowe jest brak skanowania, czyli aktywnych prób łączenia się w celu poszukiwania ofiary. W przypadku „zagrożeń klienckich" (ang. *client-side threats*) to ofiara nawiązuje nieświadomie połączenie z atakującym poprzez (najczęściej skompromitowaną) stronę www. Zazwyczaj polityka bezpieczeństwa statystycznej sieci korporacyjnej pozwala na połączenia komputerów użytkowników końcowych (pracowników) do stron internetowych wszystkich lub co najmniej wybranych. Ta specyfika powoduje, że tego typu atak nie może zostać zablokowany na urządzeniach filtrujących takich jak firewall – ruch odbywa się na dozwolonym porcie i z użyciem dozwolonych protokołów (HTTP). W sieciach domowych ruch związany ze stronami www jest praktycznie zawsze przepuszczany. W dalszej części artykułu zostanie wykazane, że systemy IDS/IPS, a także oprogramowanie antywirusowe oparte na porównywaniu sygnatur nie radzi sobie z „atakami klienckimi".

Pasywny sposób pozyskiwania ofiary polegający na czekaniu, aż ofiara sama wejdzie na złośliwą stronę internetową, powoduje jednocześnie, że trudniej zauważyć źródło ataku. Atakujący nie wystawia się i nie zwraca na siebie uwagi aktywnym skanowaniem. W przypadku, gdy jakiś nowy robak rozpoczynał autopropagację przez Internet, był od początku śledzony przez specjalistów, co powodowało szybką reakcję przemysłu zaj-

mującego się zabezpieczeniami komputerowymi. Ataki przez strony internetowe cechują się tym, że skompromitowane strony mogą pozostawać niewykryte przez stosunkowo długi czas. Atakowana jest nie jak największa liczba potencjalnych ofiar (cecha charakterystyczna dla robaków internetowych), lecz dużo mniejsza grupa dobrze dobranych ofiar.

Owa selekcja w wyborze ofiar jest inną ważną cechą „zagrożeń klienckich". Zarówno w nagłówkach protokołu HTTP docierających do serwera WWW, jak i poprzez kod umieszczony na stronie internetowej (najczęściej jest to JavaScript) atakujący może określić profil potencjalnej ofiary. W szczególności chodzi o informacje o systemie operacyjnym, używanej wersji przeglądarki internetowej oraz wersjach zainstalowanych dodatków do niej. W ten sposób ofiara spełniająca pewien profil (posiadająca oprogramowanie w wersji umożliwiającej przeprowadzenie ataku zakończonego sukcesem) może być przekierowywana do serwera, na którym następuje atak, a co do innych klientów nie jest podejmowane żadne działanie lub są przekierowywani na niezłośliwą stronę.

Tabela 1. Porównanie zagrożeń propagujących się przez skanowanie i przez aplikacje klienckie

	Zagrożenia „server-side"	**Zagrożenia „client-side"**
Sposób propagacji	Aktywny (przez skanowanie)	Pasywny (oczekiwanie na nawiązanie połączenia przez ofiarę)
Cel ataku	Usługi sieciowe otwarte na dostęp z zewnątrz	Aplikacje klienckie, które korzystają z usług sieciowych
Dobór ofiar	Atakowanie możliwie jak największej liczby potencjalnych ofiar	Stosunkowo niewielka, lecz dobrze dobrana grupa ofiar podatnych na atak
Wykrywalność ataku	Masowa autopropagacja (skanowanie) jest łatwo zauważalna przez analizatory ruchu sieciowego	Brak masowości, faza ataku niknie w natłoku legalnego ruchu http

3.1.1 Technika drive-by download

Często używaną przez przestępców komputerowych techniką wykorzystania luki w przeglądarce internetowej bądź jej dodatku i w efekcie zainfekowanie komputera ofiary, jest technika *drive-by download*. Cechuje się minimalizacją interakcji z człowiekiem – wymagane jest jedynie wejście z użyciem przeglądarki na stronę internetową. Wykorzystanie luki, ściągnięcie na komputer ofiary złośliwego kodu i wykonanie go odbywa się już bez wiedzy oraz interakcji użytkownika.

Aby wykorzystać podatność, w przeglądarce musi zostać wykonany złośliwy fragment kodu. Aby przeglądarka go zinterpretowała musi się znaleźć w kodzie strony w formie zrozumiałej dla przeglądarki lub jej dodatku. Najpowszechniej stosowane do tego celu są skrypty napisane w języku JavaScript, oraz (rzadziej) VBscript. Wykorzystanie luki we wtyczce flash może wymagać osadzenia w stronie specjalnie spreparowanego pliku SWF [4], a dla wtyczki czytnika PDF może to być spreparowany plik PDF.

3.1.2 Złośliwy javascript

Język JavaScript pełni istotną rolę w procesie ataku i infekcji komputerów ofiar. Kod JavaScript jest powszechnie wykorzystywany, jako nośnik exploita do wykorzystania podatności. W zależności od miejsca, w którym znajduje się podatność, przy pomocy JavaScript-u można tworzyć także inne obiekty, np. kontrolki ActiveX. Skrypty JS są też używane jako element wspomagający w innych wektorach ataku – np. dokumenty PDF mają możliwość obsługi języka JavaScript (czytniki plików PDF są wyposażone w interpreter języka JavaScript), dlatego często złośliwy kod wykorzystujący lukę w czytniku PDF napisany jest z użyciem języka JavaScript [5].

Złośliwy kod JavaScript umieszczony na stronie nie musi służyć tylko i wyłącznie jako nośnik exploita. Bardzo często jest on także wykorzystywany w celu przekierowywania (czasami wielokrotnie) ofiary na dalsze strony internetowe. Jedną z najpopularniejszych technik jest wstawianie za pomocą skryptu JS elementu IFRAME[8] w kod skompromitowanej strony. Element taki serwuje kod innej strony www, czyli przekierowuje na kolejną stronę, która może przekierowywać dalej, aż do dotarcia do strony, w której nastąpi przesłanie exploita. Nierzadko zdarza się, że rozmiar takiego IFRAME-a jest zerowy (0x0 piksela). Pomimo tego dokument HTML, którego adres znajduje się w tym elemencie, zostanie wczytany i zinterpretowany przez silnik przeglądarki internetowej.

JavaScript nadaje się również bardzo dobrze do ukrywania wektora ataku – zarówno samego exploita, jak i wszystkich pośrednich czynności włącznie z przekierowywaniem. Ukrywanie polega na „zaciemnianiu" kodu JS (ang. *obfuscation*) [6]. Zaciemnianie ukrywa prawdziwy cel skryptu zarówno przed wzrokiem człowieka, ale przede wszystkim przed systemami ochrony opartymi na porównywaniu sygnatur (IDS/IPS, programy antywirusowe), analiza takiego JavaScript-a s jest utrudniona. Po dotarciu zaciemnionego kodu do przeglądarki jej silnik „dekoduje" go i uruchamia. Wykorzystując różne techniki

[8] IFRAME – element za którego pomocą w dokumencie HTML można zawrzeć inny dokument HTML: http://www.w3schools.com/tags/tag_iframe.asp

zaciemniania, stosując wielokrotne zaciemnianie wraz z mieszaniem tych technik, oraz rozmieszczając fragmenty całości kodu JavaScript w różnych miejscach na tej samej stronie lub na kolejnych znajdujących się na innych serwerach, przestępcy komputerowi powodują, że ten sam kod za każdym razem wygląda inaczej. Z tego powodu bezużyteczne stają się systemy ochrony infrastruktury IT bazujące na porównywaniu sygnatur (sygnatur złośliwego skryptu JS zaciemnionego wielokrotnie w różnych kombinacjach technik może być teoretycznie nieskończenie wiele). W ten sposób zaciemniać można sam exploit, ale również obiekty IFRAME wraz z adresami kolejnych stron, na które następują przekierowania. Na rysunku 2. przedstawiono przykład jednokrotnego zaciemnienia prostego kodu Java-Script o treści alert ("Malicious JavaScript!!!") przy pomocy czterech różnych technik. Za każdym razem kod wynikowy wygląda inaczej. Niemożliwe jest stworzenie wspólnej sygnatury opisującej wszystkie skrypty. Natomiast efekt wykonania wszystkich przykładów w przeglądarce internetowej będzie taki sam.

Rys 2. Przykład różnych typów zaciemnienia prostego kodu JavaScript

3.1.3 Przekierowania na kolejne strony www

Złośliwe strony internetowe, które bezpośrednio są odwiedzane przez nieświadomą ofiarę, w przeważającej większości nie zawierają właściwego kodu exploita, lecz jedynie przekierowują do kolejnego serwera WWW. Takich przekierowań może być więcej niż jedno, ale ostatecznie przeglądarka ofiary trafia na stronę, która serwuje exploit. Ostatnie ogniwo w łańcuchu przekierowań, na którym znajduje się kod wykorzystujący podatność w przeglądarce internetowej, nazywany jest *exploit-server* [7]. Serwery, które uczestniczą w komunikacji pomiędzy wejściem ofiary na pierwszą stronę, a dotarciem do *exploit-servera*, to tzw. serwery pośredniczące, lub krótko: pośrednicy.

W przeważającej większości pierwsza strona – bezpośrednio odwiedzana przez ofiarę – została skompromitowana i jej administrator lub tzw. webmaster nie mają świadomości tego faktu. Strona może być skompromitowana na wiele sposobów. Najpopularniejsze z nich to:

- poprzez luki w serwerze WWW lub systemie operacyjnym serwera,
- poprzez błędy w kodzie strony/aplikacji internetowej (w szczególności skryptach PHP, ASP, itp., w wyniku ataków typu *XSS*[9] lub *SQL injection*[10]),
- w wyniku przejęcia haseł do serwera WWW (konto FTP, shell, itp.) np. poprzez wcześniejszą infekcję komputera administratora/webmastera lub w wyniku ataku socjotechnicznego.

W wyniku skompromitowania na takiej stronie zostaje umieszczony (wstrzyknięty – z ang. *injected*) krótki kod, który przekierowuje na zewnętrzny serwer. Najczęściej jest to JavaScript (często zaciemniony) wpisujący w stronę niewidoczny obiekt IFRAME kierujący do serwera pośredniczącego.

Zadania pośredników są różne. Jednym z nich jest ukrycie dalszych pośredników oraz exploit-servera przed niechcianymi połączeniami – następuje próba zdiagnozowania, czy serwer ma do czynienia z rzeczywistą przeglądarką, czy może narzędziem do wykrywania „ataków klienckich". W tym celu stosuje się m.in. badanie nagłówków HTTP (szczególnie User-Agent i Referer), sprawdzanie ustawionych wcześniej ciasteczek, a także zliczanie czasu żądań do kolejnych pośredników. Innym zadaniem jest selekcja ofiar – kierowanie ofiar posiadających podatność X do exploit-servera A, a tych posiadających podatność Y do serwera B itd. Przekierowanie może odbywać się na poziomie kodu interpretowanego w przeglądarce (np. wspomniany wcześniej IFRAME, lub odnoś-

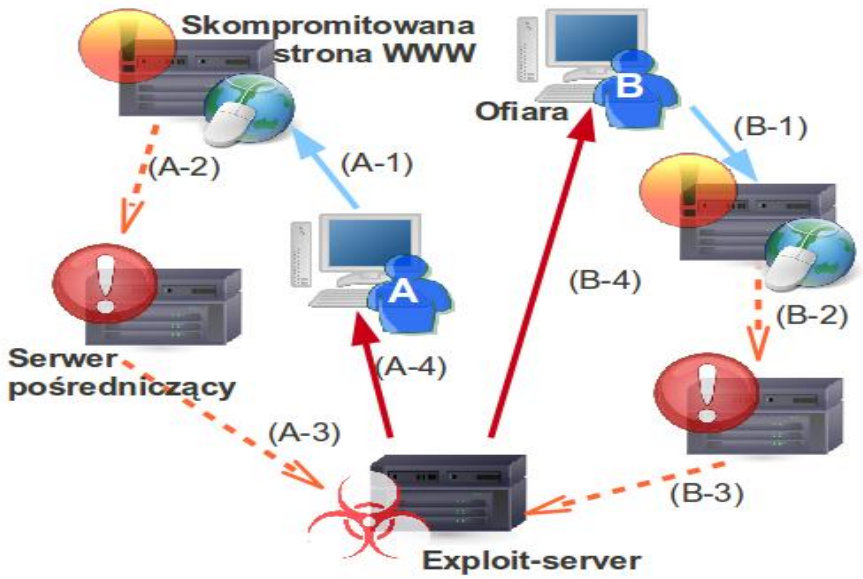
[9] XSS – ang. *Cross-Site Scripting* – metoda ataku polegająca na umieszczeniu nieautoryzowanego fragmentu kodu w treści strony WWW, patrz: http://pl.wikipedia.org/wiki/Cross-site_scripting

[10] SQL injection: atak na źle zabezpieczony serwis WWW pozwalający na wykonywanie nieautoryzowanych operacji na bazie danych z poziomu przeglądarki internetowej – patrz: http://pl.wikipedia.org-/wiki/SQL_injection

nik do skryptu JavaScript lub osadzonego obiektu znajdującego się na innym serwerze). Przekierowanie może także mieć miejsce na poziomie kodu wykonywanego po stronie serwera np. przy użyciu skryptów PHP, ASP oraz samego serwera WWW, np. przy pomocy kodów odpowiedzi HTTP – ang. *status codes* – 3xx [8], w szczególności 301 – Moved Permanently, 302 – Found, 303 – See Other).

Ostatnim ogniwem w łańcuchu przekierowań jest *exploit-server*. Na nim znajdują się exploity najczęściej na kilka lub kilkanaście podatności (w zależności od wersji systemu operacyjnego ofiary, jej przeglądarki i dodatków do niej). To z tego typu serwerów następuje rzeczywista infekcja. Do jednego *exploit-servera* może kierować wielu pośredników, a ofiary odwiedzające setki różnych skompromitowanych stron mogą *summa summarum* zostać zainfekowane z jednego *exploit-servera*.

Przykładowy system powiązań został zobrazowany na rys. 3. Zostali na nim przedstawieni dwaj użytkownicy: A i B. Użytkownik najpierw wchodzi na skompromitowaną stronę WWW (1), zostaje przekierowany na serwer pośredniczący (2), który kieruje do exploit-servera (3). Następnie serwowany jest kod, który infekuje go wykorzystując lukę w przeg-lądarce użytkownika (4).



Rys 3. Schemat przekierowań ofiary ze strony WWW do exploit-servera

3.1.4 Technika fast-flux

W ostatnich latach przestępcy komputerowi zaczęli powszechnie wykorzystywać technikę ukrywania serwerów WWW za nazwami domenowymi zwaną fast-flux. Technika ta polega na serwowaniu złośliwych stron internetowych poprzez ciągle zmieniający się zbiór adresów IP przejętych komputerów, które pełnią rolę pośredników [9]. Złośliwy kod dostępny jest pod pewnym adresem domenowym. Adres ten rozwiązywany jest (ang. *resolve*) na różne adresy IP. Ważność odpowiedzi DNS jest krótka (niskie TTL). Po upłynięciu czasu ważności kolejne zapytanie DNS o ten sam adres domenowy zwraca zupełnie inny zestaw adresów IP. Tych adresów IP mogą być setki oraz tysiące. Należą one do przejętych przez przestępców komputerów, które połączone są w sieć zwaną botnetem.

Głównym celem wykorzystania techniki fast-flux jest głównie ukrycie własnych serwerów, stworzenie infrastruktury odporniejszej na pełne wykrycie. Fast-flux może również pełnić rolę load-balancera, jak również rozproszonej sieci serwerów proxy.

4 HONEYPOTY DO WALKI Z ATAKAMI NA APLIKACJE KLIENCKIE

W przypadku zagrożeń propagujących się w sposób aktywny okazało się, że honeypoty są jedną z najskuteczniejszych metod wykrywania nowych rodzajów ataków. Niestety, standardowe podejście do pułapek nie sprawdza się w kontekście ataków na aplikacje klienckie. Z tego powodu zostało opracowane pojęcie honeypotów klienckich (ang. *client honeypot*, albo w skrócie *honeyclient*).

W przeciwieństwie do pułapek „serwerowych", honeypoty klienckie muszą aktywnie poszukiwać zagrożeń. W przypadku ataków przez przeglądarki internetowe oznacza to, że aktywność honeypota sprowadza się do przemieszczania się po stronach internetowych i badania ich zawartości pod kątem złośliwości. Jednakże analizy takie muszą uwzględniać techniki opisane w rozdziale 3.1. Z powodu łatwego wykrycia (częste połączenia do serwerów WWW z jednego adresu IP) honeypoty klienckie powinny działać przez serwery proxy, które często zmieniają adres IP a najlepiej są rozproszone w sieci (pod względem logicznym).

Klienckie honeypoty mogą być – tak jak „serwerowe" – podzielone na niskointeraktywne oraz wysokointeraktywne. Mają także podobne wady i zalety [6]: niskointeraktywne pułapki są szybsze i łatwiejsze w obsłudze i zarządzaniu, ale słabiej wykrywają nowe zagrożenia oraz rzadziej prowadzą do końcowej fazy infekcji (pozyskania złośliwego oprogramowania) – częściej kończą działanie na początkowej fazie ataku. Z kolei wysokointeraktywne honeypoty są wolniejsze i trudniej nimi zarządzać, ale potrafią

przejść przez wszystkie fazy infekcji pozyskując na końcu złośliwe oprogramowanie instalowane w systemie.

4.1 Klienckie honeypoty niskointeraktywne

Nistointeraktywny kliencki honeypot na pierwszy problem napotyka już na etapie rozwiązywania adresu domenowego na IP oraz pozyskiwania kodu strony internetowej. Mechanizmy obsługujące zapytania DNS muszą wykrywać technikę fast-flux. Jeżeli fast-flux zostanie wykryty, to należy to także uwzględnić w dalszej analizie. Powinno się także rozważyć sprawdzenie więcej niż jednego adresu IP z puli pozyskanej w odpowiedzi DNS.

W dalszym etapie działania honeypota mechanizmy służące do łączenia się z serwerem WWW oraz pobrania zawartości muszą jak najbardziej upodobnić się z zachowania do przeglądarki internetowej. Przede wszystkim należy zadbać o odpowiednie ustawienie pól w nagłówkach HTTP (szczególnie: User-Agent i Referer). Po udanym pobraniu następuje najtrudniejsza część do realizacji przez klienckiego honeypota – interpretacja i analiza kodu pobranej strony.

Testy wykonane w trakcie tworzenia systemu składającego się z wysoko- i niskointeraktywnych klienckich honeypotów HoneySpider Network (w skrócie: HSN) pokazały, że niskointeraktywne honeypoty oparte na wykrywaniu zagrożeń przez porównywanie kodu strony bądź ruchu HTTP z sygnaturami (czy to pochodzącymi z systemów IDS/IPS, czy programów antywirusowych) nie spełniają swojej roli z powodu dużej liczby fałszywych pominięć (z ang. *False-negatives*) [6]. Wynika to głównie z faktu wykorzystywania przez przestępców komputerowych opisanych w rozdziale 3.1.2 technik zaciemniania kodu JavaScript. Z tego powodu mało zadowalająca jest jakakolwiek statyczna analiza kodu strony WWW. Konieczne wydaje się więc użycie interpretera języka HTML oraz silnika JavaScript, który wykonując zaciemnione skrypty JS będzie w stanie zdekodować je do ostatecznej postaci. Podobnie jak w przypadku „serwerowych" honeypotów niskointeraktywnych trudno zasymulować nieznaną lukę w podatnej aplikacji, tak w tym przypadku nie może powieść się wykonanie skryptu JS, który wykorzystuje (nieznaną) lukę w silniku rzeczywistej przeglądarki internetowej. Dlatego exploity na nieznane luki (tzw. *0-day*) muszą być wykrywane w inny sposób.

4.1.1 Analiza złośliwych javascriptów w niskointeraktywnym honeypocie

Oprócz silnika JavaScript, twórcy systemu HoneySpider Network do analizy skryptów JS wykorzystali zaawansowane techniki uczenia maszynowego (ang. *machine learning*)

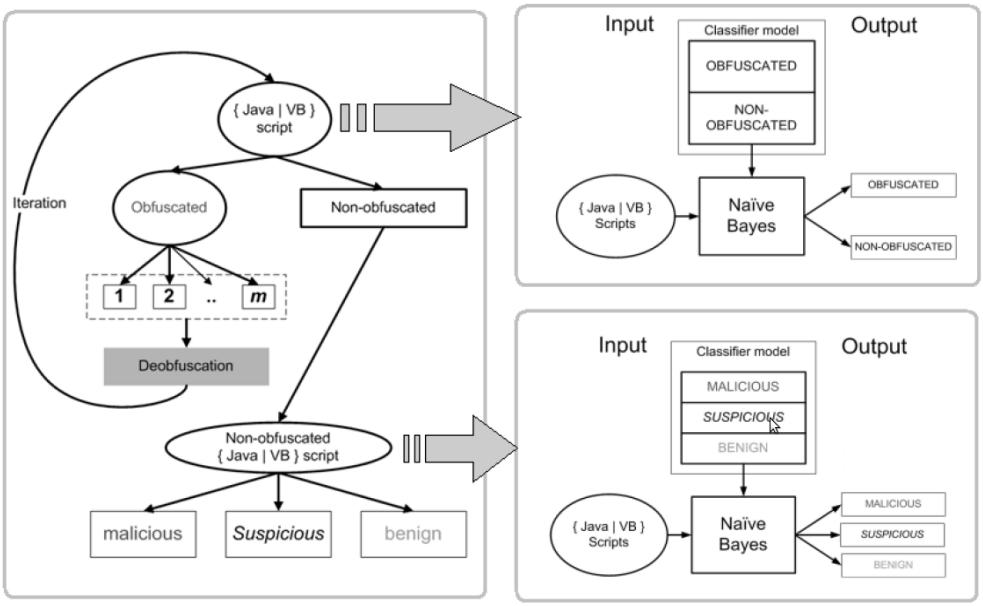
i klasyfikator Bayesowski (ang. *Naive Bayes Classifier*) [10]. Klasyfikator ten jest powszechnie wykorzystywany w narzędziach wykrywających spam w poczcie elektronicznej. System HSN korzysta z gotowego narzędzia opensource Weka[11] zawierającego szereg algorytmów uczenia maszynowego, w tym wspomniany klasyfikator. Analizator Bayesa jest wykorzystywany dwojako: najpierw w celu stwierdzenia, czy kod JavaScript jest zaciemniony, a następnie po zdekodowaniu rozstrzyga, czy niezaciemniony kod jest złośliwy. Aby klasyfikator mógł poprawnie rozpoznawać dane, musi zostać wcześniej nauczony. Zbiór danych uczących został stworzony przez człowieka (ręcznie dobrane skrypty, co do których istnieje pełna pewność, że są złośliwe), tak samo jak zbiór danych testujących. Analizator jest najpierw uczony na podstawie zbioru uczącego, a następnie jego umiejętności są weryfikowane z wykorzystaniem zbioru testującego. W trakcie działania honeypota oba zbiory są sukcesywnie wzbogacane o nowe skrypty JS.

Wykonane doświadczenia pokazały, że analiza w klasyfikatorze Bayesowskim kompletnego kodu JavaScript nie jest optymalna i duże lepsze wyniki daje podzielenie skryptu na mniejsze fragmenty o stałej wielkości z wykorzystaniem mechanizmu przesuwającego się okna (o jeden znak) (ang. *sliding window mechanism*), a następnie analiza każdego z nich osobno [6]. Poszczególne fragmenty skryptu nazywane są n-gramami (ang. *ngrams*), a do ich tworzenia wykorzystywane jest narzędzie opensource Google N-grams[12]. Rozbicie JavaScripta na n-gramy dotyczy zarówno analizy rozstrzygającej o zaciemnieniu kodu skryptu, jak i o jego złośliwości. Na rysunku 4 pokazano schemat procesu analizy kodu JavaScript (oraz VBscript) metodami heurystycznymi z wykorzystaniem klasyfikatora Bayesa.

Jak zostało opisane w rozdziale 3.1 kod JavaScript często służy do przekierowywania przeglądarki ofiary na kolejne strony. Niskointeraktywny honeypot powinien – udając zachowanie przeglądarki – podążać za takimi przekierowaniami. Jest to kolejny powód – poza dekodowaniem zaciemnionych skryptów JS – do użycia silnika JavaScript.

[11] Weka – Zbiór algorytmów uczenia maszynowego napisany w języku Java: http://www.cs.waikato.ac.nz/ml/weka/

[12] Google N-grams - ppakiet oprogramowania do analiz typu n-gram: http://code.google.com/p/ngrams/



Rys. 4. Proces analizy kodu JS z wykorzystaniem klasyfikatora Bayesa na przykładzie systemu HoneySpider Network
Źródło: [10]

4.2 Klienckie honeypoty wysokointeraktywne

Opisane w rozdziale 4.1 problemy, na jakie napotyka niskointeraktywny kliencki honeypot, nie występują w przypadku pułapki wysokointeraktywnej, ponieważ użyta jest rzeczywista przeglądarka działająca na rzeczywistym systemie operacyjnym. Problemem w tym wypadku jest wspomniana wcześniej wydajność i prędkość działania (pomimo użycia wirtualizacji). Ponadto, taki honeypot będzie w stanie wykryć jedynie ataki na posiadaną wersję przeglądarki internetowej z konkretnym zestawem wtyczek działającej na konkretny systemie operacyjnym. Rozwiązaniem jest uruchomienie maksymalnie dużej liczby instancji wysokointeraktywnego honeypota, każda w innej konfiguracji (kombinacja systemów operacyjnych wraz z ich poziomem aktualizacji, oraz różnymi przeglądarkami internetowymi w różnych wersjach i z różnym zestawem dodatków). Problemem w takim wypadku może być wydajność serwerów, na których będzie działał taki zestaw honeypotów, a także kwestie finansowe wynikające z konieczności zakupu licencji na oprogramowanie.

Oddzielnym problemem jest sposób monitorowania systemu wystawionego na atak, czyli de facto wykrywania infekcji. Wysokointeraktywny kliencki honeypot, jakim jest

Capture-HPC[13], zapisuje zmiany na dysku oraz rejestrach systemu MS Windows, jakie miały miejsce podczas odwiedzin badanej strony internetowej. Z założenia jest to dobre rozwiązanie, ale w praktyce generuje dużo fałszywych alarmów (ang. *false-positives*). Korzystając z Capture-HPC należy opracować kompletne tzw. listy wykluczeń zawierające procesy, które mają być pomijane przez honeypota w trakcie analizy stanu systemu-pułapki.

5 PODSUMOWANIE

W dzisiejszych czasach zagrożenia pochodzące z Internetu są coraz poważniejsze, a wektor ataku zmienia się coraz szybciej i częściej. W chwili obecnej głównym nośnikiem zagrożeń dla użytkownika końcowego (domowego) są aplikacje klienckie, a w szczególności przeglądarki internetowe wraz z dodatkami. Nie oznacza to jednak, że zagrożenia „serwerowe" - propagujące się przez aktywne skanowanie, zniknęły nagle z Internetu. Wręcz przeciwnie – również one stanowią cały czas niebezpieczeństwo, a użytkownik może być teraz zaatakowany na różnych frontach.

W niniejszym artykule – oprócz zawarcia skrótowego opisu metod i sposobów funkcjonowania typowych zagrożeń – została zwrócona szczególna uwaga na honeypoty – rodzaj narzędzi, przy pomocy których można efektywnie wykrywać i śledzić owe zagrożenia. Mam nadzieję, że zachęciłem do zapoznania się i używania, a być może także rozwijania honeypotów.

Literatura

1. ARAKIS: System wczesnego ostrzegania o zagrożeniach w sieci rozwijany przez Naukową i Akademicką Sieć Komputerową: http://arakis.pl/pl/-faq.html.
2. Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, Nagendra Modadugu; *The Ghost In The Browser - Analysis of Web-based Malware*, Konferencja HotBots 2007 (Cambridge), Usenix.
3. Raport CERT Polska za rok 2010, http://www.cert.pl/PDF/Raport-_CP_2010.-pdf.
4. SWF – format pliku dla technologii Adobe Flash: http://www.adobe.-com/-devnet/swf.html.
5. Paweł Jacewicz, *Złośliwe pliki PDF*, Biuletyn NASK nr 3/2009, s.11-13.

[13] Capture-HPC – wysokointeraktywny honeypot wykrywający ataki na aplikacje klienckie: https://projects.honeynet.org/capture-hpc.

6. Piotr Kijewski, Carol Overes, Rogier Spoor, *The HoneySpider Network – fighting client-side threats*, konferencja roczna Forum for Incident Response and Security Teams 2008.
7. Christian Seifert, Ramon Steenson, Thorsten Holz, Bing Yuan, Michael A. Davis, *Know Your Enemy: Malicious Web Servers*, The Honeynet Project 2008, http://www.honeynet.org/papers/mws.
8. Network Working Group, document RFC 2616: *Hypertext Transfer Protocol – HTTP/1.1*, sekcja 6.1.1, http://tools.ietf.org/html/rfc2616.
9. *Know Your Enemy: Fast-Flux Service Networks*, The Honeynet Project 2007, http://www.honeynet.org/papers/ff.
10. Paweł Jacewicz, Tomasz Grudziecki, *Potwornie Dziurawy Format – JavaScript a luki w PDF*, konferencja Pingwinaria 2010 (Spała, Polska).