

WOJSKOWA AKADEMIA TECHNICZNA

im. Jarosława Dąbrowskiego

Wydział Cybernetyki



PRACA DYPLOMOWA

Studia stacjonarne I°

Metody wstrzykiwania kodu - analiza i przykłady praktyczne

Autor
sierż. pchor. Damian Krata

Promotor
dr inż. Piotr Bora

Konsultant
mgr inż. Michał Glet

Warszawa 2017

OŚWIADCZENIE

„Wyrażam zgodę na udostępnianie mojej pracy przez
Archiwum WAT”

Dnia

.....

(podpis)

Pracę przyjąłem

promotor pracy
dr inż. Piotr Bora

Spis treści

1	Opis algorytmu wstrzykiwania kodu	10
1.1	Wybór celu	10
1.2	Określenie potrzeb	11
1.3	Algorytm	14
2	Charakterystyka i sposoby realizacji poszczególnych etapów algorytmu	15
2.1	Tworzenie wstrzykiwalnego kodu	15
2.1.1	Shellcode	16
2.1.2	Plik PE	19
2.2	Wybór celu	20
2.2.1	Wstrzykiwanie kodu do istniejącego procesu	20
2.2.2	Wstrzykiwanie kodu do nowo uruchomionego procesu	21
2.3	Wpisywanie kodu do zdalnego procesu	22
2.3.1	Alokacja fragmentu pamięci	22
2.3.2	Dodanie nowej sekcji	23
2.4	Metody przekierowania do wstrzykniętego kodu	24
2.4.1	Uruchomienie kodu w nowym wątku	24
2.4.2	Użycie bibliotek nieudokumentowanych	25
2.4.3	QueueUserAPC()	25
2.4.4	Nadpisanie punktu wejścia procesu	26
3	Opis przykładowej implementacji	27
3.1	Zadanie	27

3.2	Wstępne przygotowania	27
3.2.1	Przygotowanie komputera	27
3.2.2	Narzędzie do obsługi certyfikatów	29
3.2.3	Tworzenie certyfikatu i konfiguracja XAMPP	34
3.2.4	Środowisko programistyczne i język programowania	39
3.3	Wybrany schemat wstrzykiwania - DLL Injection	40
3.4	Realizacja zadania	42
3.4.1	Przygotowanie biblioteki DLL	42
3.4.2	Wstrzyknięcie kodu	43
3.4.3	Prawa administratora	48
3.5	Wyniki	53

Wstęp

Dzisiejszy świat Internetu i mediów społecznościowych zmienia się w sposób bardzo dynamiczny. Według szacunków firmy We Are Social, liczba osób posiadających dostęp do sieci wynosi już około 4 miliardów.¹ Tak wielki przyrost w porównaniu do sytuacji sprzed kilkunastu lat, mógł zostać osiągnięty poprzez wzrost liczby urządzeń mobilnych oraz polepszenie infrastruktury zapewniającej komunikację z nawet najbardziej opuszczonymi miejscami na Ziemi. Nic więc dziwnego, że wraz ze wzrostem liczby użytkowników oraz stale utrzymującym się trendem traktującym świat jako globalną wioskę, powstają coraz to nowsze zagrożenia.

Zagrożenia wynikają z podatności. Wraz ze wzrostem całkowitej liczby użytkowników, rośnie także liczba tych nieświadomych, którzy są łatwym celem dla ludzi zajmujących się złośliwym oprogramowaniem (ang. *malware*). Zagrożenia nie wynikają jednak tylko z nieświadomości. W dzisiejszych czasach złośliwe oprogramowanie jest bowiem tworzone nie tylko przez hobbystów czy hackerów pracujących w swoim domowym zaciszu. Coraz częściej można spotkać się z zespołami doświadczonych programistów wykonujących zlecenia dla zorganizowanych grup przestępczych a nawet państw. Do szerokiego wachlarza celów tych zespołów można zaliczyć przede wszystkim kradzież poufnych informacji, wysyłanie SPAM'u, wykradanie danych osobowych (w tym loginów i haseł do danych stron internetowych), a także ataki typu DDoS (ang. *distributed denial of service*).

Ludzie tworzący złośliwe oprogramowanie prześcigają się w wymyślaniu nowych sposobów atakowania swoich ofiar. Nie dziwi więc fakt, że poza standardowymi wirusami czy koniami trojańskimi, codzienny użytkownik musi odpierać ataki

¹Simon Kemp, Digital in 2017: Global Overview

robaków, exploitów, rootkitów, keyloggerów, ransomwerów a nawet oprogramowania szpiegującego. Wymienione zagrożenia to niestety tylko kropla w morzu. Każdego dnia powstają nowe, na które jedyną ochroną może być instalacja i codzienna aktualizacja programów antywirusowych nabytych u zaufanych dostawców. Niestety działanie takiego oprogramowania nie rozwiązuje wszystkich problemów. Polityka tworzenia ochrony przed złośliwym oprogramowaniem opiera się bowiem na zasadzie, że najpierw występuje atak a dopiero kolejnym krokiem jest wytworzenie zdolnemu mu zapobiec antidotum. Wynika to z faktu, że nie jesteśmy w stanie przewidzieć jaki będzie kolejny krok hakerów dopóki oni sami nam tego nie zdradzą. Dlatego też, poza odpowiednim oprogramowaniem, społeczeństwo korzystające ze smartfonów, tabletów, laptopów i innych tego typu urządzeń powinno przede wszystkim zachować stałą czujność i świadomość traktując o tym, żeby wykorzystywać tylko zaufane strony internetowe, nie podawać danych logowania etc. nieznanym (podejrzany) witrynom oraz nie ulegać atakom phishingowym wykorzystującym tak zwaną inżynierię społeczną.

Najlepszą metodą na ochronę przed malware'm jest dobra znajomość technik użytych do jego tworzenia. Biorąc pod uwagę przytoczone aspekty, celem niniejszej pracy jest zapoznanie się z jednym z przykładów złośliwego oprogramowania. Za pomocą implementacji wykorzystującej technikę DLL injection do ataku na zbiór certyfikatów systemu Windows 7 pokazane zostanie przykładowe działanie złośliwego wstrzyknięcia oraz wskazane zostaną zmiany do jakich doszło po jego zakończeniu. W niniejszej pracy przyjęty został następujący układ:

- Rozdział 1 - *Opis algorytmu wstrzykiwania kodu* przyjął formę przeglądu właściwości, jakimi kieruje się programista złośliwego oprogramowania na etapie jego projektowania. W tym rozdziale znajdują się ogólne wskazówki, które podpowiedzą jakimi zasadami się kierować, tak aby wytworzony kod był najbardziej wartościowy;
- Rozdział 2 - *Charakterystyka i sposoby realizacji poszczególnych etapów algorytmu* wskazuje na metody dzięki którym możemy efektywnie realizować kolejne kroki wstrzykiwania kodu. Rozdział ten daje konkretne narzędzia, które należy wykorzystać, aby otrzymać oczekiwane rezultaty;

- Rozdział 3 - *Opis przykładowej implementacji* dotyczy realizacji konkretnego zadania postawionego przed programistą oraz pozwala na prześledzenie krok po kroku jego realizacji. W tym rozdziale wskazane jest środowisko pracy oraz punkt startu niezbędny do poprawnego przeprowadzenia wstrzyknięcia za pomocą *Dll injection*. Pokazane są także wyniki działania programu.

Rozdział 1

Opis algorytmu wstrzykiwania kodu

Istnieje wiele technik wstrzykiwania kodu. Możemy je rozróżnić ze względu na wiele czynników. Jednym z kryteriów wybrania konkretnej techniki może być na przykład wybór celu. Z innej zaś strony będziemy się interesowali metodą, za pomocą której jesteśmy w stanie wykonać przekierowanie do wstrzykniętego kodu. Niezależnie od naszych intencji oraz środowiska, w jakim będziemy chcieli używać *code injection*, przedstawione w tej publikacji techniki wstrzykiwania kodu posiadają pewne cechy wspólne. Przede wszystkim ich celem jest umieszczenie kodu w zdalnym procesie, a następnie wykonanie go.

1.1 Wybór celu

Najważniejszym zagadnieniem, które należy poruszyć na etapie projektowania złośliwego oprogramowania jest wybór swojej *ofiary*. Jest to zagadnienie istotne z tego względu, że znając swojego przeciwnika, twórca malware'u może dokonać początkowej analizy korzyści płynących z zaatakowania konkretnego programu. Nie tylko powinny być rozważane korzyści, ale również potencjalne zagrożenia jakie niesie ze sobą zainfekowanie danego programu. Dla przykładu, jeśli będziemy starali się wstrzyknąć złośliwe oprogramowanie, które generuje jakikolwiek ruch sieciowy do kalkulatora systemowego, to nawet nieświadomy użytkownik będzie wiedział, że w jego komputerze/systemie dzieją się rzeczy niepożądane. Wynika z tego wprost, że to

atakujący występuje w roli osoby odpowiedzialnej za swoje ewentualne powodzenie lub jego brak.

1.2 Określenie potrzeb

Drugim wyzwaniem stojącym przed programistą malware'u jest określenie własnych potrzeb. W rozumieniu programu, oznacza to odpowiedź na takie pytania jak:

- Co ma robić szkodliwe oprogramowanie?
- Jakie uprawnienia powinien mieć przypisany kod, aby został pomyślnie wykonany?
- W jaki sposób dostarczyć kod?
- Co zrobić aby zostać niewykrytym?

Odpowiedzi do zadanych powyżej pytań jest wiele. Niektórzy tworzą szkodliwe oprogramowanie aby dodać funkcjonalności do pewnych programów, głównie gier komputerowych. Dzięki temu mają większy wpływ na sterowanie, a nawet są w stanie zmieniać wartości rejestrów z których korzysta gra, przechowujących np. informacje o liczbie amunicji. Niestety nie jest to pożądane działanie, ponieważ tacy gracze później oszukują podczas chociażby wieloosobowych rozgrywek przez Internet. Malware jest w stanie szkodzić nie tylko na poziomie programów niewymagających praw administratora. W niniejszej pracy wskazany zostanie efekt oddziaływania i modyfikowania kodu na znacznie wyższym poziomie. *Code injection* nie jest wykorzystywany tylko do szkodliwych celów. Z przedstawionych technik korzystają również oprogramowania antywirusowe, które wstrzykując się w dane procesy są w stanie na bieżąco monitorować i skanować dyski twarde, pocztę elektroniczną etc.

Nie ulega wątpliwości, że do poprawnego działania *loadera* potrzebne są odpowiednie uprawnienia. Jeżeli funkcjonalność *payload'u* będzie polegała na możliwości skopiowania jednego folderu do drugiego, to żadne prawa administratora nie są potrzebne. Niestety sytuacja zmienia się wraz z potrzebami *ładunku*

wstrzykiwanego. Jeśli chcemy, aby ładunek skopiował ten sam folder, ale do folderu systemowego, wtedy pojawi się komunikat o potrzebie praw administratora do wykonania konkretnego zadania. Wzbudzi to niepokój osoby użytkującej dany komputer, który był atakowany, a w konsekwencji może doprowadzić do zdemaskowania nieuprawnionego działania. Jak widzimy, programista powinien już wcześniej przewidzieć wszystkie możliwe problemy, jakie może napotkać jego oprogramowanie podczas atakowania poszczególnych systemów/podatności/sprzętu.

Kolejną kwestią, z jaką trzeba się zmierzyć jest dostarczenie *malware'u*. Można na przykład utworzyć samowystartujący się plik *.exe* na pamięci przenośnej. Wtedy po podłączeniu popularnego pendrive'a oprogramowanie samo się wykona. Można rozsyłać do użytkowników poczty pliki zarażone, posiadające znane rozszerzenia. Ukrywane jest prawdziwe rozszerzenie pliku (będące np. rozszerzeniem *.exe*) a na końcu nazwy pliku dopisane zostaje po prostu *.pdf* bądź *.jpg*. Wtedy mniej doświadczony użytkownik pobiera plik i próbuje go otworzyć. Oczywiście otwierany przez niego plik nie ma nic wspólnego z na przykład obrazkiem i dochodzi do zarażenia komputera.

Pozostaje jeszcze kwestia bycia niewykrytym. Dzisiejsze programy antywirusowe oraz zapora sieciowa (dla systemów z rodziny *Windows*) jest coraz bardziej odporna na działania szkodliwego oprogramowania. Ponadto, usługi pocztowe wprowadzają dodatkowe funkcjonalności, takie jak np. analiza wysyłanych i odbieranych maili bądź załączników, przez co występuje coraz mniejsza możliwość rozprzestrzeniania *payload'u* tymi właśnie sposobami. Skanery plików online oraz inne narzędzia skutecznie utrudniają pracę hakerom, crackerom czy potocznie mówiąc dystrybutorom niechcianego oprogramowania. Program antywirusowy bada plik wykonywalny, wyszukując w nim ustalonych schematów. Analizując je, jest w stanie powiedzieć, jakie funkcje z danych bibliotek były używane podczas pisania programu. Istnieje zbiór funkcji, które są traktowane jako szczególnie niebezpieczne. Do ich zbioru należą wszelkiego rodzaju funkcje próbujące modyfikować pamięć, posiadające dostęp do konkretnych rejestrów czy mające na celu utworzenie procesu z zadanymi parametrami. System czy program antywirusowy wychytuje sytuacje, które nie powinny się dziać podczas standardowego użytkowania komputera. Wykrywając jakiegokolwiek nieprawidłowości, usuwa dany plik, przenosi go do kwarantanny, bądź na bieżąco

informuje użytkownika o wynikach, pozwalając mu samodzielnie dokonać oceny sytuacji i wynikających z niej zagrożeń.

Istnieje wiele technik pozwalających na radzenia sobie z zabezpieczeniami wynikającymi z oprogramowania antywirusowego a także ochraniających *malware* od analizy przez ludzi zajmującymi się badaniem złośliwego oprogramowania. Do najczęściej używanych można zaliczyć:

- Zaciemnianie kodu - polegające na umieszczeniu pomiędzy instrukcje kodu, który w żaden sposób nie wpływa na wynik działania programu. Te dodane fragmenty określa się potocznie mianem śmieciowego kodu czyli ***Junk Code***. Oprócz wstawiania ich, w opisanej technice stosuje się szyfrowanie napisów używanych przez aplikację.
- Antyodpluskwanie - polegające na dodaniu do programu kodu, który jest odpowiedzialny za ochronę przed debugowaniem (czyli używaniem tzw. odpluskwiaczy). Skutkuje to brakiem dostępu do pliku wykonywalnego w czystej postaci, przez co utrudnia analizę kodu. Jednym z najłatwiejszych sposobów wychwycenia odpluskwiaczy jest użycie funkcji podsystemu *Win32* o nazwie:

IsDebuggerPresent

Funkcja ta zwraca wartość TRUE gdy aplikacja jest debugowana, w przeciwnym wypadku FALSE.

- Antydisasemblacja - zabieg mający na celu zapobiegnięcie disasemblacji, czyli odtworzeniu pliku wykonywalnego do listingu języka Asembler. Polega na użyciu instrukcji skoku z taką samą etykietą oraz wykorzystaniu *bajtu zabójcy* - czyli fragmentu kodu w języku assembler z którym nie poradzi sobie disasembler (najczęściej przyjmuje postać nieistniejącej funkcji tego języka).
- Antyemulacja - często przy analizie złośliwego oprogramowania korzysta się z środowiska virtualnego, które nie pozwala zainfekować prawdziwego systemu. W związku z tym, ważnym jest by program kończył swoje działanie po wykryciu

takiego środowiska. Osoba zajmująca się wstrzykiwaniem kodu powinna zatem wziąć pod uwagę poniższe czynniki:

1. Wykrywanie **VMware** - oprogramowania służącego do wirtualizacji.
2. Wykrywanie **Sandbox'ów** - izolowanych obszarów, w których uruchamiane są szkodliwe pliki bez ryzyka.
3. Wykrywania serwisów, które przeprowadzają analizę nieznanych plików binarnych (takich jak **Anubis**).²

1.3 Algorytm

Wszystkie techniki wstrzykiwania kodu korzystają z ogólnie przyjętego schematu:

1. Znalezienie procesu, który już istnieje, bądź utworzenie nowego procesu w stanie wstrzymanym.
2. Pobranie uchwytu procesu, który znaleźliśmy.
3. Alokacja pamięci, potrzebnej do zapisania naszego kodu. Jest ona wykonywana z odpowiednimi flagami, ponieważ interesuje nas możliwość odczytu i zapisu nowego fragmentu pamięci, przy zachowaniu jej adresu.
4. Wpisanie własnego kodu do wcześniej zarezerwowanego miejsca (przestrzeni adresowej).
5. Przekierowanie wykonania do wstrzykniętego kodu.
6. Wybudzenie wątku posiadającego wstrzyknięty kod.³

²na podstawie Dawid Farbaniec str. 90-93.

³na podstawie Redakcja naukowa Mateusz Jurczyk i Gynvael Coldwind, str. 228.

Rozdział 2

Charakterystyka i sposoby realizacji poszczególnych etapów algorytmu

W niniejszej części postaramy się dokładniej przyjrzeć konkretnym rozwiązaniom, które są odpowiedzialne za pomyślne wykonanie kolejnych kroków algorytmu przedstawionego w poprzednim rozdziale. Wszystkie funkcje systemowe, zaprezentowane przykłady rozwiązań implementacyjnych czy ogólnie schematy - działają dla 32-bitowych wersji systemu *Windows*.

Tytułowe metody wstrzykiwania kodu to tak naprawdę różne drogi wykonania algorytmu. Jeżeli przyjmiemy, że np program wstrzykujący może wybrać swój cel na dwa różne sposoby, natomiast przekierować wykonanie programu do wstrzykniętego kodu na trzy sposoby, to otrzymujemy 6 różnych technik.

2.1 Tworzenie wstrzykiwalnego kodu

Zanim jednak przejdziemy do opisu poszczególnych sposobów, jakie użytkownik może użyć w implementacji, postaramy się odpowiedzieć na pytanie, co w ogóle jesteśmy w stanie wstrzyknąć. Przede wszystkim musimy pamiętać, że aplikacja czy proces, do którego będziemy chcieli się dostać nie powinny stracić swojej ciągłości. Przez ciągłość rozumiemy tutaj zabezpieczenie przed sytuacją, w której proces wykonując się, natrafia na adres, którego nie jest w stanie zrozumieć. Następuje wtedy

błąd krytyczny, w konsekwencji którego dochodzi do zakończenia procesu. Skąd się biorą takie problemy?

Głównym powodem jest to, że możemy nie zapanować nad adresami bezwzględными. Wynika to z tego, że adres, pod który chcemy wpisać nowy kod, w większości przypadków jest losowy. Jeżeli nie będzie szansy na załadowanie kodu pod oryginalnie przewidziany do tego adres bazowy, wszystkie adresy bezwzględne stracą rację bytu.

Drugim aspektem jest importowanie funkcji. Niejednokrotnie spotkamy się z wywołaniem wybranej funkcji z biblioteki zewnętrznej. Musimy zadbać o to, aby dana biblioteka była wcześniej załadowana do pamięci pod adresem, który znamy.

Przedstawione problemy można rozwiązać na dwa sposoby:

1. **zewnętrzny** - aplikacja wstrzykująca po wstrzyknięciu kodu dokonuje relokacji adresów i ustala wszelkie zależności *ładunku*.
2. **wewnętrzny** - ładunek posiadający cechy *shellcodu* jest w stanie sam załadować potrzebne elementy.⁴

2.1.1 Shellcode

Shellcode to nieskomplikowany program prezentowany najczęściej w postaci kodu maszynowego odpowiedzialny za wywołanie powłoki systemowej (ang. *shell*). Jest on zwykle ograniczany limitami wielkościami, takimi jak rozmiar bufora wysyłanego do atakowanej aplikacji. Najczęściej używa się go, aby wykonać określone zadanie tak wydajnie jak to tylko możliwe w kontekście wykorzystania błędów zabezpieczeń przez *exploity*.

Pisanie shellcodu wymaga głębokiego zrozumienia języka assembler dla architektury, w której działa dana aplikacja, którą chcemy zaatakować. Istnieje wiele odmian języka assembler. Ze względu na popularność, do najczęściej używanych należą x86 oraz x64. Każda odmiana ma swoje unikalne cechy, może różnić się konwencjami składniowymi, symboliką mnemoników, a nawet sposobem interpretacji argumentów. Dlatego tak ważna jest znajomość *ofiary* złośliwego oprogramowania, ponieważ bez dobrze napisanego shellcodu nie jesteśmy w stanie nic zrobić nawet jeśli pomyślnie uda

⁴na podstawie Redakcja naukowa Mateusz Jurczyk i Gynvael Coldwind, str. 242.

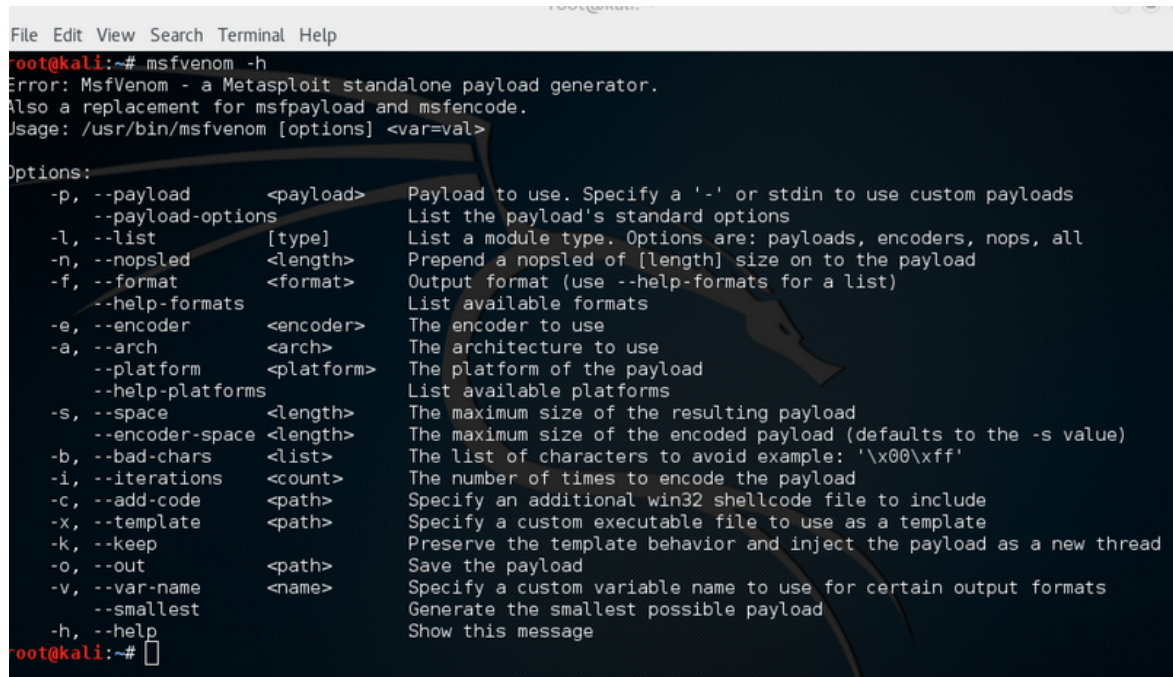
nam się włamać do innego programu.

Rodzaje *shellcodu*:

- **local** - używany przez atakujących, posiadających ograniczony dostęp do maszyny, ale mogących wykorzystywać podatności takie jak np. przeładowanie bufora w procesie o wyższych uprawnieniach.
- **remote** - używany podczas ataku na podatny proces działający w obrębie sieci lokalnej, intranetu.
- **download & execute** - rodzaj shellcodu typu remote polegający na ściągnięciu i wykonaniu jakiegoś malware'u.
- **staged** - podzielony na etapy. Najpierw wykonuje się jedną część, następnie inną. Używany dla ograniczonych przepływności danych.
- **egg-hunt** - inna forma typu staged. Używana, jeśli atakujący może umieścić większy kawałek shellcodu w procesie, ale nie jest w stanie powiedzieć gdzie on się skończy. W celu uniknięcia danej sytuacji, najpierw wgrywa się tzw. small egg-hunt, który przeszukuje przestrzeń adresową procesu, a następnie podpina większy shellcode i wykonuje go.
- **omelette** - działa podobnie do egg-hunt, z tą jednak różnicą że nie szuka całego wolnego bloku pamięci, aby pomieścić właściwy kod, ale składa go z małych fragmentów dostępnych po drodze przeszukiwania.⁵

Shellcode ma do wykonania często jedno z góry określone zadanie, np. pobrać i uruchomić plik. Może być wstępnym etapem w wykonaniu się poważniejszego malware'u. Nieważne jaką funkcję spełnia, zawsze warto znać zasady jego tworzenia. Generowanie shellcodu może być także automatyczne. Bardzo pomocne może się do tego okazać narzędzie *msvenom* z platformy *Metasploit*. W sieci można znaleźć także wiele przykładów gotowych kodów.

⁵na podstawie Eran Goldstein, str. 30-33

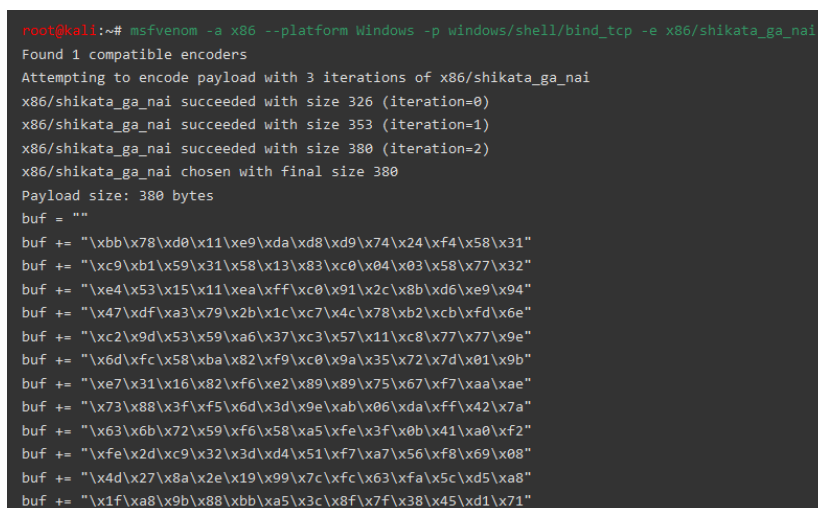


```
File Edit View Search Terminal Help
root@kali:~# msfvenom -h
Error: MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>

Options:
  -p, --payload <payload>      Payload to use. Specify a '-' or stdin to use custom payloads
  --payload-options             List the payload's standard options
  -l, --list <type>            List a module type. Options are: payloads, encoders, nops, all
  -n, --nopsled <length>       Prepend a nopsled of [length] size on to the payload
  -f, --format <format>        Output format (use --help-formats for a list)
  --help-formats                List available formats
  -e, --encoder <encoder>       The encoder to use
  -a, --arch <arch>             The architecture to use
  --platform <platform>        The platform of the payload
  --help-platforms             List available platforms
  -s, --space <length>         The maximum size of the resulting payload
  --encoder-space <length>     The maximum size of the encoded payload (defaults to the -s value)
  -b, --bad-chars <list>       The list of characters to avoid example: '\x00\xff'
  -i, --iterations <count>     The number of times to encode the payload
  -c, --add-code <path>        Specify an additional win32 shellcode file to include
  -x, --template <path>        Specify a custom executable file to use as a template
  -k, --keep                    Preserve the template behavior and inject the payload as a new thread
  -o, --out <path>             Save the payload
  -v, --var-name <name>        Specify a custom variable name to use for certain output formats
  --smallest                    Generate the smallest possible payload
  -h, --help                    Show this message

root@kali:~#
```

Rysunek 1: Narzędzie msfvenom z platformy Metasploit dostępnej dla systemu Kali Linux. Źródło [5], stan na dzień 05.12.2017



```
root@kali:~# msfvenom -a x86 --platform windows -p windows/shell/bind_tcp -e x86/shikata_ga_nai
Found 1 compatible encoders
Attempting to encode payload with 3 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 326 (iteration=0)
x86/shikata_ga_nai succeeded with size 353 (iteration=1)
x86/shikata_ga_nai succeeded with size 380 (iteration=2)
x86/shikata_ga_nai chosen with final size 380
Payload size: 380 bytes
buf = ""
buf += "\xbb\x78\xd0\x11\xe9\xda\xd8\xd9\x74\x24\xf4\x58\x31"
buf += "\xc9\xb1\x59\x31\x58\x13\x83\xc0\x04\x03\x58\x77\x32"
buf += "\xe4\x53\x15\x11\xea\xff\xc0\x91\x2c\x8b\xd6\xe9\x94"
buf += "\x47\xdf\xa3\x79\x2b\x1c\xc7\x4c\x78\xb2\xcb\xfd\x6e"
buf += "\xc2\x9d\x53\x59\xa6\x37\xc3\x57\x11\xc8\x77\x77\x9e"
buf += "\x6d\xfc\x58\xba\x82\xf9\xc0\x9a\x35\x72\x7d\x01\x9b"
buf += "\xe7\x31\x16\x82\xf6\xe2\x89\x89\x75\x67\xf7\xaa\xae"
buf += "\x73\x88\x3f\xf5\x6d\x3d\x9e\xab\x06\xda\xff\x42\x7a"
buf += "\x63\x6b\x72\x59\xf6\x58\xa5\xfe\x3f\x0b\x41\xa0\xf2"
buf += "\xfe\x2d\xc9\x32\x3d\xd4\x51\xf7\xa7\x56\xf8\x69\x08"
buf += "\x4d\x27\x8a\x2e\x19\x99\x7c\xfc\x63\xfa\x5c\xd5\xa8"
buf += "\x1f\xa8\x9b\x88\xbb\xa5\x3c\x8f\x7f\x38\x45\xd1\x71"
```

Rysunek 2: Przykładowy wynik działania narzędzia. Źródło [5] Stan na dzień 29.11.2017

Rysunek 2 przedstawia shellcode jaki uzyskujemy za pomocą narzędzia msfvenom po użyciu komendy

```
msfvenom -a x86 --platform Windows -p windows/shell/bind_tcp  
-e x86/shikata_ga_nai -b '\x00' -i 3 -f python
```

Otrzymujemy tzw. Windows Bind Shell w formacie python, czyli kod, który poprzez wykonanie się utworzy port akceptujący zdalną sesję z atakującym (za pośrednictwem Metasploit).

2.1.2 Plik PE

PE (ang. Portable Executable) jest to format plików, do którego należą pliki wykonywalne, DLL - biblioteki dynamiczne, a także pliki obiektowe. Jest to struktura danych przechowująca wszystkie informacje niezbędne systemowi operacyjnemu do zarządzania kodem programu. Do informacji tych należą:

- informacje o uruchamianych wątkach;
- dane o zasobach programu;
- tablice eksportowanych i importowanych funkcji Windows API;
- odnośniki do bibliotek DLL.

Wstrzykiwanie plików PE pozwala na osiągnięcie znacznie większej funkcjonalności niż użycie samego shellcodu. W znaczącej ilości przypadków, złośliwe oprogramowanie w całości jest kopiowane do pamięci innych programów, co umożliwia jego wykonywanie się bez ujawniania (jest ono niewidoczne na liście uruchomionych procesów). Pliki PE podczas uruchamiania zostają automatycznie mapowane do pamięci a całą ich obsługą zajmuje się loader systemu Windows. Podczas wstrzykiwania niestety loader nie działa, dlatego wszystkie czynności związane z mapowaniem trzeba zaimplementować samodzielnie.⁶

⁶na podstawie Redakcja naukowa Mateusz Jurczyk i Gynvael Coldwind str. 244-245.

2.2 Wybór celu

Kod możemy wstrzykiwać do już działających programów, oraz tych, które dopiero uruchamiamy samodzielnie. W praktyce sprowadza się to do dwóch opcji. W pierwszej z nich monitorujemy wszystkie procesy użytkownika i dopiero po stwierdzeniu, że program, który chcemy zarazić, jest wykonywany, atakujemy go. W drugiej natomiast, za pomocą funkcji systemowych, powołujemy do życia nowy proces.

2.2.1 Wstrzykiwanie kodu do istniejącego procesu

Podczas dodawania wątku do istniejącego procesu, trzeba zachować ostrożność, aby nie naruszyć stabilności programu. Oznacza to, że program, w który się wstrzykujemy nie powinien nagle się zawiesić etc. ale powinien pracować dalej tak jakby nic się nie stało. Wynika z tego, że najrozsądniejszym sposobem przekierowania wykonania do wstrzykniętego kodu będzie uruchomienie go w nowym wątku. W tym celu należy na samym początku znaleźć proces, którym jesteśmy zainteresowani. Do przeszukiwania uruchomionych procesów można użyć dwóch alternatywnych sposobów udostępnionych przez **WinAPI**. Pierwszy z nich to funkcja

```
BOOL WINAPI EnumProcesses(  
    _Out_ DWORD *pProcessIds,  
    _In_   DWORD cb,  
    _Out_ DWORD *pBytesReturned  
);
```

Należy ona do biblioteki *Psapi.lib*. Funkcja zwraca tylko zmienną typu logicznego. Aby jej użyć na początku należy stworzyć odpowiednio dużą tablicę zmiennych typu DWORD (ponieważ nie wiemy ile procesów mamy uruchomionych). Po pomyślnym wykonaniu się tej funkcji, tablica zostanie uzupełniona listą identyfikatorów procesów, natomiast zmienna pBytesReturned wskaże liczbę bajtów zwróconą w tablicy pProcessIds. W celu obliczenia ilości wykonywanych procesów należy wykonać podzielenie pBytesReturned przez sizeof(DWORD). Jeżeli natomiast będziemy chcieli otrzymać nazwy i numery PID procesów, powinniśmy na każdym procesie

z tablicy wykonać najpierw udokumentowaną funkcję `EnumProcessModulesEx()` (z odpowiednimi parametrami), a później w celu wydobycia nazwy kolejną `GetModuleBaseName()`.

Kolejnym sposobem znalezienia ustalonego procesu jest kombinacja funkcji:

`CreateToolhelp32Snapshot`, `Process32First` i `Process32Next`

Funkcja `CreateToolhelp32Snapshot` umożliwia wylistowanie procesów i należących do nich wątków, modułów oraz pamięci. Jeśli wywołanie się powiedzie, funkcja zwróci uchwyt do listy z danymi. Po skończeniu operacji związanych z tym uchwyt, należy go zamknąć funkcją `CloseHandle`. W przypadku gdy wywołanie się nie powiedzie, funkcja zwraca wartość `INVALID_HANDLE_VALUE`. Dodatkowe informacje o błędzie można pobrać za pomocą funkcji `GetLastError`. Do późniejszego poruszania się po tej liście służą `Process32First` i `Process32Next`.

2.2.2 Wstrzykiwanie kodu do nowo uruchomionego procesu

W celu uruchomienia nowego procesu, wykorzystujemy funkcję `CreateProcess()`. Musimy jednak pamiętać, aby w argumencie określającym flagi kontrolujące klasę priorytetu i tworzenie się procesu, wpisać `CREATE_SUSPENDED`. Spowoduje to utworzenie nowego procesu w stanie wstrzymanym. Oznacza to, że zostanie on zmapowany do pamięci, ale jego wykonanie się nie rozpocznie. Analogicznie można użyć funkcji `CreateProcessInternal()`.

Sposób wykorzystania omawianej metody przedstawia:

```
CreateProcess(NULL, "svchost.exe", NULL, NULL, FALSE,  
              CREATE_SUSPENDED, NULL, NULL, &si, &pi);
```

Podany fragment kodu uruchamia proces systemowy `svchost.exe` w stanie wstrzymanym. Należy zwrócić uwagę na fakt, iż wszelkie dane odnośnie nowego procesu przechowywane są w dwóch oddzielnych strukturach. Pierwszą z nich jest `STARTUPINFO`, dla której mamy odniesienie `&si`. Jest ona głównie odpowiedzialna za przechowywanie informacji o wyglądzie okna w czasie tworzenia procesu - jego rozmiarach, nazwie oraz czy w ogóle okno będzie wyświetlane. Drugą natomiast jest

PROCESS_INFORMATION, reprezentowana przez `&pi`, która przechowuje informacje na temat uchwytu do nowo utworzonego procesu, uchwytu do jego głównego wątku oraz jego ID.

2.3 Wpisywanie kodu do zdalnego procesu

Uruchomienie złośliwego kodu w zdalnym procesie na samym początku wymaga skopiowania go do przestrzeni adresowej tego procesu. Ponownie istnieją dwie możliwości na zrobienie miejsca na kod. Można alokować w zdalnym procesie fragment pamięci lub dodać nową sekcję.

2.3.1 Alokacja fragmentu pamięci

Bezpośrednie zapisywanie kodu do zdalnej sekcji możemy osiągnąć używając funkcji `WriteProcessMemory()` i podając tzw. handler (uchwyt) do procesu, do którego się podpinamy. Składnia wygląda następująco:

```
BOOL WINAPI WriteProcessMemory(  
    _In_   HANDLE   hProcess,  
    _In_   LPVOID   lpBaseAddress,  
    _In_   LPCVOID  lpBuffer,  
    _In_   SIZE_T   nSize,  
    _Out_  SIZE_T   *lpNumberOfBytesWritten  
);
```

Zauważmy, że funkcja zwraca zmienną typu `bool`, wskazującą czy udało się zapisać czy nie. Do odpowiedniego działania potrzebuje takich argumentów jak:

1. `hProcess` - uchwyt do pamięci procesu, która będzie modyfikowana. Uchwyt ten powinien posiadać dostęp do procesu rzędu `PROCESS_VM_WRITE` i `PROCESS_VM_OPERATION`.
2. Wskaźnik do podstawowego adresu w wybranym procesie, do którego zapisuje się dane. Przed wykonaniem transferu danych, system weryfikuje czy dana przestrzeń adresowa jest dostępna z prawami do zapisu.

3. Wskaźnik do bufora zawierającego dane.
4. Liczba bajtów do zapisu.
5. Wskaźnik do zmiennej przechowującej liczbę przekopiowanych danych do konkretnego procesu, wartość najczęściej ustawiana na NULL.⁷

Na uwagę zasługuje fakt, że najczęściej z funkcją `WriteProcessMemory()` używa się innej `VirtualAllocEx()`. Jej działanie polega na zarezerwowaniu albo modyfikowaniu obszaru pamięci umieszczonego w obrębie wirtualnej przestrzeni adresowej wyszczególnionego procesu.

2.3.2 Dodanie nowej sekcji

Alternatywą dla przedstawionej powyżej metody jest dodanie sekcji do wykonującego się procesu. Wykonuje się to za pomocą funkcji `ZwCreateSection()`. Następnym krokiem jaki należy wykonać jest przemapowanie tej sekcji w kontekst procesu zdalnego. Służy do tego funkcja `NtMapViewOfSection()`. Kolejne kroki dla omawianego sposobu wyglądają następująco:

1. Utworzenie sekcji w kontekście lokalnego procesu - należy pamiętać o udzieleniu jej odpowiednich praw dostępu - zazwyczaj realizuje się to za pomocą `SECTION_ALL_ACCESS`.
2. Mapowanie sekcji do pamięci lokalnego procesu.
3. Kopiowanie zawartości bufora do nowej sekcji.
4. Mapowanie tej samej sekcji w kontekst zdalnego procesu.
5. Odmapowanie sekcji z lokalnego procesu.⁸

⁷na podstawie MSDN Microsoft Documentation, `WriteProcessMemory` function.

⁸na podstawie Redakcja naukowa Mateusz Jurczyk i Gynvael Coldwind, str. 231-232.

2.4 Metody przekierowania do wstrzykniętego kodu

Istnieje bardzo wiele sposobów na *zmuszenie* procesu/aplikacji do wykonania wstrzykniętego kodu. Na potrzeby zadania implementacyjnego użyte zostało wstrzykiwanie biblioteki dynamicznej, dlatego w głównej mierze zaprezentowane w tej części techniki będą odnosiły się do tzw. *DLL injection*.

2.4.1 Uruchomienie kodu w nowym wątku

DLL injection za pomocą `CreateRemoteThread()` jest uważana za klasyczną i najbardziej popularną technikę wstrzykiwania DLL. Dzięki swojej prostocie i częstości używania, jest także najlepiej udokumentowaną. Polega ona na dodaniu wątku do zewnętrznej aplikacji. Dzięki zastosowaniu funkcji systemowej możemy utworzyć nowy wątek jako wstrzymany (odpowiednia flaga ustawiona na `CREATE_SUSPENDED`) lub do natychmiastowego uruchomienia. Parametrami `CreateRemoteThread()` są:

1. `hProcess` - uchwyt procesu, w którym chcemy utworzyć wątek.
2. `lpThreadAttributes` - wskaźnik do struktury `SECURITY_ATTRIBUTES` określający atrybuty bezpieczeństwa nowego wątku.
3. `dwStackSize` - rozmiar stosu w bajtach.
4. `lpStartAddress` - wskaźnik do funkcji, która zostanie wykonana po utworzeniu wątku.
5. `lpParameter` - wskaźnik do zmiennej, która będzie przekazana jako parametr funkcji wątku.
6. `dwCreationFlags` - wartość kontrolująca utworzenie wątku.
7. `lpThreadId` - wskaźnik do zmiennej otrzymującej ID wątku.

Wartością zwracaną jest uchwyt utworzonego wątku.⁹

⁹na podstawie MSDN Microsoft Documentation, `CreateRemoteThread` function.

2.4.2 Użycie bibliotek nieudokumentowanych

Kolejną techniką jaką można przyjąć przy realizacji zagadnienia wstrzykiwania kodu jest użycie *NtCreateThreadEx()*. Funkcja ta, może być zaimportowana z biblioteki *ntdll.dll*. Problemem, jaki może wystąpić przy implementacji tej metody jest fakt, że funkcja ta może w każdym momencie przestać działać albo zmieniać się w przyszłości. Działanie jej jest podobne do *CreateRemoteThread()* z tą różnicą, że implementacja okazuje się być bardziej wymagająca, ponieważ potrzebujemy specjalnych struktur, za pomocą których możemy wysyłać do niej argumenty oraz specjalnych, aby odbierać. Kolejna nieudokumentowana funkcja z podanej biblioteki to *NtQueueApcThread()*. Deklaracja tej funkcji nie jest niestety dostępna w *ntddk.h*, dlatego należy napisać ją samemu. Każdy wątek posiada własną kolejkę APC (ang. *Asynchronous Procedure Calls*), czyli strukturę zawierającą kolejne adresy punktów wejścia kodu do wykonania. Manipulując daną kolejką można przekierować ją do wykonania naszego kodu. Warto w tym momencie nadmienić, że w przeciwieństwie do pierwszej funkcji, druga nie tworzy dodatkowego wątku, ale jest dodawana do istniejącego. Mimo że używanie nieudokumentowanych bibliotek jest niebezpieczne, ponieważ nie mamy pełnej kontroli nad wykonywanym kodem, to jednak wielokrotnie się z nich korzysta, ponieważ dzięki nim łatwiej można obejść zabezpieczenia czy programy antywirusowe.

2.4.3 QueueUserAPC()

Przekierowanie do wstrzykniętego kodu można również zrealizować za pomocą *QueueUserAPC()*. Technika ta także nie tworzy nowego wątku, ale operuje na podobnej zasadzie do ostatnio omówionej *NtQueueApcThread()*. Dodaje ona obiekt wywołania asynchronicznego procedury do kolejki APC dla konkretnego wątku. Posiada ona trzy argumenty:

1. *pfnAPC* - wskaźnik do funkcji APC wspomaganej przez aplikację, do wywołania w momencie gdy określony wątek będzie chciał wykonać *hThread*.
2. *hThread* - uchwyt do wątku. Musi on posiadać prawa dostępu typu *THREAD_SET_CONTEXT*.

3. dwData - osobna wartość przekazywana do funkcji APC wskazywanej przez pfnAPC.

Przy użyciu tej techniki występuje jednak jeden dodatkowy szczegół, który wiąże się bezpośrednio ze sposobem w jaki MS Windows wykonuje APC. Nie ma żadnego wglądu w kolejkę APC, co oznacza, że kolejka jest badana jedynie wtedy, gdy wątek przechodzi do *stanu alarmowego* (ang. *alertable state*).¹⁰

2.4.4 Nadpisanie punktu wejścia procesu

Podczas uruchamiania procesu, który jest wprowadzony w stan wstrzymania (odpowiednia flaga ustawiona na SUSPENDED), mamy pewność, że wykonanie jeszcze się nie rozpoczęło. Punkt wejścia programu nie został osiągnięty, dlatego istnieje możliwość nadpisania kodu wykonania programu własnym kodem. W związku z tym, po wznowieniu będzie możliwe jego automatyczna dalsza realizacja. Analogicznie, możliwym jest nadpisanie pliku na dysku. W tym celu należy dodać sekcję z kodem, który chcemy wykonać oraz wstawić w punkt wejścia trampolinę do tej sekcji. Ransomware Cerber jest jednym z najlepszych przykładów zastosowania tego typu wstrzyknięcia.

Etapy realizacji techniki:

- znalezienie jak najbardziej aktualnego punktu wejścia (ang. *point of entry*) pliku Portable Executive (.exe/dll/etc.) programu do którego chcemy wstrzyknąć kod. W wyniku działania na strukturze PROCESS_BASIC_INFORMATION oraz funkcji NtQueryInformationProcess() jesteśmy w stanie wyszukać adres PEB (ang. *Process Environment Block*) procesu wykonywanego zdalnie a w konsekwencji uzyskać adres bazowy modułu. Dzięki temu otrzymujemy miejsce, w którym plik PE startuje w pamięci;
- nadpisanie istniejącego kodu naszym. Możemy wpisać albo cały shellcode, albo nadpisać punkt wejścia trampoliną do dodanej sekcji;
- odmrożenie procesu będące końcowym etapem. Nasz kod zostaje wykonany.¹¹

¹⁰na podstawie MSDN Microsoft Documentation, QueueUserAPC function.

¹¹na podstawie Redakcja naukowa Mateusz Jurczyk i Gynvael Coldwind, str. 235-236.

Rozdział 3

Opis przykładowej implementacji

3.1 Zadanie

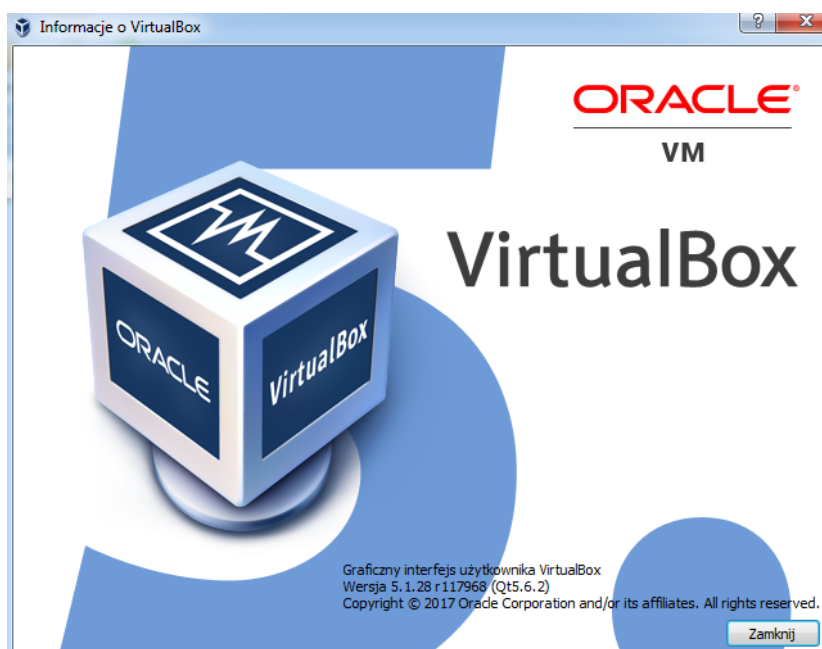
Zadanie, jakie zostało zrealizowane w ramach pracy dyplomowej, to wstrzyknięcie kodu do procesu systemowego. Działanie to polegało na wykonaniu metody *DLL injection* na wybranym procesie. Ofiarą w tym przypadku był 32-bitowy system Windows w wersji Professional. Złośliwy kod miał za zadanie zmodyfikować listę zaufanych certyfikatów. Fałszywy certyfikat miał być dodany do listy znajdującej się w rejestrze *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SystemCertificates\AuthRoot\Certificates*.

3.2 Wstępne przygotowania

3.2.1 Przygotowanie komputera

Analiza złośliwego oprogramowania to zajęcie szkodliwe dla systemu, na którym pracujemy. Niejednokrotnie podczas pisania własnego kodu, można zupełnie przez pomyłkę wyrządzić wiele strat. Sposobem uniknięcia potencjalnych zagrożeń, takich jak usunięcie plików systemowych, bądź format dysku, jest próba pisania pliku źródłowego a w konsekwencji uruchamianie pliku wykonywalnego na maszynie

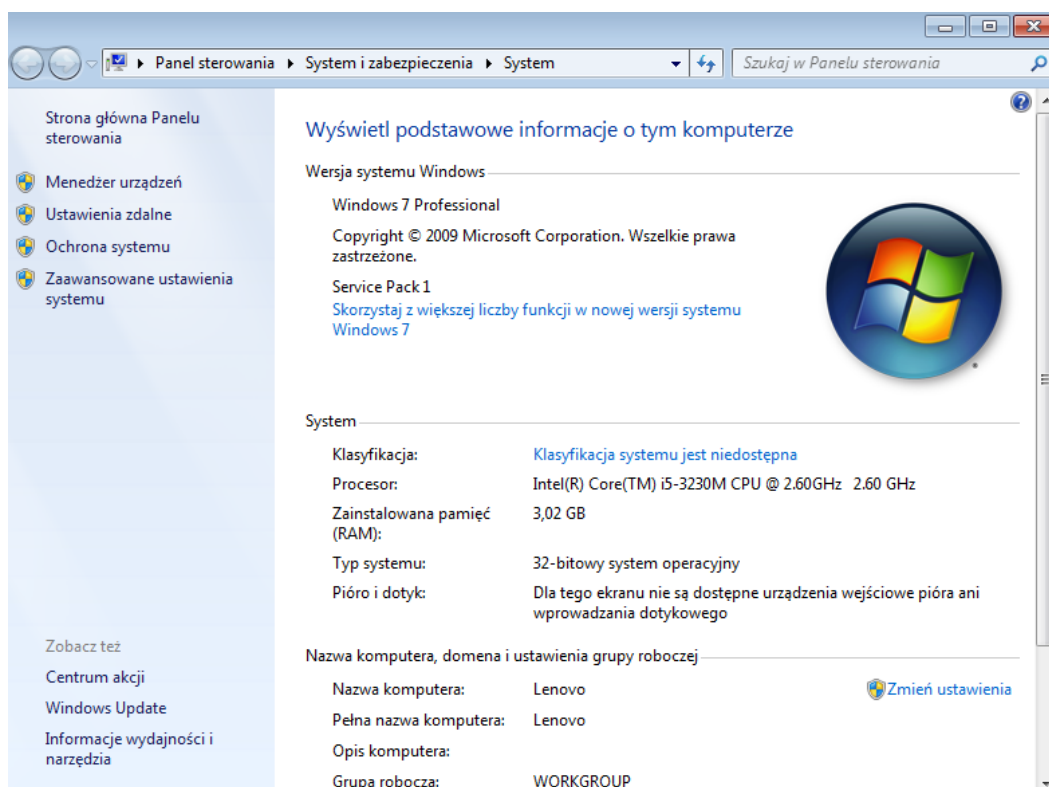
wirtualnej. Do obsługi wirtualnego systemu niezbędny jest menedżer maszyn wirtualnych. Podczas realizacji zadania posłużymy się popularnym narzędziem firmy ORACLE, a mianowicie *VirtualBox*'em w wersji 5.1.28.



Rysunek 3: Informacja o wersji użytego menedżera maszyn wirtualnych. Opracowanie własne, stan na dzień 04.12.2017

Na podanej maszynie wirtualnej, zainstalowany został 32-bitowy system operacyjny z rodziny **Windows** w wersji *Professional*. Został on wybrany w bardzo wielu względów:

1. Najbardziej popularny system wybierany przez użytkowników domowych.
2. Występuje mała świadomość na ataki wśród użytkowników - wynika to z faktu, że wszystkie grupy wiekowe i społeczne korzystają z tego oprogramowania.
3. Posiada bardzo wiele podatności, które są bardzo często wykorzystywane.
4. Pomimo wprowadzenia nowszych wersji, zakłady pracy takie jak np. urzędy, fabryki dalej z niego korzystają, ponieważ wymiana oprogramowania w takich miejscach wiąże się z wielomilionowymi inwestycjami w sprzęt, na które nie zawsze można sobie pozwolić.



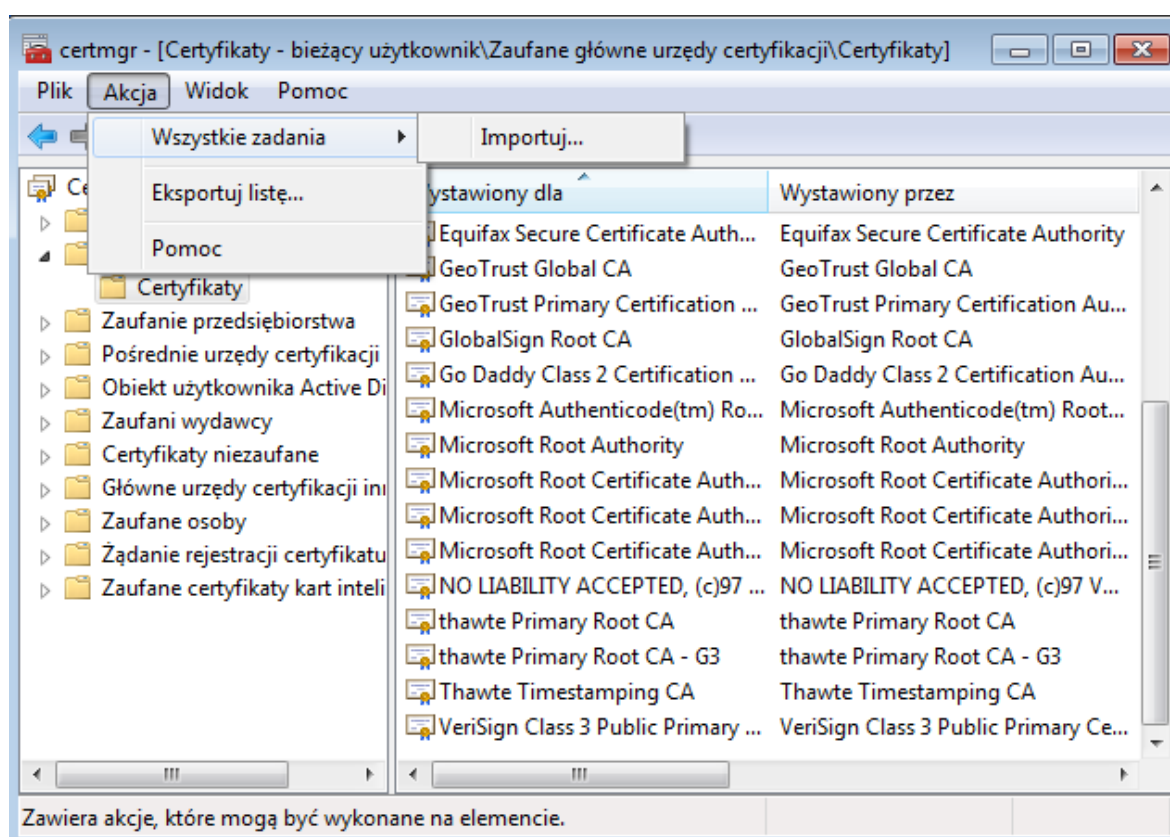
Rysunek 4: Informacja o wersji testowanego systemu Windows. Opracowanie własne, stan na dzień 04.12.2017

Zgodnie z Rysunkiem 4 widzimy, że zainstalowane zostało 3 GB pamięci RAM. Wynika to z tego względu, że w dzisiejszych czasach do poprawnego działania wymagane jest co najmniej tyle pamięci.

3.2.2 Narzędzie do obsługi certyfikatów

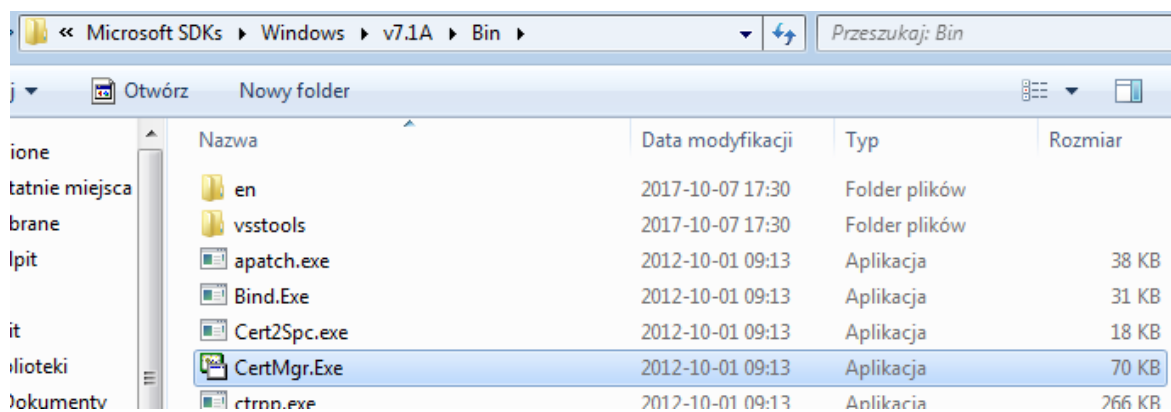
Ogólnym zamierzeniem zadania implementacyjnego jest uzyskanie pełnej kontroli nad ilością i rodzajem certyfikatów przechowywanych w systemie. Największym utrudnieniem, jakie stoi przed programistą jest fakt, że nie możemy bezpośrednio modyfikować certyfikatów poprzez ustawianie odpowiednich wartości ani nawet całych zmiennych w rejestrach. Zmian można jedynie dokonać poprzez odpowiednie korzystanie z narzędzi takich jak *Certmgr.exe* oraz *Certmgr.msc*. Zgodnie z definicją pochodzącą z dokumentacji Microsoft, narzędzie menedżer certyfikatów

(Certmgr.exe) zarządza certyfikatami, listami zaufania certyfikatów (CTL) oraz listami odwołania certyfikatów (CRL). Certmgr.exe jest narzędziem wiersza poleceń, natomiast narzędzie Certyfikaty (Certmgr.msc) przystawką programu Microsoft Management Console (MMC). Z praktycznego punktu widzenia, różnica pomiędzy tymi dwoma narzędziami wynika wyłącznie z ich obsługi. Podczas gdy Certmgr.msc jest wygodnym narzędziem dla mniej biegłych użytkowników dostępnym za pomocą aplikacji okienkowej, w której wszystkie czynności jesteśmy w stanie zrealizować poprzez tzw. kreatory, to Certmgr.exe działa na dwa sposoby. Może być aplikacją konsolową, w której dostęp otrzymujemy poprzez wpisanie odpowiednich komend, takich jak np. `certmgr [/add | /del | /put] [options]` w wierszu poleceń, bądź aplikacją okienkową.



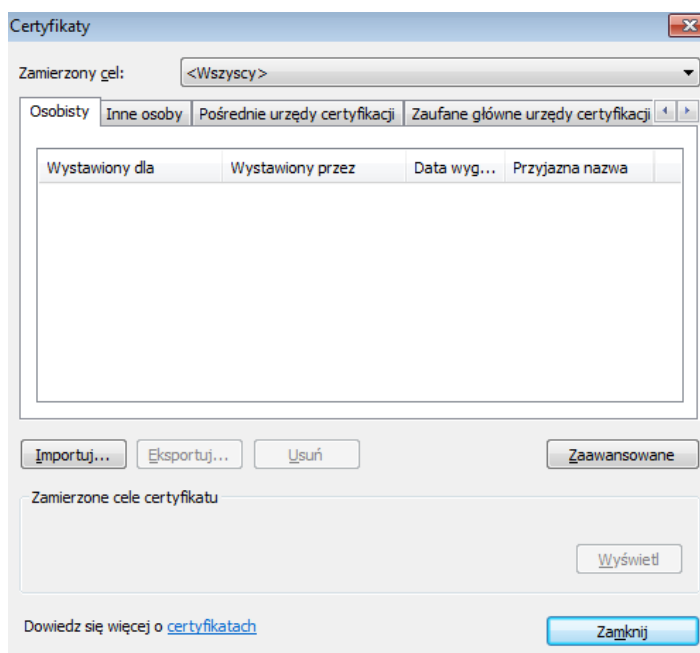
Rysunek 5: Narzędzie Certmgr.msc. Opracowanie własne, stan na dzień 04.12.2017

Rysunek 5 przedstawia jak łatwo jesteśmy w stanie z poziomu użytkownika realizować wszystkie czynności związane z certyfikatami. Na zdjęciu widoczna jest lista akcji. Użytkownik może importować zarówno nowe certyfikaty do listy jak i eksportować jej zawartość.



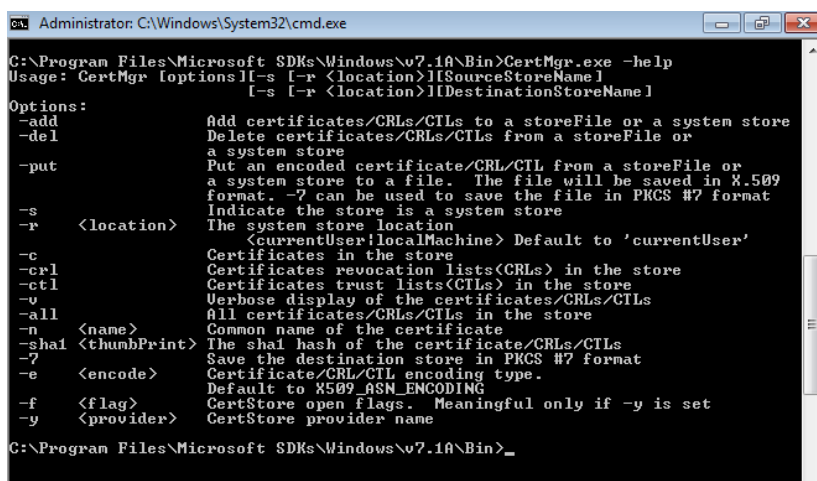
Rysunek 6: Lokalizacja Certmgr.exe. Opracowanie własne, stan na dzień 04.12.2017

Pierwszym utrudnieniem, jakie napotyka użytkownik przy korzystaniu z Certmgr.exe, jest jego znalezienie. W przeciwieństwie do przystawki programu Microsoft Management Console (.mmc), nie wystarczy wybranie w wierszu polecenia komendy CertMgr. Takie postępowanie zawsze doprowadzi do uruchomienia programu z rozszerzeniem .msc. Dzieje się tak dlatego, że Certmgr.msc znajduje się w katalogu systemu Windows, w związku z czym posiada pierwszeństwo przed ścieżką do narzędzia Certmgr.exe.



Rysunek 7: Certmgr.exe w wersji okienkowej. Opracowanie własne, stan na dzień 04.12.2017

Na Rysunku nr 7 zaprezentowano działanie CertMgr.exe. Interfejs, bardzo podobny do poprzedniego narzędzia, oferuje takie same funkcjonalności i jest bardzo intuicyjny.



Rysunek 8: Certmgr.exe w wersji konsolowej (wywołanie z *cmd*). Opracowanie własne, stan na dzień 04.12.2017

W przeciwieństwie do poprzednich rozwiązań, podane na Rysunku nr 8 wymaga od potencjalnego użytkownika, chociażby podstawowej znajomości składni komend, jakie można zastosować. Zgodnie z listingiem wyszczególnionym na tym rysunku widzimy, że do możliwości jakie oferuje CertMgr.exe należą:

- wyświetlanie na konsoli certyfikatów oraz list CTL i CRL;
- dodawanie certyfikatów oraz list CTL i CRL do magazynu certyfikatów;
- usuwanie certyfikatów oraz list CTL i CRL z magazynu certyfikatów;
- zapisywanie certyfikatów X.509, list CTL lub CRL z magazynu certyfikatów do pliku.

3.2.3 Tworzenie certyfikatu i konfiguracja XAMPP

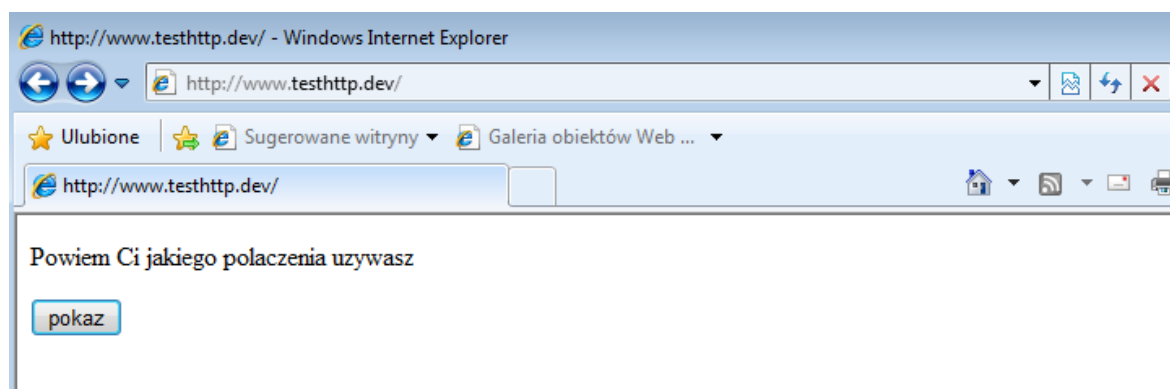
XAMPP to wieloplatformowy, zintegrowany, a przede wszystkim zupełnie darmowy pakiet, składający się głównie z serwera *Apache*, bazy danych *MySQL* oraz interpreterów dla skryptów napisanych w *PHP* i *Perl*. Wydawany jest na licencji GNU jako darmowy serwer WWW do obsługi dynamicznych stron.



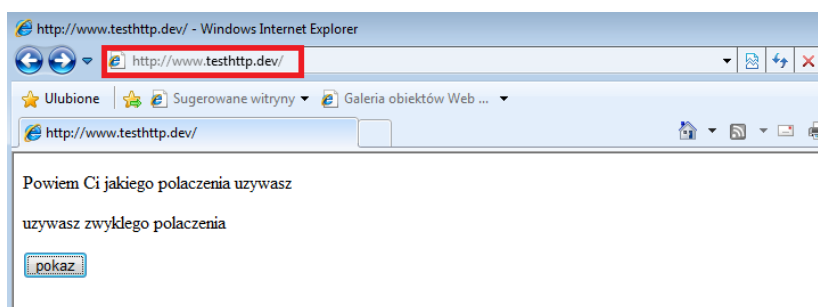
Rysunek 9: Logo XAMPP. Źródło [9], stan na dzień 15.12.2017

Tworzenie strony internetowej

Realizacja zadania implementacyjnego wymagała postawienia lokalnego serwera WWW w celu utworzenia strony internetowej, która badałaby działanie złośliwego oprogramowania przy modyfikowaniu certyfikatów. Budowa witryny opiera się na języku HTML i składni do obsługi przycisku tzw. *button*, która rozpoznaje rodzaj protokołu używanego do komunikacji.

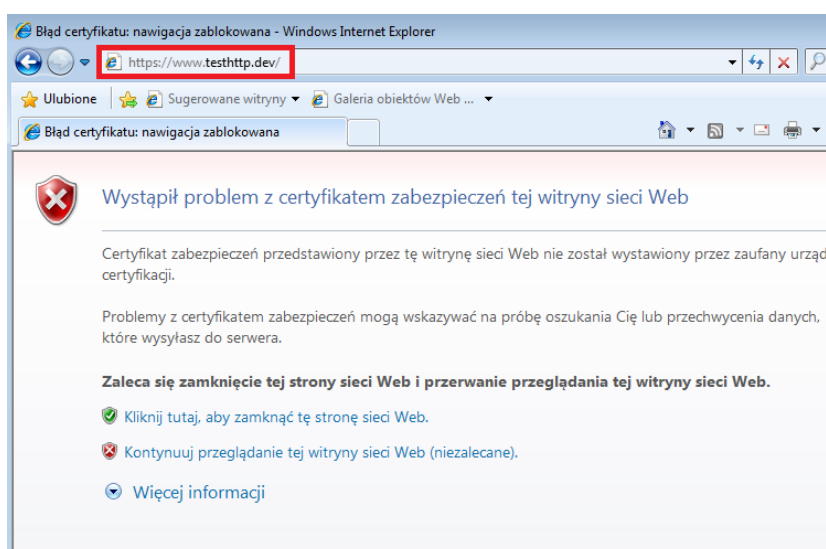


Rysunek 10: Strona internetowa używana do sprawdzenia działania złośliwego oprogramowania. Opracowanie własne, stan na dzień 04.12.2017



Rysunek 11: Rodzaj protokołu - nieszyfrowany *http*. Opracowanie własne, stan na dzień 04.12.2017

Protokół transferu hipertekstu HTTP (ang. *Hypertext Transfer Protocol*) jest protokołem interakcji między serwerem WWW a klientem internetowym. "Protokół HTTP definiuje reguły żądań i odpowiedzi. Gdy klient, zwykle przeglądarka internetowa, wysyła żądanie do serwera, protokół HTTP definiuje typy wiadomości używanych przez klienta do zażądania strony internetowej oraz typy wiadomości używanych przez serwer do udzielenia odpowiedzi.¹²" Jest najczęściej używany do przesyłania danych.



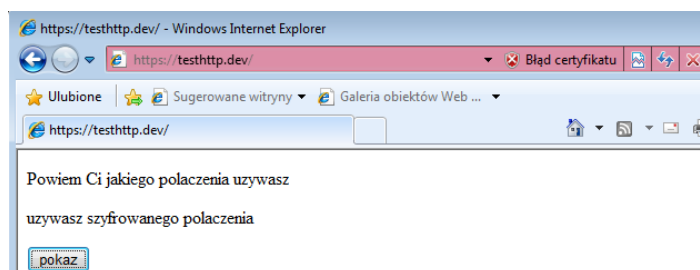
Rysunek 12: Rodzaj protokołu - szyfrowany *https*. Opracowanie własne, stan na dzień 04.12.2017

¹²Mark A. Dye, Rufi Antoon W., Rick McDonald, str. 121.

Protokół HTTP, nie jest bezpieczny. W wiadomościach informacje są przesyłane do serwera w postaci tekstu jawnego, który można odebrać i odczytać. Odpowiedzi serwera nie są szyfrowane. Do bezpiecznego komunikowania się w sieci używa się zatem protokół **HTTPS**. Protokół ten do bezpiecznego przesyłania danych używa uwierzytelniania i szyfrowania¹³. Szyfrowanie danych odbywa się przy pomocy protokołu SSL (dokładniej TLS ang. *Transport Layer Security* - rozwinięcie protokołu SSL) zapewniającego poufność i integralność transmisji danych oraz wykorzystującego klucze asymetryczne oraz certyfikaty X.509. Potwierdzeniem różnic występujących w obu protokołach jest wynik, który otrzymaliśmy od serwera po wysłaniu żądania poprzez naciśnięcie przycisku *pokaz*. Dla wykorzystywanego protokołu HTTP (Rysunek 11) wyświetlony został komunikat *używasz zwykłego połączenia*, podczas gdy dla protokołu HTTPS (Rysunek 12) otrzymaliśmy komunikat o błędzie certyfikacji. HTTPS do poprawnego działania potrzebuje aktualnego certyfikatu wystawionego przez właściwy (zaufany) organ certyfikacji (CA - ang. *Certification Authority*). Dla rozwiązania powstałego błędu można wykonać jedną z dwóch czynności:

- opuścić stronę WWW;
- kontynuować przeglądanie witryny sieci Web.

Dla drugiej czynności kontynuowane będzie działanie strony internetowej, niemniej jednak pasek adresu przez cały czas będzie wskazywał na próbę oszustwa lub przechwycenia danych (zgodnie z Rysunkiem nr 13).



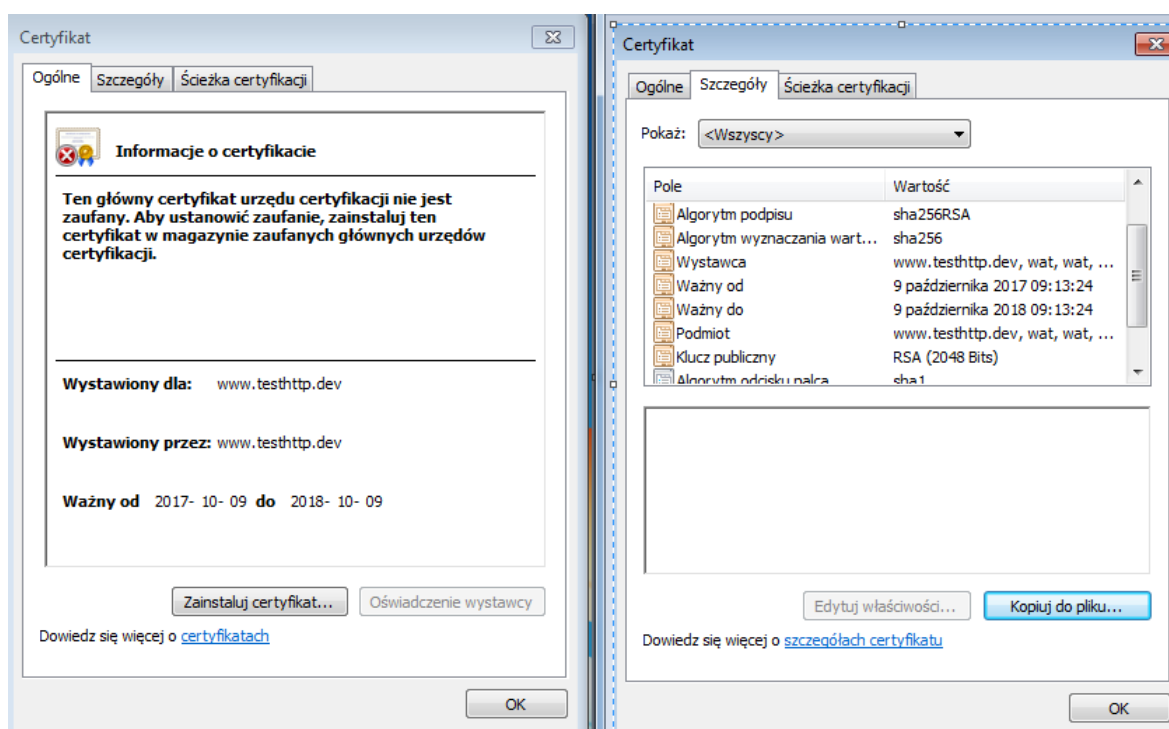
Rysunek 13: Strona po naciśnięciu przycisku kontynuacji. Opracowanie własne, stan na dzień 04.12.2017

¹³na podstawie Mark A. Dye, Rufi Antoon W., Rick McDonald, str. 122.

Celem tworzonego *malware'u* jest umieszczenie certyfikatu wystawionego dla strony `www.testhttp.dev` w magazynie certyfikatów systemu *Windows* bez wiedzy użytkownika za pośrednictwem `CertMgr.exe`.

Utworzenie certyfikatu

Po zainstalowaniu programu XAMPP, pierwszą rzeczą jaką należy wykonać w celu skonfigurowania `https` dla `localhosta` jest utworzenie certyfikatu dla strony internetowej. Realizacja tego zadania opiera się na użyciu narzędzia *makecert.bat* znajdującego się w katalogu instalacyjnym XAMPP w folderze *apache*. Po uruchomieniu jesteśmy proszeni o wpisanie wszystkich informacji potrzebnych przy tworzeniu certyfikatu.

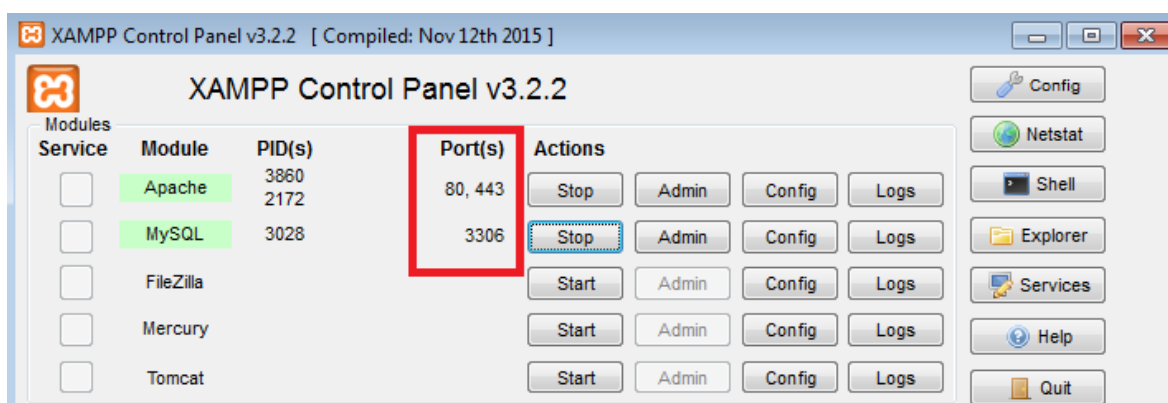


Rysunek 14: Certyfikat otrzymany po ukończeniu procedury *makecert*. Opracowanie własne, stan na dzień 04.12.2017

Konfiguracja XAMPP

Kolejnym krokiem, jaki należy zrobić to konfiguracja *Apache* tak, aby zamiast HTTP próbowało łączyć się przez HTTPS. W tym celu należy zmodyfikować plik znajdujący się w `xampp\apache\conf\extra\httpd-xampp.conf`, dopisując frazę ***SSLRequireSSL***. Kolejnym plikiem, który należy zmienić w ten sposób jest `xampp\webdav`.

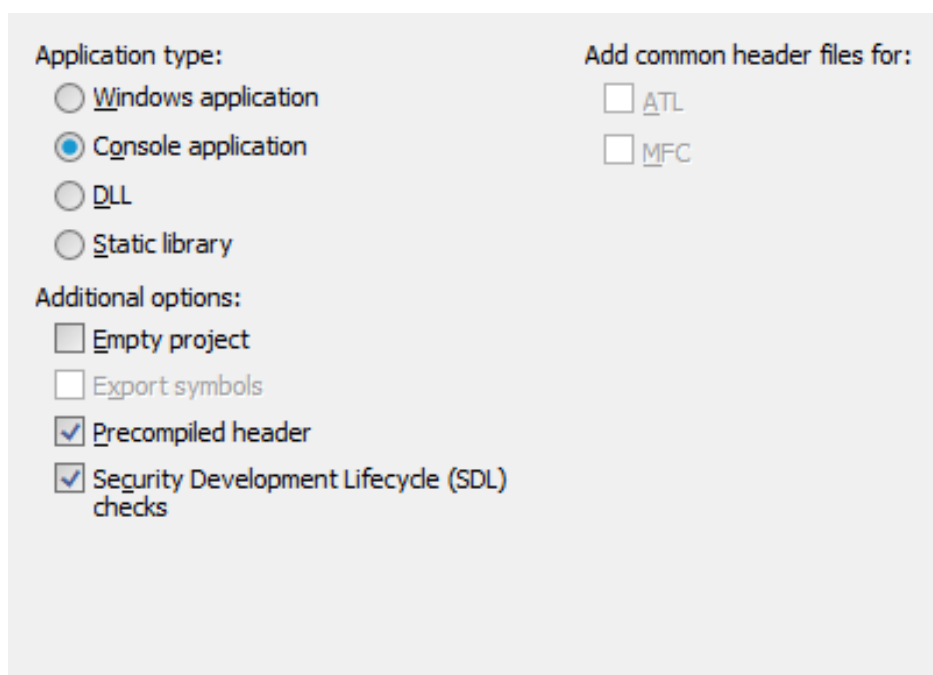
Ostatnim zadaniem jakie należy wykonać, jest konfiguracja tzw. *virtual hosta*. Przechodzimy do katalogu `xampp\apache\conf\extra\httpd-vhosts.conf` i tworzymy nową wirtualną konfigurację dla portu 443 (obsługującego SSL).



Rysunek 15: Widok przedstawiający serwer działający na dwóch portach. Opracowanie własne, stan na dzień 07.12.2017

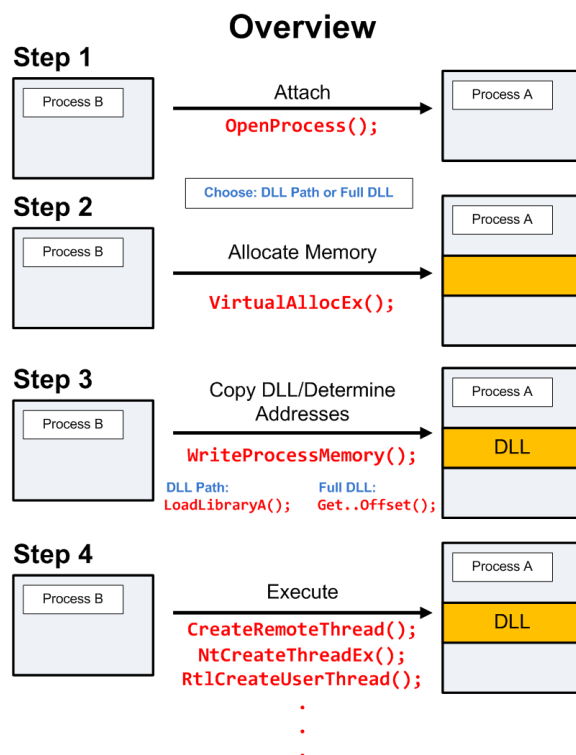
3.2.4 Środowisko programistyczne i język programowania

Realizacja zadania implementacyjnego odbyła się w zintegrowanym środowisku programistycznym firmy Microsoft - *Visual Studio 2013*. Wykorzystany język programowania to *C++*, ponieważ bardzo dobrze współpracuje on z Windows API oraz pozwala na wykorzystanie dużej liczby funkcji bibliotek standardowych. Typy aplikacji, które zostały wykorzystane to klasyczna aplikacja konsolowa oraz biblioteka DLL.



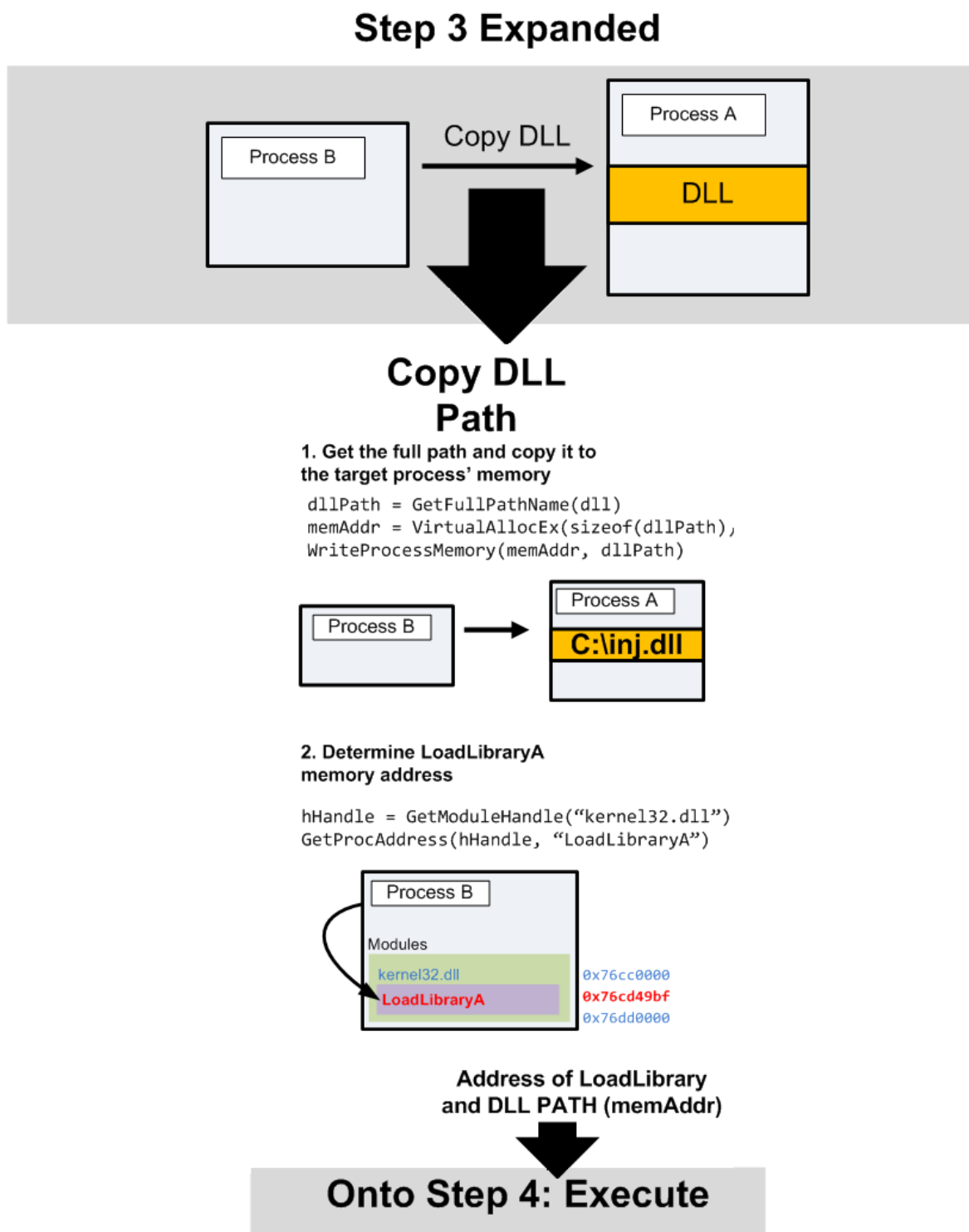
Rysunek 16: Widok przedstawiający opcje tworzenia projektu. Opracowanie własne, stan na dzień 07.12.2017

3.3 Wybrany schemat wstrzykiwania - *DLL Injection*



Rysunek 17: Schemat działania DLL Injection - zastosowane funkcje. Źródło [11], stan na dzień 08.12.2017

Ogólny schemat działania przedstawiony na Rysunku nr 17 bardzo dokładnie odzwierciedla kolejne kroki, jakie należy wykonać, aby uzyskać dostęp do innego programu oraz zarządzać sekwencją wykonywanych przez niego działań. Schemat ten jest analogiczny do przedstawionego algorytmu w części teoretycznej w Podrozdziale 2.3. Przedstawia on także funkcje Windows API, jakie posłużyły do wykonania kolejnych zadań. Kolejny rysunek, Rysunek nr 18, przedstawia bardziej szczegółowo wykonanie kroku trzeciego, czyli skopiowania ścieżki do modułu DLL do pamięci procesu zdalnego.



Rysunek 18: Wykonanie kroku nr 3. Źródło [11], stan na dzień 08.12.2017

Rozszerzenie kroku trzeciego uwzględnia dwa zasadnicze punkty:

1. Kopiowanie pełnej ścieżki do pamięci zdalnego procesu - złożliwe oprogramowanie (oznaczone na Rysunku 18 jako Process B) w tym przypadku rezerwuje region pamięci w wirtualnej przestrzeni adresowej Procesu A, a następnie zwraca adres do przydzielonego obszaru pamięci, w którym można zapisać ścieżkę do biblioteki DLL za pomocą `WriteProcessMemory()`.
2. Określenie adresu pamięci dla `LoadLibraryA` za pomocą `GetProcAddress()` oraz `GetModuleHandle()`.

3.4 Realizacja zadania

3.4.1 Przygotowanie biblioteki DLL

Przygotowana na potrzeby zadania biblioteka DLL zbudowana jest w oparciu o trzy pliki. Jeden z nich (`funkcje.h`) to plik nagłówkowy zawierający deklarację funkcji `Hello()`, natomiast pozostałe dwa to pliki źródłowe z rozszerzeniem `.cpp`. Plik `funkcje.cpp` zawiera definicję funkcji `Hello()`. Na początku za pomocą polecenia `MessageBox()` z odpowiednimi parametrami zostało utworzone okienko, które ma za zadanie tylko zaprezentować nam, że rzeczywiście podczas uruchomienia pliku głównego przechodzimy do wykonania funkcji znajdującej się w bibliotece. Podobny `Messagebox`, prezentujący kończenie wykonywania funkcji `Hello()`, został umieszczony na końcu. Dane okienka pozwalają na etapie pisania programu orientować się, w którym miejscu wykonywania się znajdujemy. Główną część biblioteki stanowi funkcja Windows API - `CreateProcess()`. Uruchamia ona nowy proces (w tym przypadku jest to `certmgr.exe`) z parametrem `-add`, tak aby dodać certyfikat (utworzony dla strony www.testhttp.dev) do zbioru certyfikatów systemu Windows. Dalsza część argumentów obejmuje wskazanie folderu, z którego pobierany jest dany certyfikat oraz nazwę magazynu docelowego, do którego po zakończeniu procesu zostanie dodany. Magazyny to nic innego jak odpowiednie rejestry. Wśród magazynów, do wyboru mamy:

- `currentUser` - wskazuje, że magazyn certyfikatów znajduje się w rejestrze `HKEY_CURRENT_USER`. Jest to wartość domyślna;
- `localMachine` - wskazuje, że magazyn certyfikatów znajduje się w rejestrze `HKEY_LOCAL_MACHINE`.

Na potrzeby złośliwego oprogramowania zaleca się korzystanie z `localMachine`, ponieważ użycie `currentUser` powoduje wyświetlenie dodatkowych informacji na ekranie komputera. Jest to niepożądane ponieważ sprzyja zorientowaniu się przez potencjalnego użytkownika - ofiarę, o wykorzystywaniu jego komputera. Ostatnim plikiem użytym do zbudowania biblioteki DLL jest `dll.cpp`. Zawiera on zbiór funkcji, które zostaną wykonane po załadowaniu modułu DLL. Przede wszystkim dla argumentu `DLL_PROCESS_ATTACH` (mówiącego o tym, że moduł DLL został załadowany do wirtualnej przestrzeni adresowej) rozpocznie się wykonywanie funkcji `Hello()`.

3.4.2 Wstrzyknięcie kodu

Procedura wstrzykiwania kodu została zaimplementowana na dwa sposoby. W pierwszym z nich wykorzystywana jest funkcja `CreateRemoteThread()`, odpowiedzialna za utworzenie zdalnego wątku. Kolejny sposób to użycie asynchronicznego wywołania procedur (APC - ang. Asynchronous Procedure Call).

Pierwsze podejście

Pierwsze podejście do rozwiązania problemu wstrzyknięcia kodu opiera się na załadowaniu modułu DLL poprzez funkcję `LoadLibraryA`. W pierwszej kolejności utworzony został projekt *Empty Project* o nazwie ***Implementacja***, do którego następnie dodane zostały trzy pliki:

- `main.cpp` - zawiera funkcję `main`;
- `dll.h` - zawiera deklarację klasy `Process`;
- `dll.cpp` - zawiera definicje zadeklarowanej klasy.

W pliku nagłówkowym zadeklarowane zostały dwie metody:

1. `int GetProcessID(nazwa procesu)` - funkcja zwracająca ID procesu o nazwie podanej jako argument;
2. `bool InjectDll(ścieżka do pliku DLL, ID procesu)` - funkcja wykonująca wstrzyknięcie DLL o podanej ścieżce do procesu o określonym ID.

Ich definicje znajdują się w `dll.cpp`. W celu znalezienia identyfikatora procesu o podanej nazwie, zastosowano funkcję `CreateToolhelp32Snapshot()`. Funkcja zwróciła uchwyt migawki systemowej, dzięki której możliwe jest uzyskanie informacji o procesach wykonywanych w systemie. Następnie została utworzona struktura typu `PROCESSENTRY32`, do której za pomocą `Process32First()` wpisane zostały dane pierwszego procesu z migawki. Cel działania `GetProcessID` to uzyskanie identyfikatora procesu, dlatego dla każdego kolejnego procesu (przechodzenie odbywa się przy wykorzystaniu `Process32Next`), nazwa procesu porównywana jest zadaną w argumencie wartością. Jeśli nazwa się zgadza, zwrócony zostaje nr ID. Metoda `InjectDll` rozpoczyna działanie od uzyskania uchwytu do procesu za pomocą funkcji `OpenProcess()`. Warto w tym miejscu zauważyć, że prawa dostępu do procesu określone w pierwszym argumencie ustawione są na `PROCESS_ALL_ACCESS`. Oznacza to, że będziemy mieli pełny dostęp do procesu. Następnie przy użyciu `VirtualAllocEx()` rezerwujemy obszar wirtualnej pamięci przestrzeni adresowej procesu. Jako typ zwracany otrzymujemy adres do tego obszaru, do którego za pomocą `WriteProcessMemory()` zapisujemy ścieżkę do biblioteki DLL. Przed uruchomieniem zdalnego wątku, wymagane jest wykonanie jeszcze jednej czynności. Musimy uzyskać uchwyt do modułu, który zawiera funkcję `LoadLibraryA`. Funkcja ta znajduje się w module *Kernel32*, dlatego niezbędne jest użycie `GetModuleHandle()`. Utworzenie zdalnego wątku w danym przypadku odbywa się z następującymi argumentami:
`CreateRemoteThread(pHandle, NULL, 0, (LPTHREAD_START_ROUTINE)`
`GetProcAddress(hK32, "LoadLibraryA"), address, 0, NULL);`

- uchwyt do procesu, dla którego powstanie dodatkowy wątek;
- domyślny (poprzez ustawienie jego wartości na `NULL`) wskaźnik struktury `SECURITY_ATTRIBUTES`;

- domyślny rozmiar stosu w bajtach;
- wskaźnik do funkcji, która zostanie uruchomiona w zdalnym wątku. Za pomocą `GetProcAddress()` otrzymujemy adres `LoadLibraryA`;
- wskaźnik do zmiennej przekazywanej jako parametr do funkcji określonej w poprzednim punkcie. Podawany jest tu adres zarezerwowany przez `VirtualAllocEx()`, w którym przechowywana jest ścieżka do DLL;
- flaga tworzenia wątku;
- wskaźnik do zmiennej posiadającej identyfikator nowo utworzonego wątku (nie jest nam potrzebny dlatego używamy `NULL`).

Po uruchomieniu funkcji tworzenia wątku, wywoływana jest funkcja `WaitForSingleObject()`, która ma za zadanie czekać, aż `LoadLibraryA` z wątku skończy działanie, czyli do czasu kiedy funkcja określona w `DllMain` się wykona. Dalsza część kodu dotyczy kończenia pracy programu. Usunięty zostaje wstrzyknięty kod oraz następuje zwolnienie pamięci.

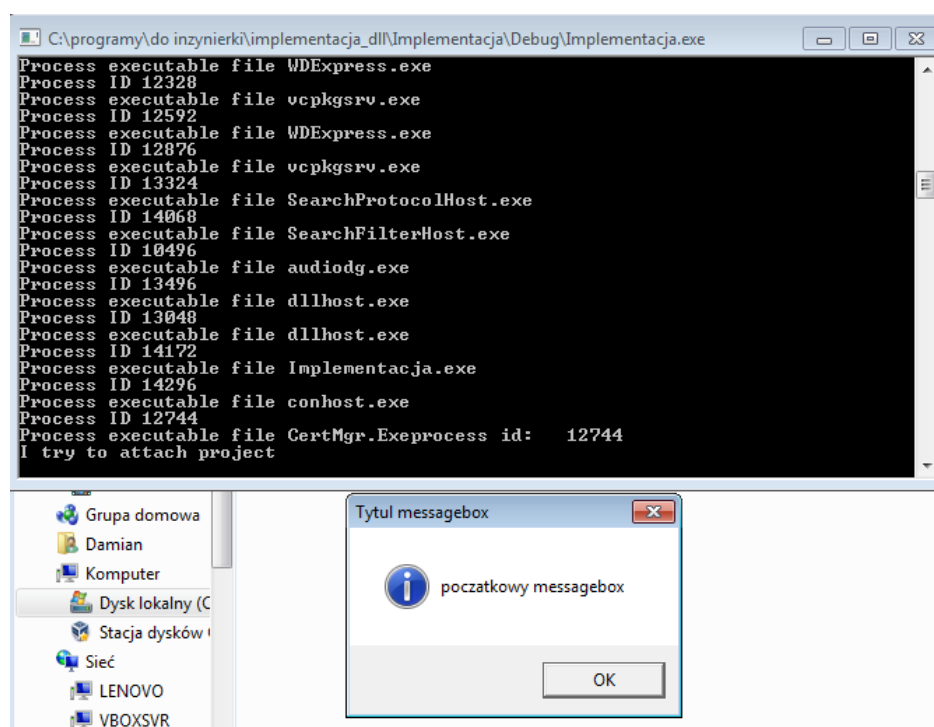
Plik `main.cpp` zawiera funkcję `main`, której działanie ogranicza się do stworzenia obiektu klasy `Process` oraz wykonaniu na nim metod określonych poprzednio¹⁴.

Drugie podejście

W drugim podejściu zaimplementowanym w projekcie *implementacja_vhost* znajduje się podobny zestaw plików. Dwa pliki (nagłówkowy i źródłowy) dla klasy `DLLInjection` oraz plik `main.cpp` zawierający, tak jak uprzednio, powołanie do życia obiektu typu `DLLInjection` oraz wywołanie na nim odpowiednich metod. Metody, jakie zostały zdefiniowane w klasie, to konstruktor, destruktor oraz `InjectDLLTosvchost()`. Wstrzyknięcie kodu w przypadku tej metody opiera się na uruchomieniu procesu w trybie wstrzymanym (za pomocą funkcji `CreateProcess()` z odpowiednią flagą ustawioną na `CREATE_SUSPENDED`). Podobnie jak w poprzednim podejściu rezerwowany jest obszar pamięci w wirtualnej przestrzeni

¹⁴na podstawie Maciej Pakulski, str. 32-35.

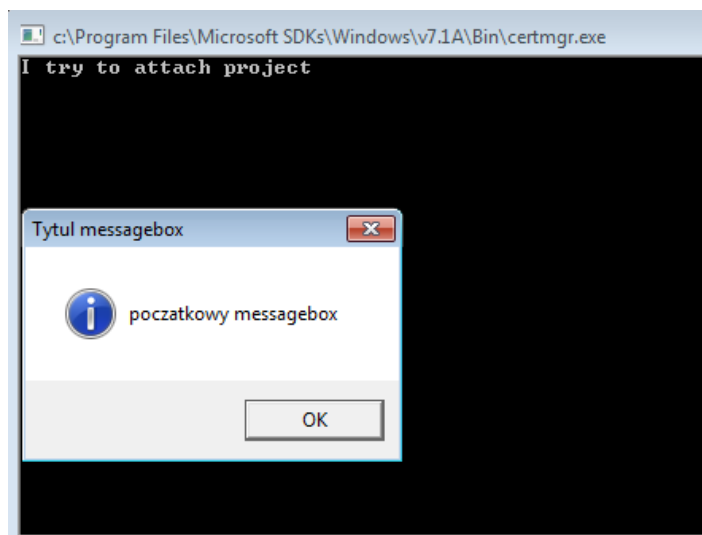
adresowej utworzonego procesu oraz zapisywana jest w tym obszarze ścieżka do modułu DLL. Nowym aspektem, który pojawia się w tym podejściu jest zastosowana funkcji `QueueUserAPC()`. Została ona omówiona w Podrozdziale 3.4.3. Dla przykładu, w danej implementacji po raz pierwszy wykonanie tej funkcji wiąże się z uruchomieniem *"LoadLibraryA"* z modułu *Kernel32*, działającej pod procesem utworzonym przez `CreateProcess()` z parametrem zawierającym adres z zapisaną ścieżką DLL. Kolejny raz funkcja `QueueUserAPC()` występuje z parametrem *ExitThread* także z modułu *Kernel32*, która zakończy działanie utworzonego wątku. Końcowy etap obejmuje wznowienie wykonywania macierzystego procesu oraz zamknięcie uchwytu do niego skojarzonego.



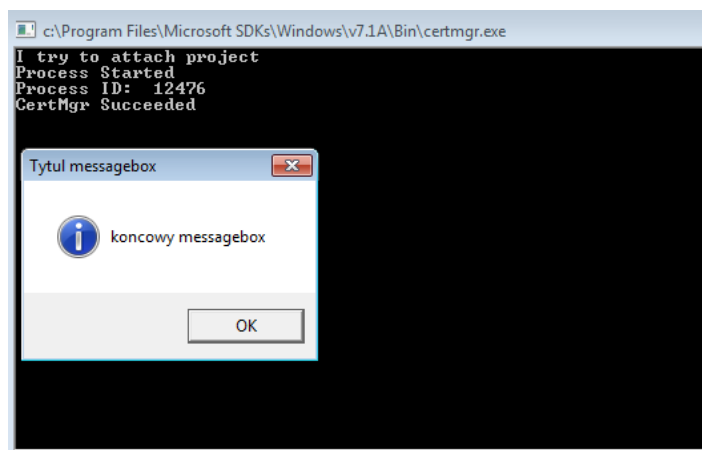
Rysunek 19: Wynik działania programu `Implementacja.exe` w fazie początkowej. Opracowanie własne, stan na dzień 08.12.2017

```
Process ID 12744  
Process executable file CertMgr.Exe  
process id: 12744  
I try to attach project  
Process Started  
Process ID: 13188  
CertMgr Succeeded
```

Rysunek 20: Wynik działania programu Implementacja.exe w fazie końcowej.
Opracowanie własne, stan na dzień 08.12.2017



Rysunek 21: Wynik działania programu implementacja_vhost.exe w fazie początkowej.
Opracowanie własne, stan na dzień 08.12.2017

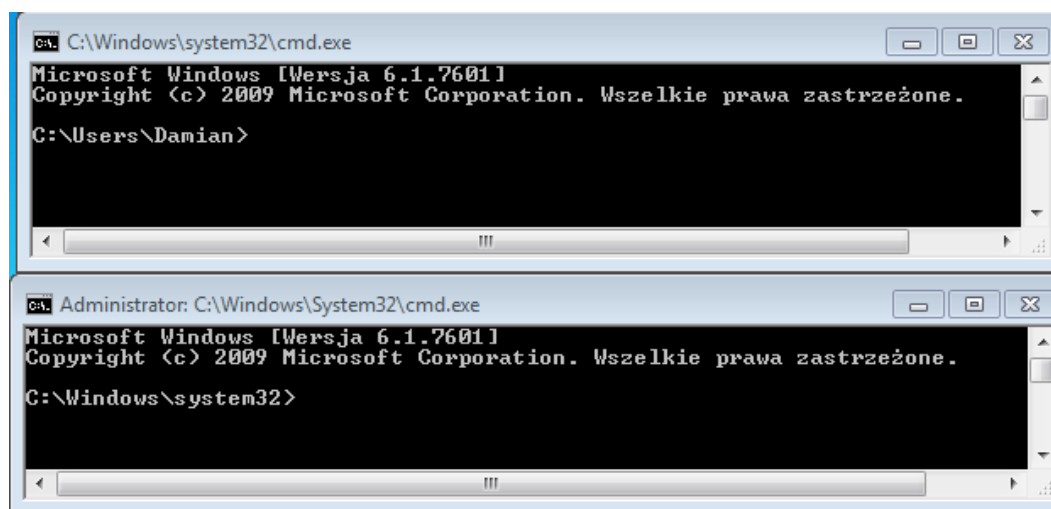


Rysunek 22: Wynik działania programu implementacja_vhost.exe w fazie końcowej.
Opracowanie własne, stan na dzień 08.12.2017

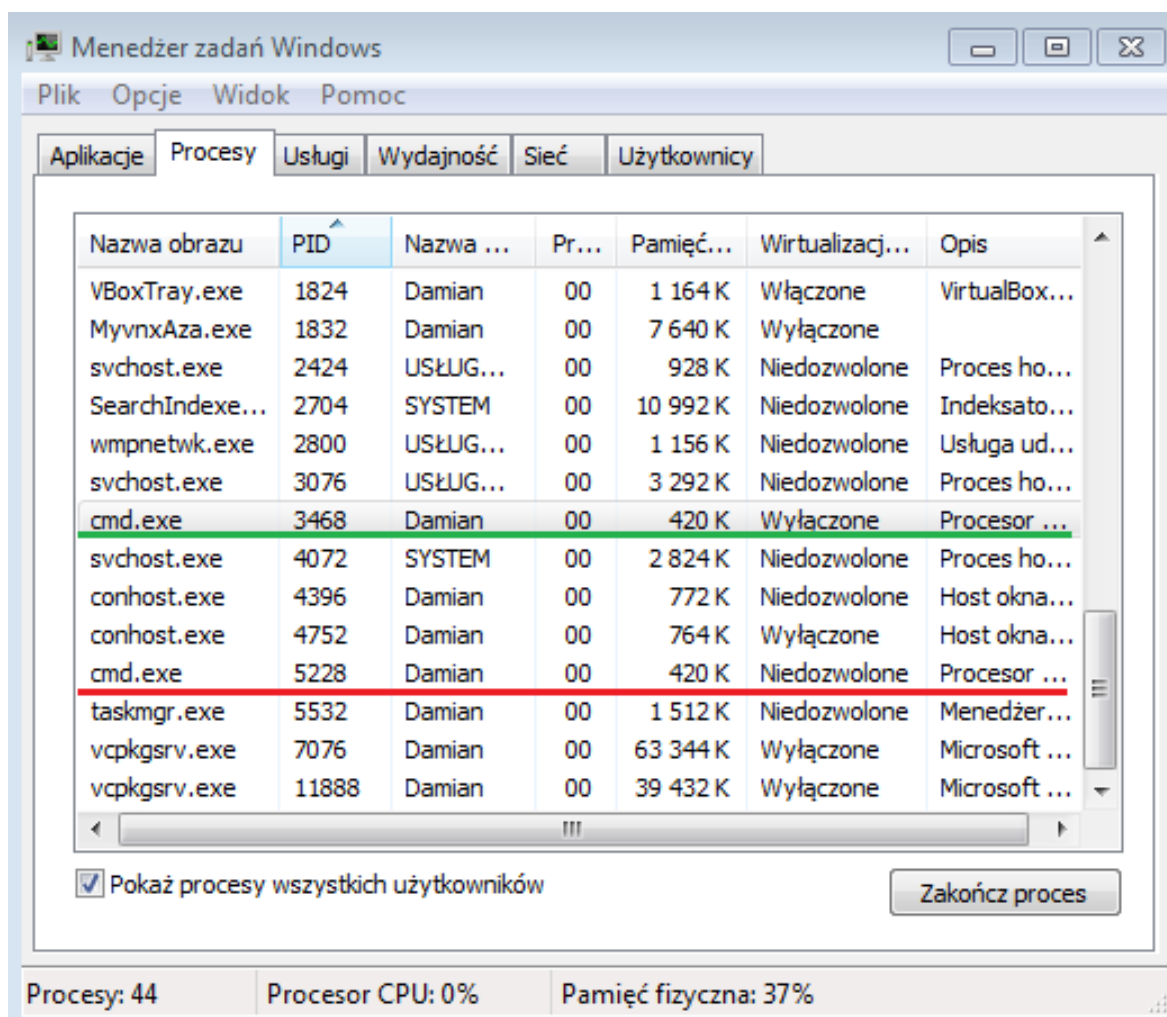
Rysunki 19 i 20 przedstawiają wykonanie się programu opierającego się na metodzie tworzącej nowy wątek. Program na początku wypisuje wszystkie procesy oraz szuka ID procesu zadanego jako argument. Wyświetlany zostaje także początkowy MessageBox, a następnie wykonywane jest dodanie certyfikatu do systemu. W przeciwieństwie do pierwszego programu, działanie drugiego (zaprezentowanego na Rysunkach 21 i 22) nie wyświetla procesów, ponieważ działa w oparciu o metodę utworzenia programu w trybie wstrzymanym.

3.4.3 Prawa administratora

Działanie metody DLL injection jest ściśle związane z poziomem uprawnień, jakie otrzymuje wstrzykiwany przez nas moduł. Biblioteka DLL, wyświetlająca MessageBox'a nie potrzebuje uprawnień administratora, dlatego wstrzyknięcie kodu zawierającego taką funkcjonalność jest łatwiejsze. Skutkuje to możliwością zaatakowania większej liczby procesów. Uwagę należy zwrócić na fakt, że w systemie Windows większość procesów nie zostaje uruchamiana z najwyższymi uprawnieniami. Najlepiej tę sytuację obrazuje Rysunek 23 oraz 24.



Rysunek 23: Dwa procesy cmd.exe uruchomione z różnymi uprawnieniami. Opracowanie własne, stan na dzień 08.12.2017



Rysunek 24: Menedżer zadań systemu Windows. Opracowanie własne, stan na dzień 08.12.2017

W oknie Menedżera zadań Windows procesy uruchomione z prawami administratora w zakładce *Wirtualizacja funkcji Kontrola konta użytkownika* mają wpis ustawiony na *Niedozwolone*, podczas gdy zwykłe procesy posiadają wpis *Wyłączone*. Posiadanie odpowiednich praw powoduje wiele problemów w aspekcie wstrzykiwania złośliwego oprogramowania. Załadowany moduł posiada bowiem prawa nie większe niż proces, pod który jego wykonywanie zostało podpięte. Wynika z tego, że aby dokonać modyfikacji w liście certyfikatów na poziomie systemu operacyjnego, należy albo dokonać wstrzyknięcia do procesu posiadającego już prawa administratora, albo

stworzyć proces z tymi prawami, do którego później nastąpi iniekcja. Niestety pierwszy pomysł nie wydaje się osiągalny. Dostęp do takich procesów, jak *svchost.exe*, *Isass.exe*, *etc.*, jest skutecznie blokowany mimo np. uruchomienia programu wstrzykującego z prawami administratora. Idealną sytuacją byłoby działanie *zwykłego* procesu z prawami administratora. Przez słowo zwykły rozumiemy proces uruchamiany wraz ze startem systemu. Ciężko jednak wyobrazić sobie sytuację, aby użytkownik uruchamiał np. kalkulator systemowy z pełnymi prawami. Drugi pomysł posiada większą liczbę rozwiązań. W celu utworzenia procesu z pełnymi prawami dostępu, program go tworzący musi zostać uruchomiony *jako administrator*. Istnieje sposób na rozwiązanie tego problemu.

Kontrola konta użytkownika

Kontrola konta użytkownika (ang. *User Account Control - UAC*) jest to komponent ochronny wprowadzony w systemie *Windows Vista* i dołączany w wersji bardziej przyjaznej użytkownikowi także w nowszych systemach (począwszy od *Windows 7* aż do *Windows 10*) ukierunkowany na poprawę bezpieczeństwa systemu operacyjnego poprzez ograniczanie aplikacjom uprawnień administratora do czasu kiedy administrator systemu nie zadecyduje o przyznaniu pełnego dostępu.

Ominięcie kontroli konta użytkownika (ang. *bypassing UAC*) jest niezbędne w celu automatycznego podniesienia uprawnień dla aplikacji. Osiągnięcie ustalonego poziomu praw realizowane jest w dwóch etapach.

1. Zapisanie struktury programu, który już z prawami administratora uruchomi program wykonujący DLL injection do bezpiecznej lokalizacji. Przez bezpieczną lokalizację rozumiany jest katalog systemowy, w którym aby wykonać jakieś modyfikacje należy mieć prawa administratora. W systemach rodziny Windows są to foldery np. *System32* bądź *ehome*.
2. Wywołanie odpowiedniego programu, który załaduje moduł wcześniej zapisany w bezpiecznej lokalizacji.

Pierwszy etap może zostać zrealizowany za pomocą dwóch sposobów:

- Użycie obiektu klasy IFileOperation COM - klasa ta posiada metodę, która pozwala na kopiowanie do bezpiecznej lokalizacji. Obiekt typu COM jest ustawiony na automatyczną eskalację uprawnień (ang. *auto elevation*), w związku z tym, proces kopiowany nie musi mieć znacznika (ang. *manifest*);
- Użycie Windows Update Standalone Installer (WUSA) - Automatyczny Instalator Rozszerzenia Windows. Narzędzie używające interfejsu API programu Windows Update Agent do instalowania pakietów aktualizacji.

Drugi etap polega na wykonaniu programu posiadającego pewne ściśle określone cechy z podmienionym plikiem np. DLL, przy czym zamiana wykonywana jest w momencie kopiowania struktury programu określonej w pierwszym etapie. Cechy, które muszą spełniać programy to:

- posiadanie podpisu;
- umieszczenie w bezpiecznej lokalizacji;
- znacznik musi określać właściwość programu na automatyczną eskalację uprawnień¹⁵.

W systemie Windows 7 występują trzy pliki wykonywalne, które posiadają odpowiednie cechy i mogą zostać zaatakowane. Ich wykaz wraz ze stowarzyszonymi z nimi plikami DLL określa Rysunek 25

¹⁵na podstawie Parvez, stan na dzień 08.12.2017

```
C:\windows\ehome\Mcx2Prov.exe
C:\Windows\ehome\CRYPTBASE.dll

C:\windows\System32\sysprep\sysprep.exe
C:\Windows\System32\sysprep\CRYPTSP.dll
C:\windows\System32\sysprep\CRYPTBASE.dll
C:\Windows\System32\sysprep\RpcRtRemote.dll
C:\Windows\System32\sysprep\UxTheme.dll

C:\windows\System32\cliconfg.exe
C:\Windows\System32\NTWDBLIB.DLL
```

Rysunek 25: Programy wraz z bibliotekami DLL posiadające podatności na exploity. Źródło [13], stan na dzień 08.12.2017

Celem ataku w wykonanej implementacji jest program Mcx2Prov.exe wraz z obsługiwaną przez niego biblioteką DLL - CRYPTBASE.dll. Metoda użyta do zapisania struktury do bezpiecznej lokalizacji to WUSA. W pierwszej kolejności, w bliźniaczym do *biblioteka.dll* projekcie o takiej samej nazwie, zaimplementowana została metoda Hello() do uruchomienia właściwego programu realizującego wstrzyknięcie kodu - *implementacja_vhost.exe* za pomocą CreateProcess(). Zbudowany projekt w rezultacie daje bibliotekę o nazwie cryptbase.dll, która następnie posłuży jako kopiowany moduł do bezpiecznej lokalizacji. Odpowiedzialny za uruchomienie programu wstrzykującego kod z prawami administratora w realizowanym projekcie jest moduł zaimplementowany pod nazwą *otwieranie.exe*. Opiera się on na omówionych metodach. W początkowej fazie, utworzony zostaje proces makecab.exe, mający za zadanie spakowanie pliku *cryptbase.dll*. Spakowanie to jest niezbędne ponieważ wykorzystana dalej metoda WUSA operuje na plikach z rozszerzeniem .cab. Następny krok to utworzenie procesu, który wykona wusa.exe z parametrem. Parametry to ścieżka do pliku spakowanego oraz miejsce docelowe przeznaczone do jego wypakowania - folder systemowy ehome, należący do zbioru

folderów określonych jako bezpieczna lokalizacja. Ostatnim etapem jest uruchomienie programu *Mcx2Prov.exe*. Jego wykonanie samo załaduje bibliotekę *cryptbase.dll* i ją wykona, przez co zostanie uruchomiony plik *implementacja_vhost.exe*, a następnie zrealizuje się właściwe wstrzyknięcie.

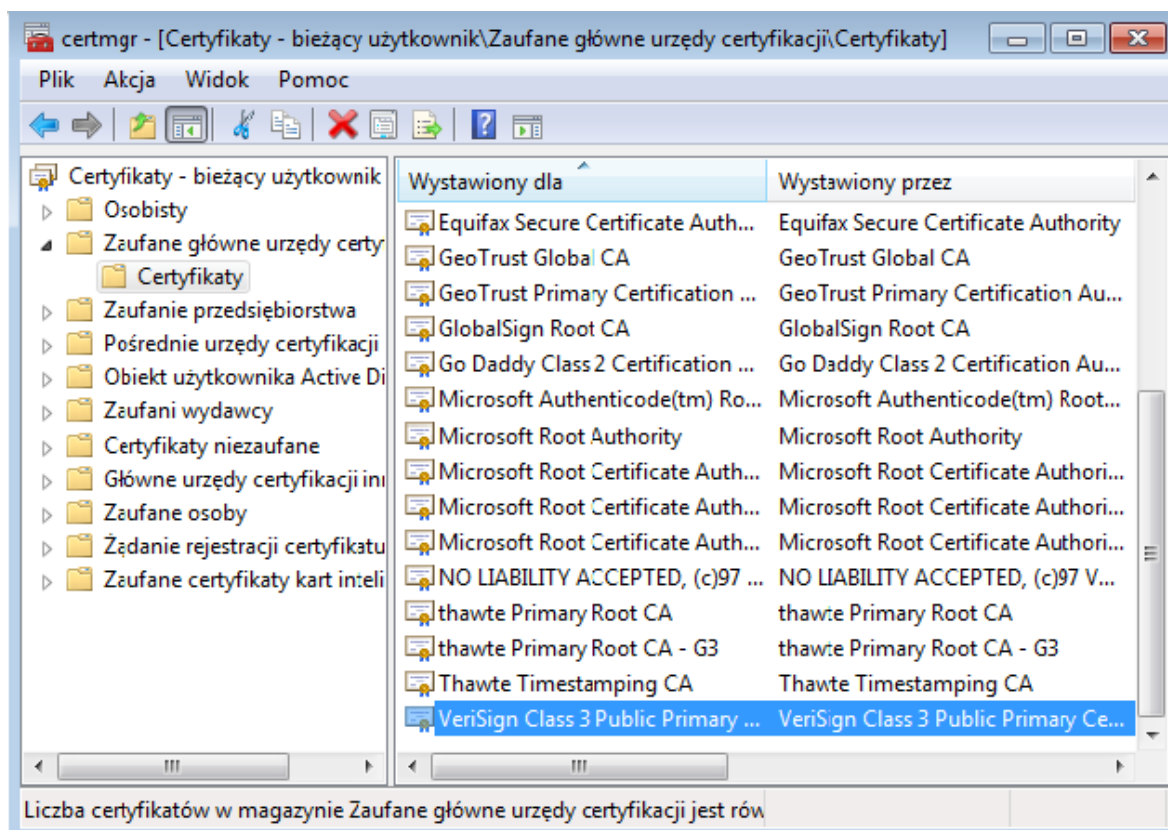
3.5 Wyniki

Poprawne wykonanie całości zaimplementowanego kodu ma na celu doprowadzenie do sytuacji, w której w systemie *Windows* zostanie dodany niepożądany certyfikat bez konieczności akceptacji tego faktu przez użytkownika. Program wstrzykujący składa się z kilku części:

1. Certyfikatu wystawionego dla strony internetowej *www.testhttp.dev*.
2. Biblioteki DLL (*biblioteka.dll*) będącej modułem wstrzykiwanym.
3. Biblioteki DLL (*cryptbase.dll*) odpowiedzialnej za uruchomienie loadera stowarzyszonej z *Mcx2Prov.exe*.
4. Programu *otwieranie.exe* mającego na celu skopiowanie *cryptbase.dll* do bezpiecznej lokalizacji w celu wykonania eskalacji uprawnień.
5. W zależności od przyjętej metody wstrzykiwania (Podejście pierwsze bądź drugie opisane w Podrozdziale 4.4.2):
 - Programu *Implementacja.exe* opierającego się na metodzie `CreateRemoteThread()`;
 - Programu *implementacja_vhost.exe* opierającego się na ACP.

Całościowy schemat działania opiera się na sentencji wykonywanych programów. Początkowo uruchamiany zostaje program *otwieranie.exe*. Kopiuje on uprzednio przygotowaną bibliotekę *cryptbase.dll* do katalogu *ehome*. Następnie wykonywany jest proces *Mcx2Prov.exe*, uruchamiający tą bibliotekę, której zadaniem jest dalsze uruchomienie np. programu *implementacja_vhost.exe*. Należy w tym momencie

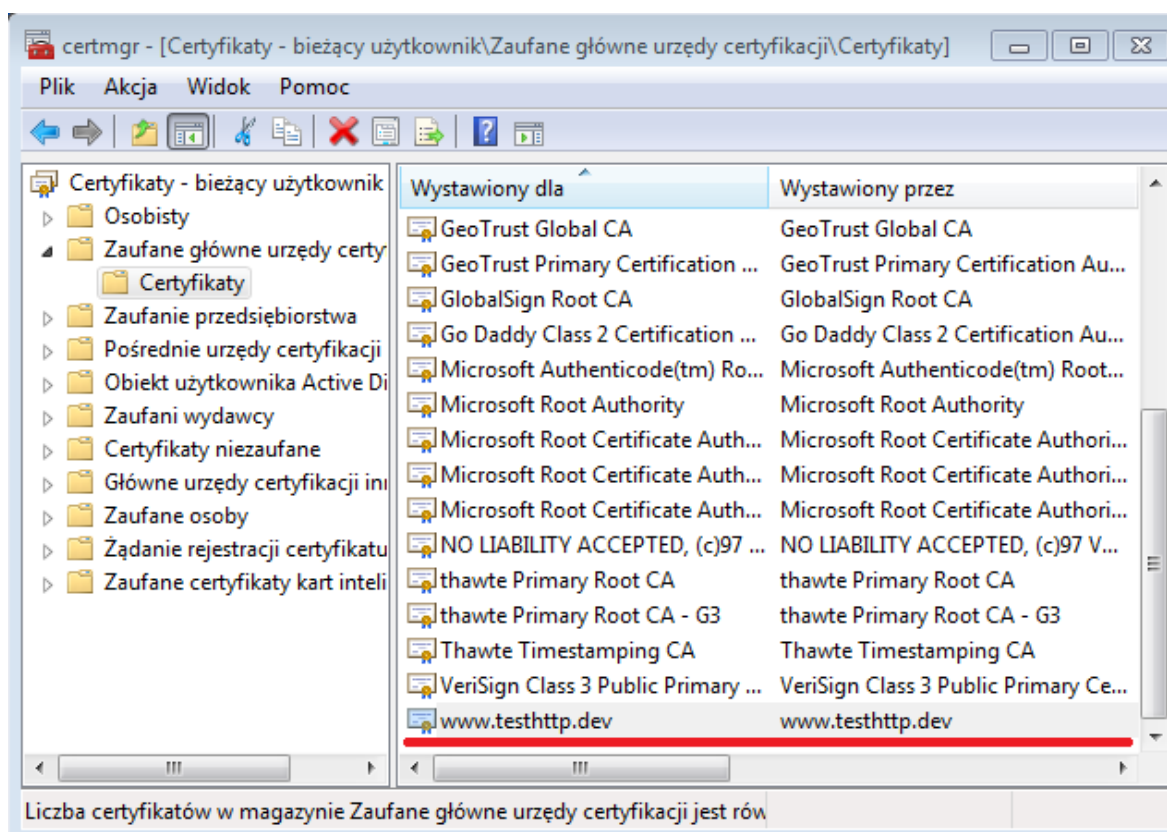
zwrócić uwagę na fakt, że wykonanie *Mcx2Prov.exe* podniosło poziom uprawnień automatycznie do praw administratora. Dlatego mimo tego, że *otwieranie.exe* został uruchomiony przez *zwykłego* użytkownika, to już *implementacja_vhost.exe* będzie działała na pełnych prawach. Wykonanie tego programu dokonuje wstrzyknięcia modułu DLL *biblioteka.dll*, którego celem jest dołączenie certyfikatu. Proces dobiega końca.



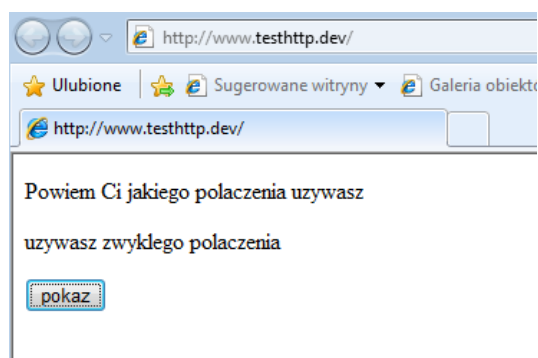
Rysunek 26: certmgr.msc z dostępnymi certyfikatami przed dokonaniem iniekcji. Opracowanie własne, stan na dzień 12.12.2017

```
I try to attach project
Process Started
Process ID: 7548
CertMgr Succeeded
```

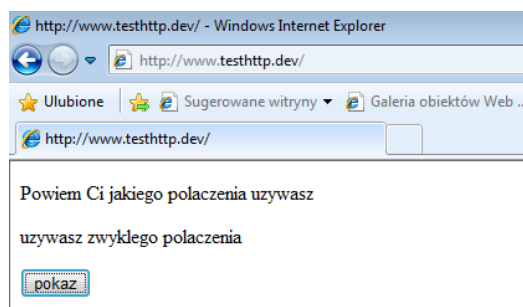
Rysunek 27: Wynik wykonania wstrzyknięcia kodu. Opracowanie własne, stan na dzień 12.12.2017



Rysunek 28: certmgr.msc z dostępnymi certyfikatami po dokonaniu iniekcji. Opracowanie własne, stan na dzień 12.12.2017

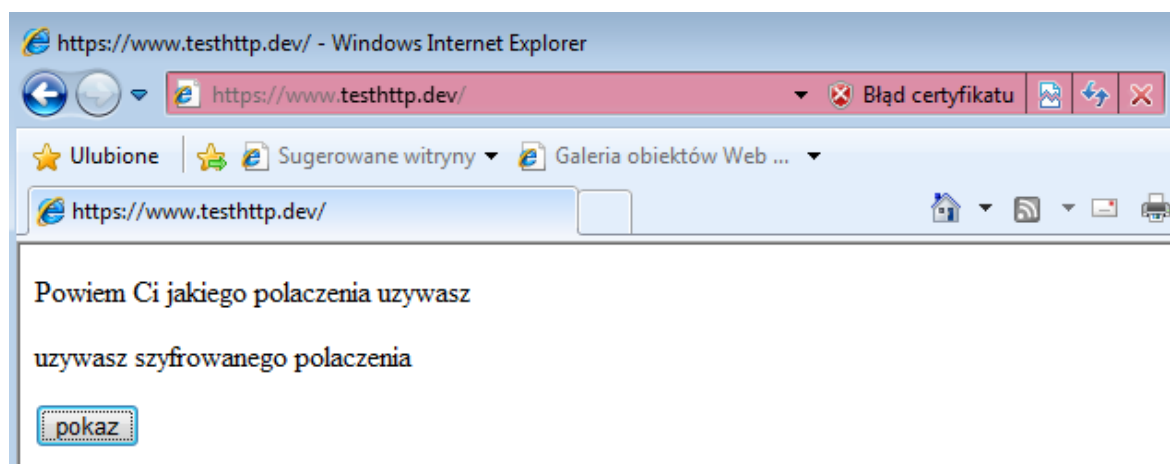


(a) Przed atakiem

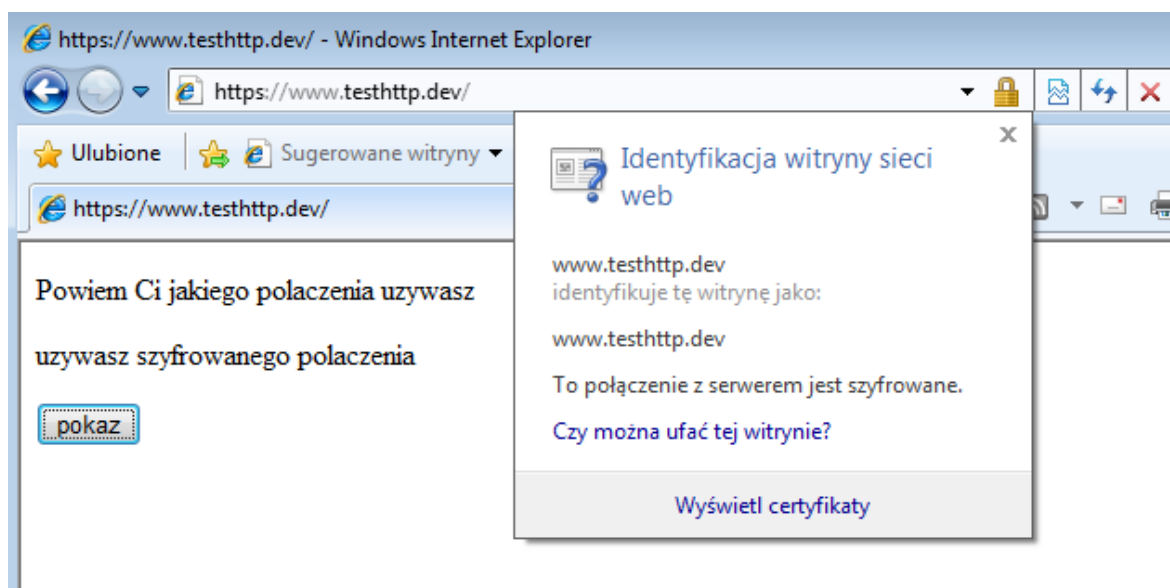


(b) Po ataku

Rysunek 29: Działanie strony internetowej dla protokołu http. Opracowanie własne, stan na dzień 12.12.2017



Rysunek 30: Działanie strony internetowej dla protokołu https przed wykonaniem ataku. Opracowanie własne, stan na dzień 12.12.2017



Rysunek 31: Działanie strony internetowej dla protokołu https po wykonaniu ataku. Opracowanie własne, stan na dzień 12.12.2017

Porównując Rysunki 26 i 28, można zauważyć, że po wykonaniu *DLL Injection*, do zbioru certyfikatów został dołączony certyfikat przygotowany wcześniej na potrzeby ataku. Dla połączenia nieszyfrowanego, nie ma żadnej różnicy pomiędzy stanem przed a po wstrzyknięciu. Wiąże się to z faktem, że *HTTP* nie wymaga identyfikacji witryny sieci Web poprzez certyfikat. Różnicę widać na Rysunkach 30 oraz 31. Przed dodaniem certyfikatu, użytkownikowi wyświetla się czerwony pasek adresu sygnalizujący, że szyfrowane połączenie, z którego próbuje korzystać nie jest zaufane. Po dodaniu certyfikatu po prawej stronie adresu widzimy kłódkę świadczącą o pomyślnej weryfikacji połączenia. Nieświadomy użytkownik może być poddany dalszemu atakowi (podmiana strony internetowej etc.).

Podsumowanie

Celem niniejszej pracy dyplomowej było zapoznanie z podstawowymi technikami wstrzykiwania kodu oraz wykonanie przykładowej implementacji obrazującej schemat potencjalnego ataku. Wykorzystując *DLL injection*, czyli metodę korzystającą z funkcji systemowej *LoadLibraryA* do uruchomienia własnej biblioteki DLL, wykonany został atak na zbiór certyfikatów systemu Windows 7 w wersji 32 bitowej. Zaletą tej metody wstrzykiwania jest prostota jej użycia, która wynika w głównej mierze z tego, że nie jest wymagana znajomość formatu dołączanego PE oraz nie ma konieczności samodzielnego implementowania modułu ładującego daną bibliotekę.

Niestety poza szeroką gamą zalet, wstrzykiwanie biblioteki DLL posiada również wiele wad. Podczas analizy zaimplementowanego programu zauważono wiele wymagań, które musiały zostać spełnione aby odnieść sukces podczas ataku. Przede wszystkim największym problemem było posiadanie odpowiednich uprawnień. Pierwszy program *ładujący* o nazwie *Implemetacja.exe* opierający się na wykorzystaniu *CreateRemoteThread()*, wstrzykiwał się do procesu znalezionej podczas wykonywania tak zwanej migawki systemowej. Początkowo podjęte zostały próby wykorzystania takich programów jak notatnik czy kalkulator, niemniej jednak nie posiadały one odpowiednich praw, które są wymagane. Próby podpięcia się pod procesy systemowe nie przyniosły pożądanego rezultatu, ze względu na fakt, iż są to procesy bardzo dobrze chronione, do których dostęp jest ograniczony. Jedynym rozwiązaniem było napisanie dodatkowego programu o nazwie *otwieranie.exe*, który uruchamiałby program, który był celem ataku z uprawnieniami administratora. Ponownie wystąpiły problemy. Mimo tego, że odpowiednie prawa zostały już przyznane, to nadal korzystaliśmy z migawki. Oznacza to, że działaliśmy na programach już uruchomionych. W systemach z rodziny

Windows niemożliwa jest eskalacja uprawnień w procesie wykonywanym, dlatego też powstał kolejny moduł o nazwie *implementacja_vhost.exe*, który dopiero wywoływał program-ofiarę za pomocą `CreateProcess()`. Dopiero dla tak utworzonej sentencji działań, możliwe było poprawne przeprowadzenie ataku.

Pomimo poprawnego działania programu, warto jeszcze zwrócić uwagę na fakt, iż podczas ataku nie jesteśmy do końca anonimowi. Oczywistym jest, że użytkownik posiadający małą wiedzę na temat wykorzystywanego przez niego systemu czy urządzenia nie zorientuje się, że coś niepożądanego dzieje się w jego komputerze. Zupełnie odwrotnie jest w przypadku świadomego użytkownika. W menedżerze zadań pojawia się odpowiedni wpis informujący o działaniu szkodliwego programu, natomiast przy pomocy bardziej zaawansowanych narzędzi monitorujących, można podejrzec jakie biblioteki są ładowane do pamięci poszczególnych procesów. Wartością działającą na korzyść zaimplementowanego programu jest fakt, że cały atak wykonuje się w ułamku sekundy, przez co możliwym jest wykorzystanie nieuwagi użytkownika.

Wykorzystywanie klasycznego DLL injection z powodów opisanych powyżej nie jest najczęściej stosowaną techniką wśród twórców malware'u, niemniej jednak na użytek własny bądź prostych ataków warto się z nią zaznajomić. Podsumowując, podjęcie się implementacji szkodliwego oprogramowania było dla autora początkiem wdrażania się w odpowiednie mechanizmy rządzące bezpieczeństwem systemów informatycznych.

Bibliografia

- [1] Simon Kemp. Digital in 2017: Global overview, 2017. URL <https://wearesocial.com/special-reports/digital-in-2017-global-overview>.
- [2] Dawid Farbaniec. *Techniki twórców złośliwego oprogramowania. Elementarz programisty*. Helion, 2014.
- [3] Redakcja naukowa Mateusz Jurczyk i Gynvael Coldwind. *Praktyczna inżynieria wsteczna*. Wydawnictwo Naukowe PWN, 2016.
- [4] Eran Goldstein. Reverse engineering - shellcodes techniques. *Haking on demand*, 03:30–33, 2013.
- [5] Offensive Security. Msfvenom, 2017. URL <https://www.offensive-security.com/metasploit-unleashed/msfvenom/>.
- [6] MSDN Microsoft Documentation. Writeprocessmemory function, 2017. URL [https://msdn.microsoft.com/pl-pl/library/windows/desktop/ms681674\(v=vs.85\).aspx](https://msdn.microsoft.com/pl-pl/library/windows/desktop/ms681674(v=vs.85).aspx).
- [7] MSDN Microsoft Documentation. Createremotethread function, 2017. URL [https://msdn.microsoft.com/pl-pl/library/windows/desktop/ms682437\(v=vs.85\).aspx](https://msdn.microsoft.com/pl-pl/library/windows/desktop/ms682437(v=vs.85).aspx).
- [8] MSDN Microsoft Documentation. Queueuserapc function, 2017. URL [https://msdn.microsoft.com/en-us/library/windows/desktop/ms684954\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms684954(v=vs.85).aspx).

- [9] Zameer Kharal. How to install a local web server with xampp (in urdu), 2014. URL <http://urdutipsworld.blogspot.com/2014/12/how-to-install-local-web-server-with.html>.
- [10] Dye Mark A. Rufi Antoon W. *Akademia sieci Cisco CCNA Exploration semestr 1 Podstawy sieci*. Wydawnictwo Naukowe PWN, 2013.
- [11] Brad Antoniewicz. Windows dll injection basics, 2013. URL <http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html>.
- [12] Maciej Pakulski. Dll injection. *Haking*, 10:32–38, 2008.
- [13] Parvez. Bypassing windows user account control (uac) and ways of mitigation, 2014. URL <https://www.greyhathacker.net/?p=796>.
- [14] Joe Schmoe. Bypass uac using dll hijacking, 2016. URL <https://null-byte.wonderhowto.com/how-to/bypass-uac-using-dll-hijacking-0168600/>.
- [15] Rui Reis. Inject all the things, 2017. URL <http://blog.deniable.org/blog/2017/07/16/inject-all-the-things/>.
- [16] Raymond. How to bypass user account control (uac) in windows, 2016. URL <https://www.raymond.cc/blog/weaknesses-windows-7-user-account-control/>.
- [17] Francisco Carrasco. Visual studio 2013: Una nueva generación de aplicaciones para desarrolladores, 2013. URL <http://www.cioal.com/2013/11/13/la-nueva-generacion-de-aplicaciones-llega-con-el-lanzamiento-de-visual-studio/>.