Shadows with a chance of BlackNix

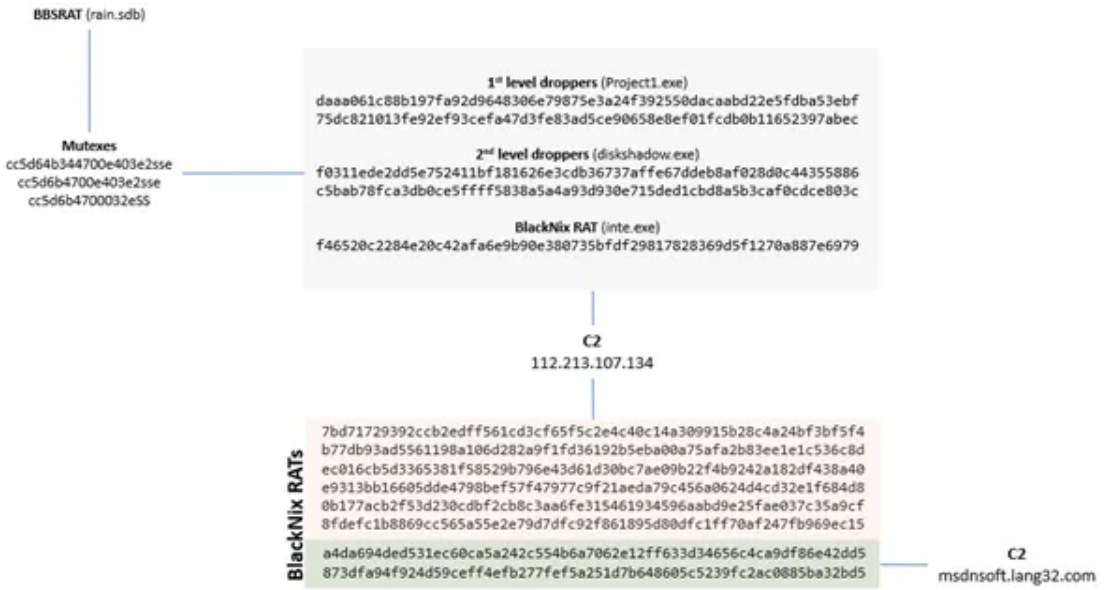
By asuna amawaka

Published: 2020-05-06 · Archived: 2026-04-05 14:02:26 UTC

In the last post, I did an analysis of a set of *BBSRAT* samples that are characterized by unique mutexes (cc5d64b344700e403e2sse, cc5d6b4700e403e2sse, cc5d6b4700032eSS) and calls back to a known **Winnti Group** C2 (bot[.]googlerenewals[.]net). In this post, I'm going to continue on analysis of samples related to the abovementioned mutexes.

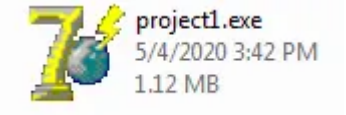
When I started on this analysis journey, I was hoping to find more *BBSRAT* samples. However, the results I arrived at deviated from expectations, and instead I found a set of dropper malware that used the same mutexes as those found in the *BBSRAT* samples I analyzed. The final payload dropped by these droppers is the *BlackNix RAT*. Pivoting from the C2 called from this *BlackNix RAT*, more *BlackNix RAT*s were found on VirusTotal. I was unable to find any technical blogs on the *BlackNix RAT*, and hence, here I am.

The following diagram is a sort of a "signpost" for this writing.



Let's dip into the first dropper!

Project1.exe



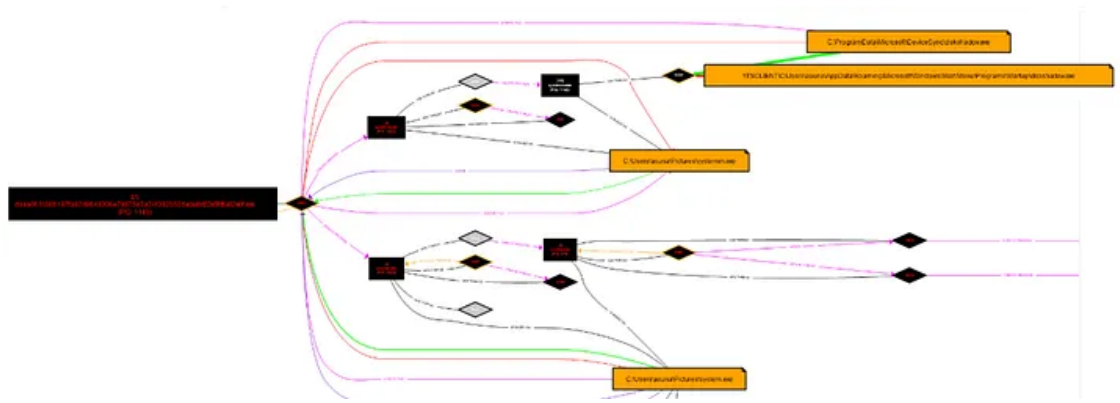
The following files are likely from the same source code:
daaa061c88b197fa92d9648306e79875e3a24f392550dacaabd22e5fdba53ebf
75dc821013fe92ef93cefa47d3fe83ad5ce90658e8ef01fcdb0b11652397abec

Judging from the executables' icon, it looks like the samples are written with Borland Delphi 7, and sadly the executables' compilation timestamps are 1992–06–19 22:22:17 (a well-known bug in Delphi 4–2006). I didn't really look into which versions are affected by the compilation time bug, because that's beside the point. The samples' compilation time can still be deduced with the timestamp within the executables' resources. (Thanks to Adam's old post on this[1])

```
C:\Users\asuna\Desktop>perl pect.pl 75dc821013fe92ef93cefa47d3fe83ad5ce90658e8ef01fcdb0b11652397abec\project1.exe

====================================
 PEct v0.1, Hexacorn.com, 2014-12-06
====================================

File:        75dc821013fe92ef93cefa47d3fe83ad5ce90658e8ef01fcdb0b11652397abec\project1.exe
PE Comp.:    1992-06-19 22:22:17 2A425E19, 708992537
.rsrc comp.: 2019-01-08 08:47:28 4E2845EE, 1311262190

C:\Users\asuna\Desktop>perl pect.pl daaa061c88b197fa92d9648306e79875e3a24f392550dacaabd22e5fdba53ebf\project1.exe

====================================
 PEct v0.1, Hexacorn.com, 2014-12-06
====================================

File:        daaa061c88b197fa92d9648306e79875e3a24f392550dacaabd22e5fdba53ebf\project1.exe
PE Comp.:    1992-06-19 22:22:17 2A425E19, 708992537
.rsrc comp.: 2019-03-28 15:13:52 4E7C79BA, 1316780474
```

This dropper will drop 2 files system.exe and systemm.exe into %USERPROFILE%\Pictures, execute them and write a diskshadow.exe to C:\ProgramData\Microsoft\DeviceSync\.

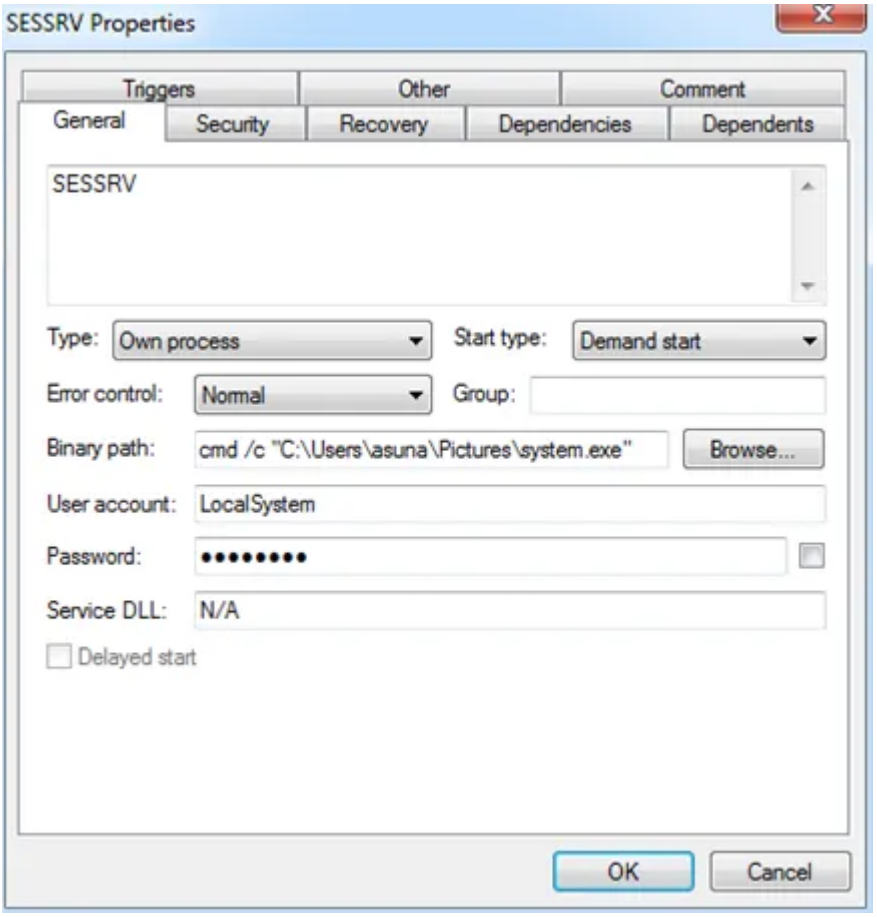
Press enter or click to view image in full size



system.exe, systemm.exe
SHA256: AEB61477C3F4F2D76AF0DC97B19B01F73C8ADA1FCE91D66E8B0E489E2E807430

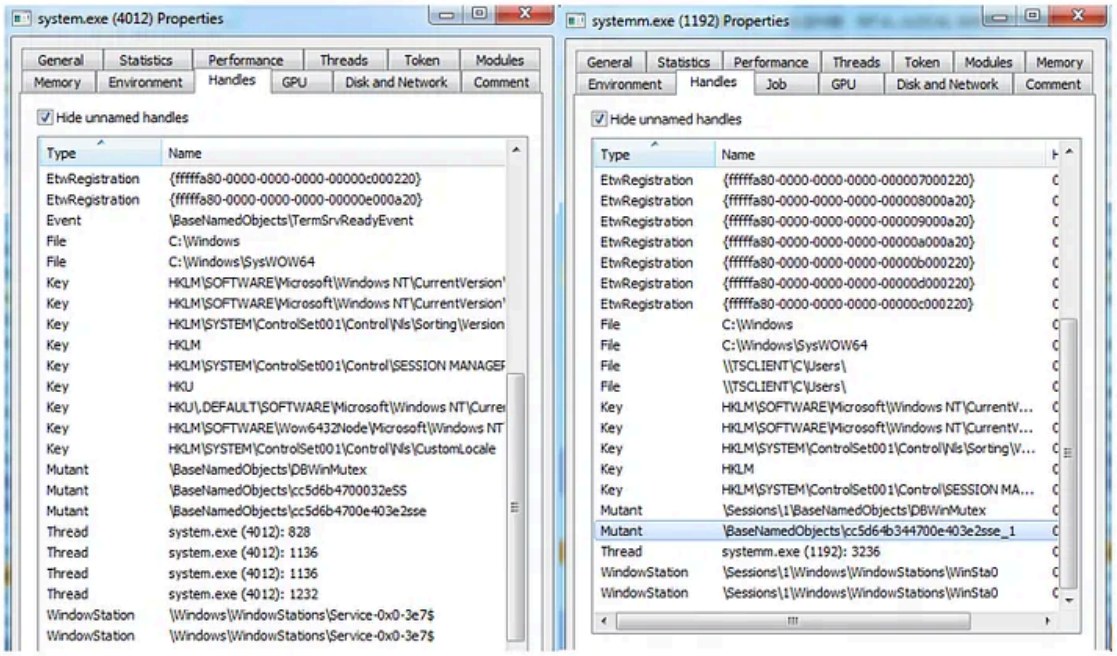
Execution of system.exe creates a service to execute itself within the context of the service.

> C:\Windows\System32\sc.exe create SESSRV binpath= "cmd /c \"C:\Users\asuna\Pictures\system.exe\""
>
> C:\Windows\System32\net.exe start SESSRV



The mutexes are set within the execution of system.exe and systemm.exe.

Press enter or click to view image in full size



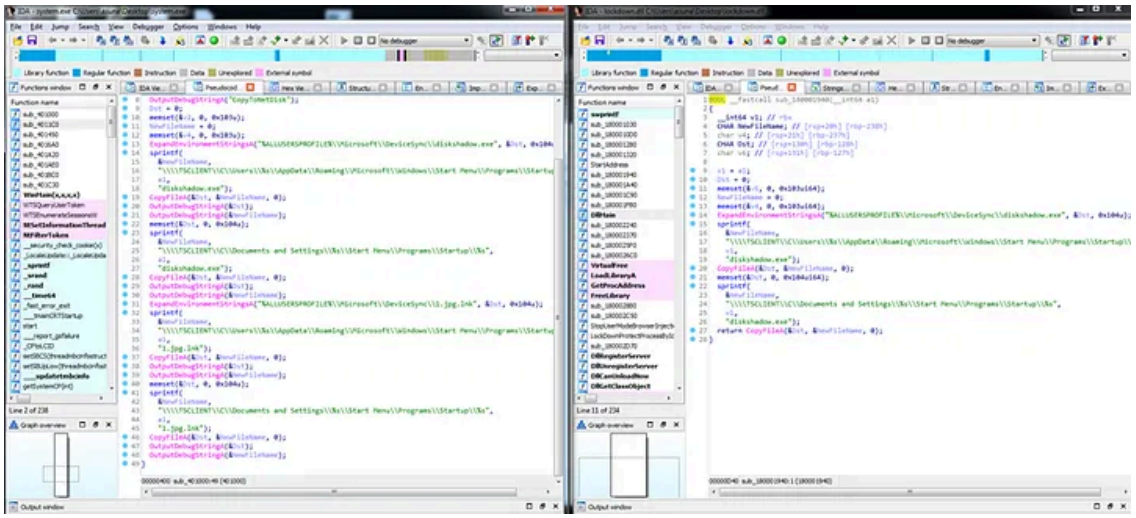
systemm.exe attempts to copy diskshadow.exe to a network location \\TSCLIENT\%userprofile%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\. The sample

also attempts to locate and copy a 1.jpg.lnk within the same directory as diskshadow.exe, but this file was not created by the previous dropper.

In gist, the sample's job is to copy the payload diskshadow.exe to a host that is connected to the current victim via RDP and set persistency to run at startup. This is possibly a tool meant for lateral movement within the victim network. The same technique is also found in the execution of the *BBSRAT* samples analyzed previously.

The following screen captures referenced one of the *BBSRAT* trinity files, lockdown.dll (MD5: 166D28FF69019D9991EECBD26DC1E266):

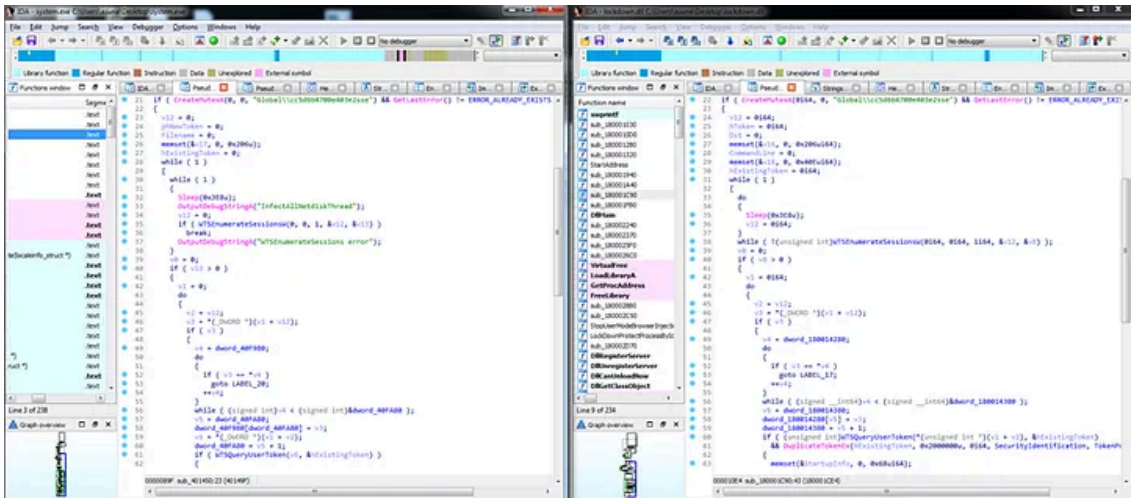
Press enter or click to view image in full size



Copy file to network location. Left: system.exe; Right: lockdown.dll

The mutexes come into play with the same "usage" as what was seen in the *BBSRAT* samples as well.

Press enter or click to view image in full size



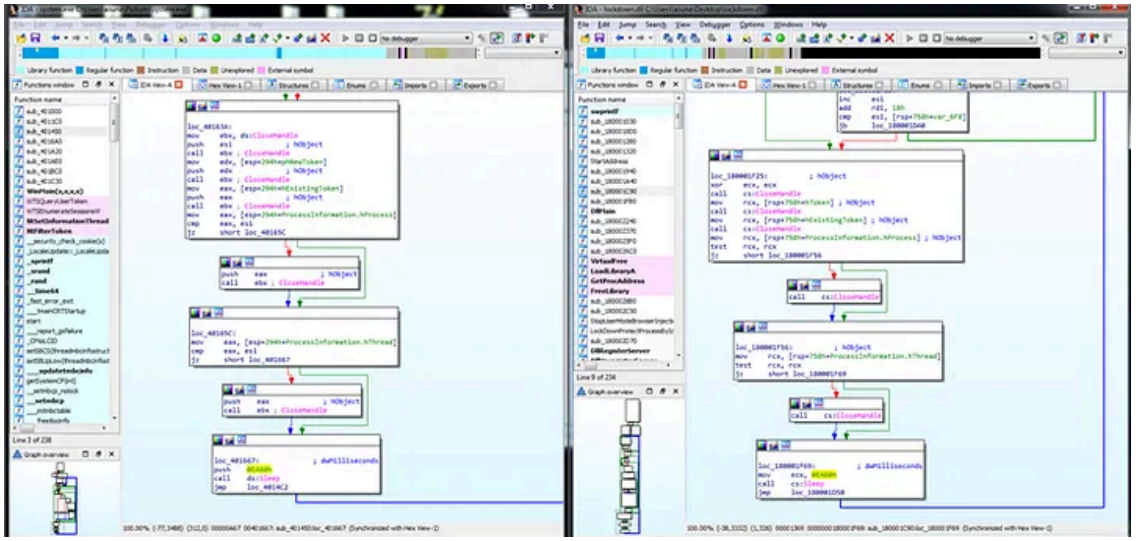
One of the places where mutex is set. Left: system.exe; Right: lockdown.dll

Given that even the sleep counter is identical, I would suspect that the two executables might share the same "base code" (or perhaps copied from the same "reference code"). The proximity of their compilation time also suggests

that perhaps the same author is behind both executables.

System.exe: 13 May 2018 19:34:27
Lockdown.dll: 6 May 2018 17:59:24

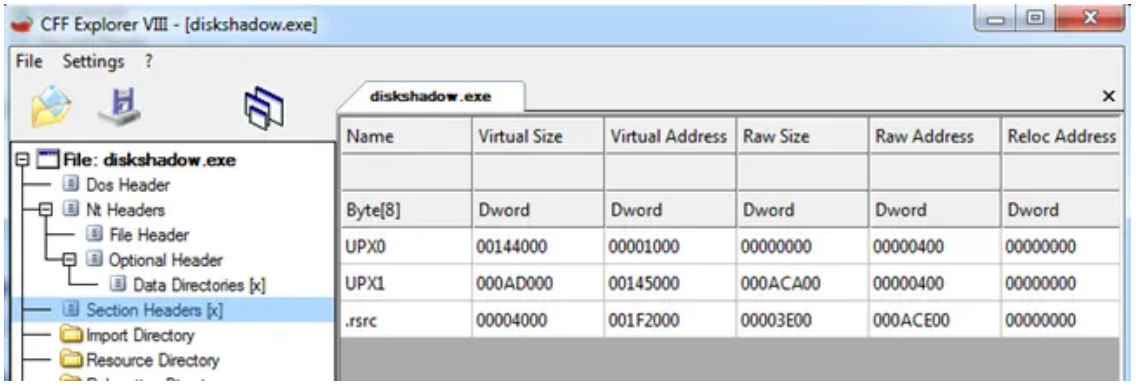
Press enter or click to view image in full size



Same sleep counter. Left: system.exe; Right: lockdown.dll

Enough of the dropper, let's look at the actual evil payload now.

Diskshadow.exe



UPX section names

With just one look at it, we'd know it's UPX-packed. So let's unpack it quickly.

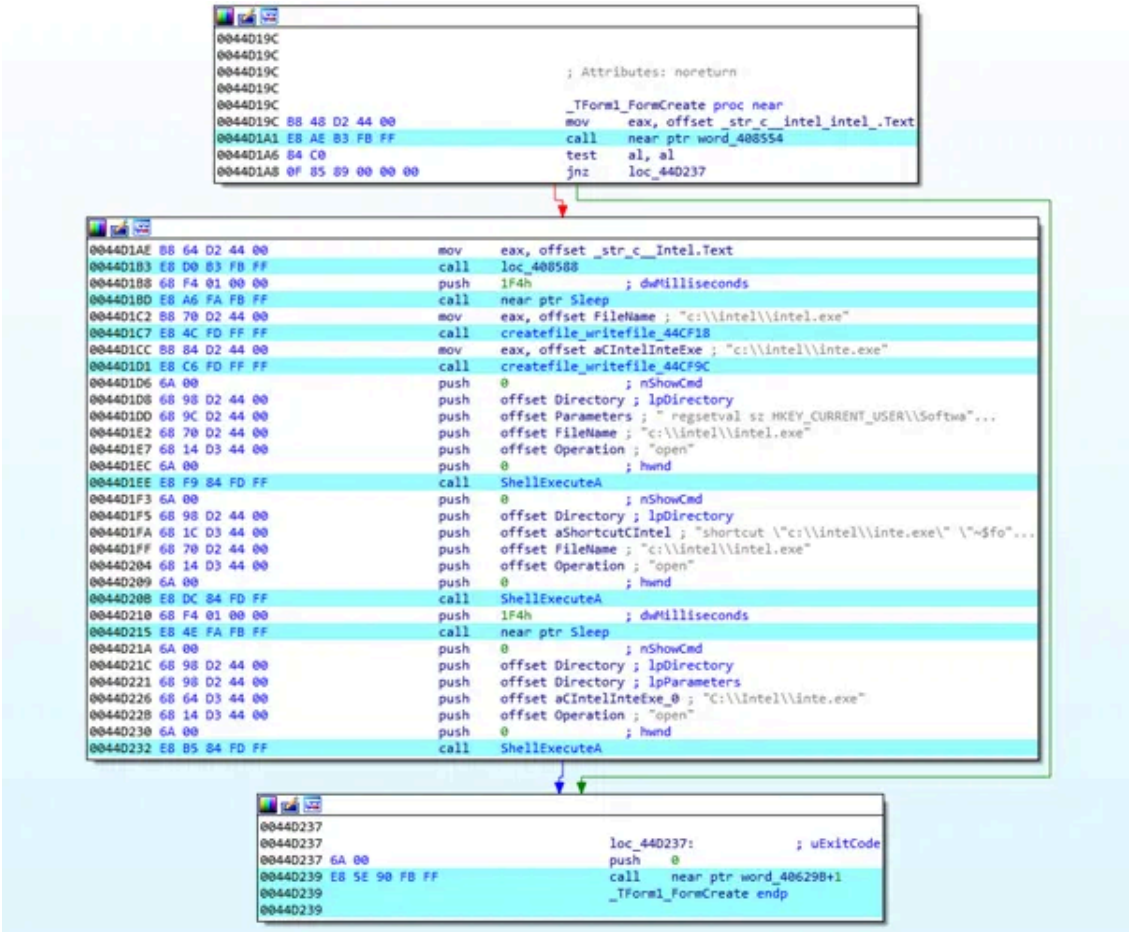
```
C:\Users\asuna\Desktop
λ upx -d diskshadow.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2017
UPX 3.94w        Markus Oberhumer, Laszlo Molnar & John Reiser   May 12th 2017

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
   1990144 <-    723968   36.38%    win32/pe     diskshadow.exe

Unpacked 1 file.
```

After unpack:

MD5: 40835ED7C92F33F7F377D4472228CB65
SHA1: 033E97D4AC3AE3CEC00A206F2AD5CCC922DBD326
SHA256: C5BAB78FCA3DB0CE5FFFF5838A5A4A93D930E715DED1CBD8A5B3CAF0CDCE803C

With the assistance of Procmon, we can see that the binary will drop 2 files, intel.exe and inte.exe into the C:\intel directory. I located the code responsible for creating the files (refer to screen capture below). The binary also sets 2 persistency mechanisms. Note the presence of Simplified Chinese words within the name of the registry key — 更新计划程序 (translates to " Update Schedule Program"). This is not the only place where Simplified Chinese words are observed.

> regsetval sz HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run "intel更新计划程序" "c:\intel\inte.exe"',0
>
> shortcut "c:\intel\inte.exe" "~$folder.startup$" "Windows Calculator"



Set persistency

The two files dropped are:

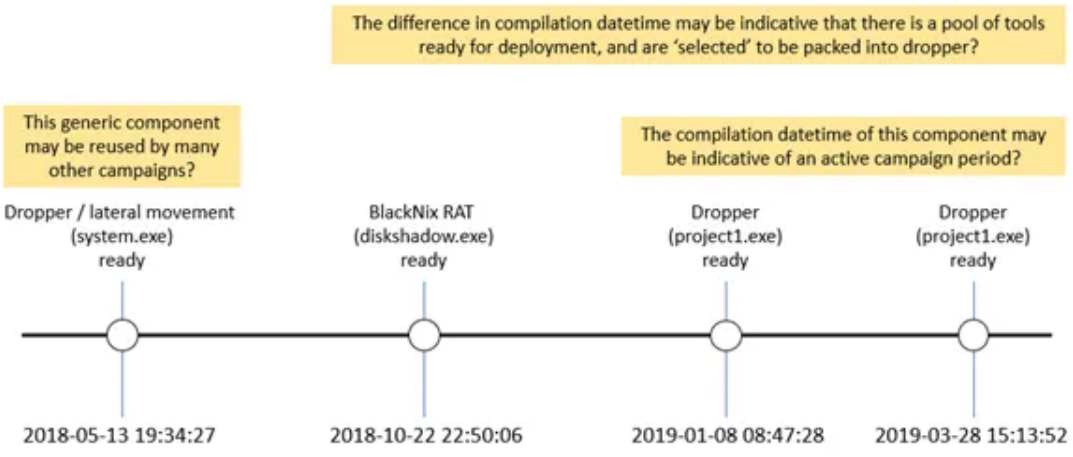
intel.exe
SHA256: 12459A5E9AFDB2DBFF685C8C4E916BB15B34745D56EF5F778DF99416D2749261
This is the NirCmd executable from Nirsoft. NirCmd is a small command-line utility that allows you to do some useful tasks without displaying any user interface.

inte.exe
SHA256: F46520C2284E20C42AFA6E9B90E380735BFDF29817828369D5F1270A887E6979
This is the actual *BlackNix RAT*, which is the meat that we want to analyze.

Both diskshadow.exe and inte.exe are written in Borland Delphi and their compilation datetime stamps are as follow:

> File: C:\ProgramData\Microsoft\DeviceSync\diskshadow.exe
> PE Comp.: 1992–06–19 22:22:17 2A425E19, 708992537
> .rsrc comp.: 2018–10–22 22:50:06 4D56B643, 1297528387
>
> File: C:\Intel\inte.exe
> PE Comp.: 1992–06–19 22:22:17 2A425E19, 708992537
> .rsrc comp.: 2018–10–22 22:49:20 4D56B62A, 1297528362

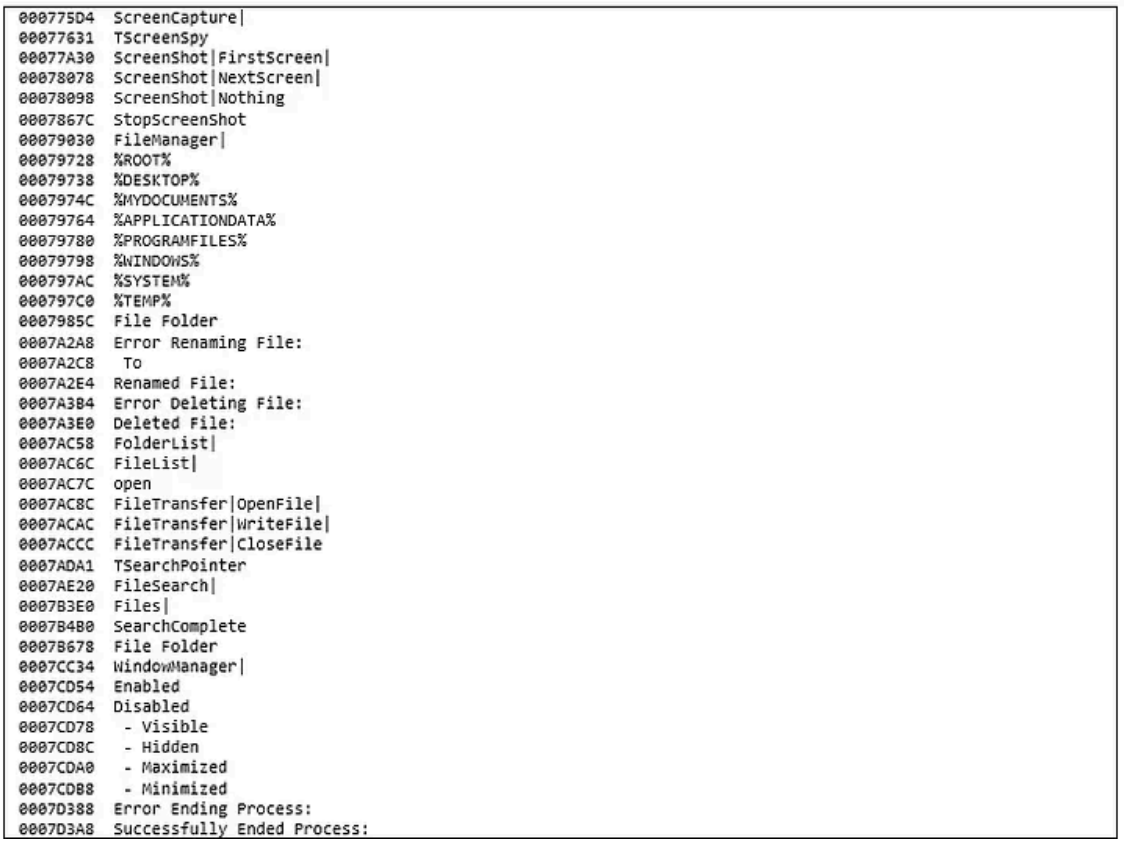
Judging from the compilation datetimes, they might be the output of a generator. Here, I could also make a guess at the chronological logic of when the files are prepared.



inte.exe

As with my usual style, I will start with a quick look at strings to try to guess the behaviour of the sample, before diving into dynamic and static analysis. Fortunately, the sample contains many helpful and descriptive strings that can help us deduce the features that this RAT provides, including Keylogger, FileManager, ProcessManager etc. Pretty typical RAT stuff.

Press enter or click to view image in full size



```
000775D4  ScreenCapture|
00077631  TScreenSpy
00077A30  ScreenShot|FirstScreen|
00078078  ScreenShot|NextScreen|
00078098  ScreenShot|Nothing
0007867C  StopScreenShot
00079030  FileManager|
00079728  %ROOT%
00079738  %DESKTOP%
0007974C  %MYDOCUMENTS%
00079764  %APPLICATIONDATA%
00079780  %PROGRAMFILES%
00079798  %WINDOWS%
000797AC  %SYSTEM%
000797C0  %TEMP%
0007985C  File Folder
0007A2A8  Error Renaming File:
0007A2C8   To
0007A2E4  Renamed File:
0007A3B4  Error Deleting File:
0007A3E0  Deleted File:
0007AC58  FolderList|
0007AC6C  FileList|
0007AC7C  open
0007AC8C  FileTransfer|OpenFile|
0007ACAC  FileTransfer|WriteFile|
0007ACCC  FileTransfer|CloseFile
0007ADA1  TSearchPointer
0007AE20  FileSearch|
0007B3E0  Files|
0007B4B0  SearchComplete
0007B678  File Folder
0007CC34  WindowManager|
0007CD54  Enabled
0007CD64  Disabled
0007CD78   - Visible
0007CD8C   - Hidden
0007CDA0   - Maximized
0007CDB8   - Minimized
0007D388  Error Ending Process:
0007D3A8  Successfully Ended Process:
```

Press enter or click to view image in full size

```
0007DB04  GetForeGroundWindow
0007DCC8  CreateToolhelp32Snapshot
0007E650  ProcessManager|
0007EFB0  Error Ending Process:
0007EFD0  Successfully Ended Process:
0007F69C  KeyLogger|
0007FBC8  <specialkey>[
0007FBE0  ]</specialkey>
0007FBF8  Clipboard|[
0007FC18  Caption|[
0007FC2C  SingleKey|
0007FD60  Start
0007FD70  Status|
0007FD88  Stop
0007FD98  Status|
0007FE5C  InstalledApplications|
00080280  DisplayName
00080294  SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\
000802EC  DisplayVersion
00080304  Publisher
00080318  UninstallString
00080330  QuietUninstallString
00080478  List
0008048C  software\microsoft\windows\currentversion\uninstall\
000804CC  Uninstall
000804DC  open
0008058C  RegEdit|
00080660  HKEY_CLASSES_ROOT
0008067C  HKEY_CURRENT_USER
00080698  HKEY_LOCAL_MACHINE
000806B4  HKEY_USERS
000806C8  HKEY_CURRENT_CONFIG
00080DF8  EnumValues|
00080ED8  RemoteShell|
000810D8  Command|
0008129C  SystemInfo|
00082054  IsNTAdmin
00082068  Administrator
00082080  Limited
000820A8  HARDWARE\DESCRIPTION\System\CentralProcessor\0\~MHz
00082188  tmp.htm
0011A53C  OnConnect|
0011A570  %ROOT%
0011A580  %DESKTOP%
0011A594  %MYDOCUMENTS%
0011A5AC  %APPLICATIONDATA%
0011AE88  SETTINGS
0011AEAC  exe.exe
0011AEBC  False
0011AECC  [Random-Number-Here]
0011AEEC  127.0.0.1:76
0011AF04  root
0011AF14  6000
0011AF30  True
0011AF40  Administrator
0011AFD0  112.213.107.134
0011B104  Idle
0011B2E0  C:\Intel\inte.exe
0011B45C  C:\Intel\inte.exe
0011B478  54623
```

A YARA rule hit

The sample happened to match James_inthe_box's YARA rule[2] on *BlackNix RAT*:

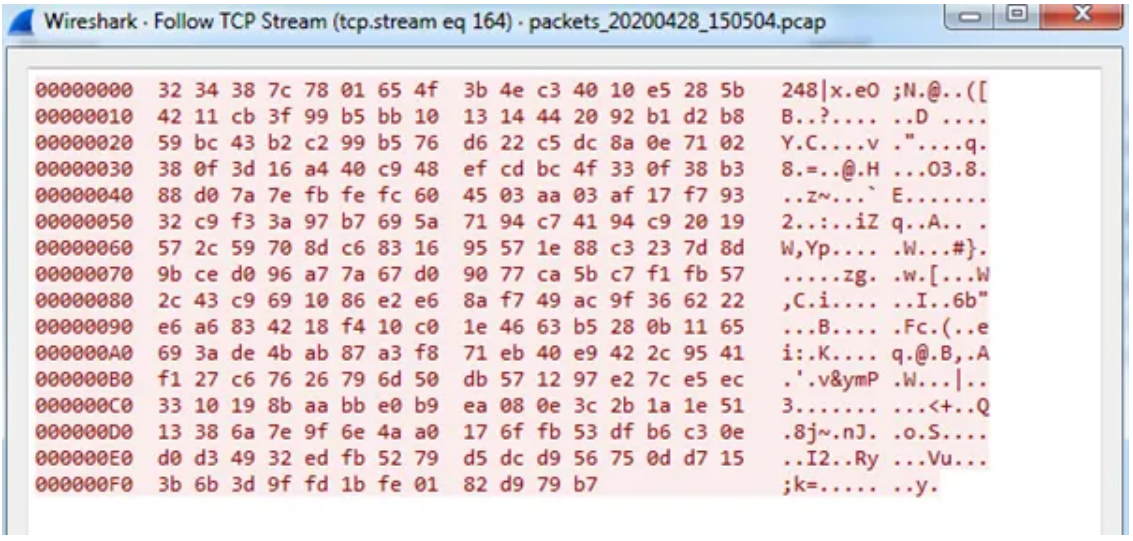
```
rule BlacknixRAT_bin{

meta:
description = "BlacknixRAT"
author = "James_inthe_box"
reference = "https://app.any.run/tasks/e3d845db-09b5-462d-8290-cbb4bb4a505f/"
date = "2019/02"
maltype = "RAT"

strings:
$string1 = "[Random-Number-Here]"
```

```
$string2 = “ScreenCapture”
$string3 = “TScreenSpy”
$string4 = “KeyLogger”
$string5 = “RemoteShell”

condition:
uint16(0) == 0x5A4D and all of ($string*) and filesize < 2000KB

}
```

Based on the strings seen above, the strings that matched the YARA rule did not look unique enough to confirm that this is indeed a *BlackNix* RAT. James_inthe_box also provided a snort rule:

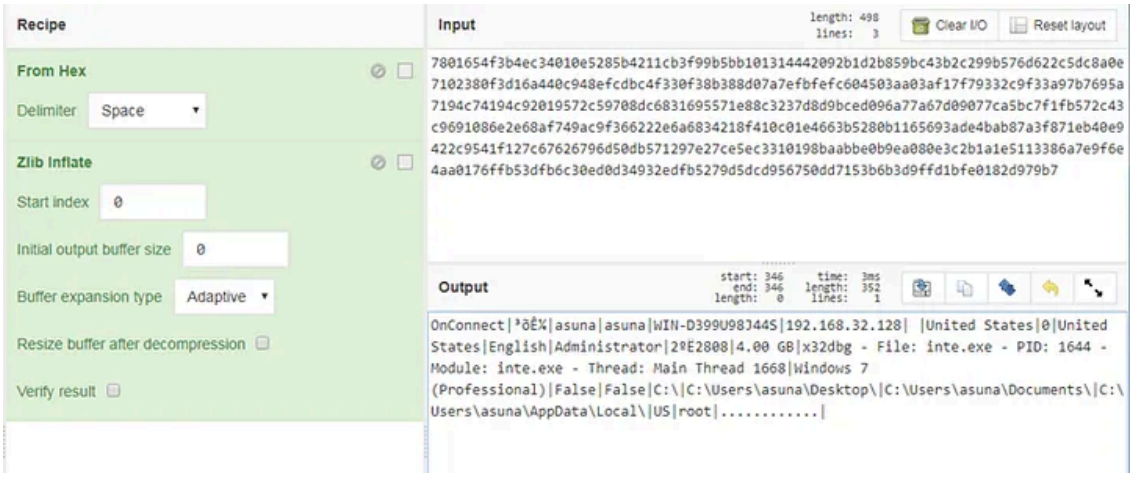
```
alert tcp any any -> any 80 (msg:”Blacknix RAT Detected”; flow:established,to_server; content:”|32|”; depth:1; content:”|7c 78 01 6d 8e|”; within:10; reference:url,https://app.any.run/tasks/e3d845db-09b5-462d-8290-cbb4bb4a505f/; classtype:trojan-activity; sid:20166298; rev:1; metadata:created_at 2019_07_18;
```

Let’s see what we have in the network data.



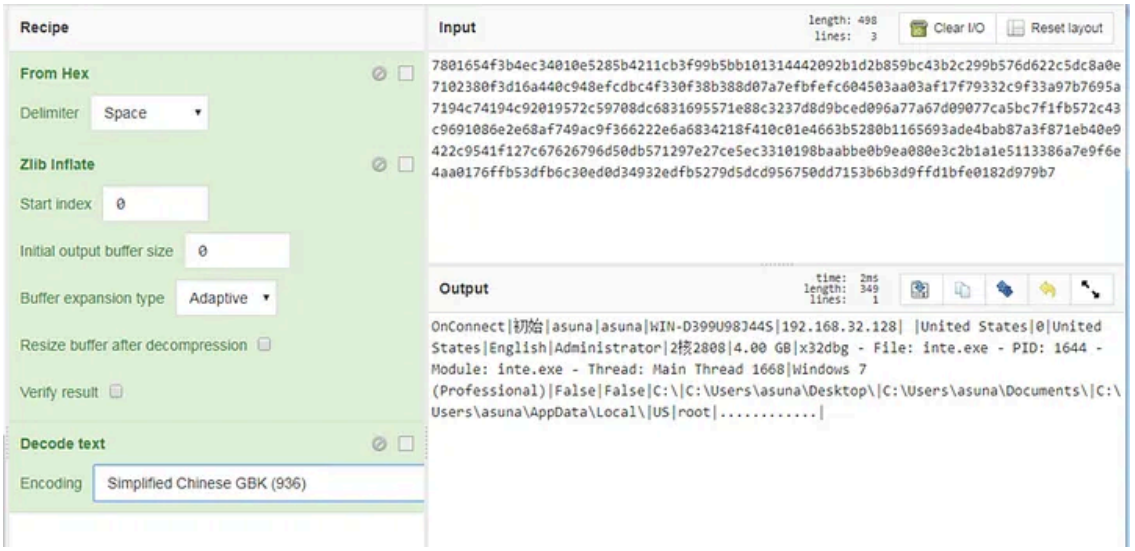
Yup, I see a 7C 78 01 but that’s not an exact match with the pattern in the snort rule. Hang on a second.. 78 01 looks like the ZLIB magic header. There we go!

Press enter or click to view image in full size



I noticed some weird characters, which could be indicative of Unicode (maybe Chinese…?). Let's try.

Press enter or click to view image in full size

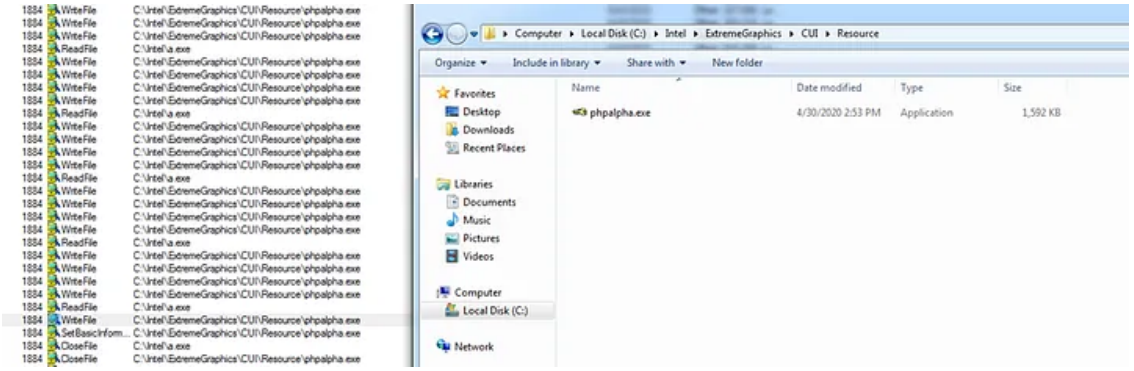


Indeed! 初始 is translated to mean Initial Start, and "2核2808" might mean 2 Cores (probably referring to the CPU cores). I'll step through the code that forms up this data in abit. Now, let us try to confirm if this callback belongs to *BlackNix* family.

The YARA and snort rules mentioned above referenced a sample (SHA256: A4DA694DED531EC60CA5A242C554B6A7062E12FF633D34656C4CA9DF86E42DD5). Let's sidetrack and see what this sample does.

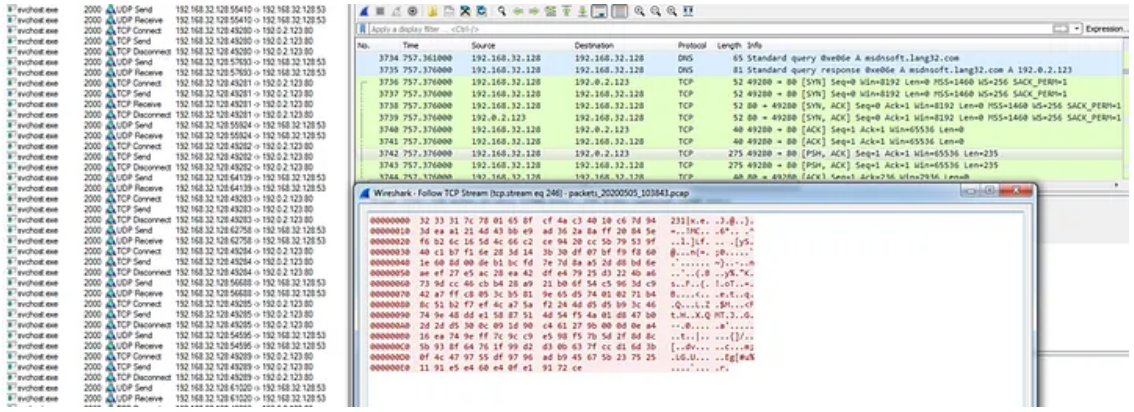
This sample is packed with VMProtect, so to save time, I'm just going to execute it and see what happens. Upon execution, a new file phpalpha.exe is created in C:\Intel\ExtremeGraphics\CUI\Resource. Turns out the file has the same hash as the parent binary.

Press enter or click to view image in full size



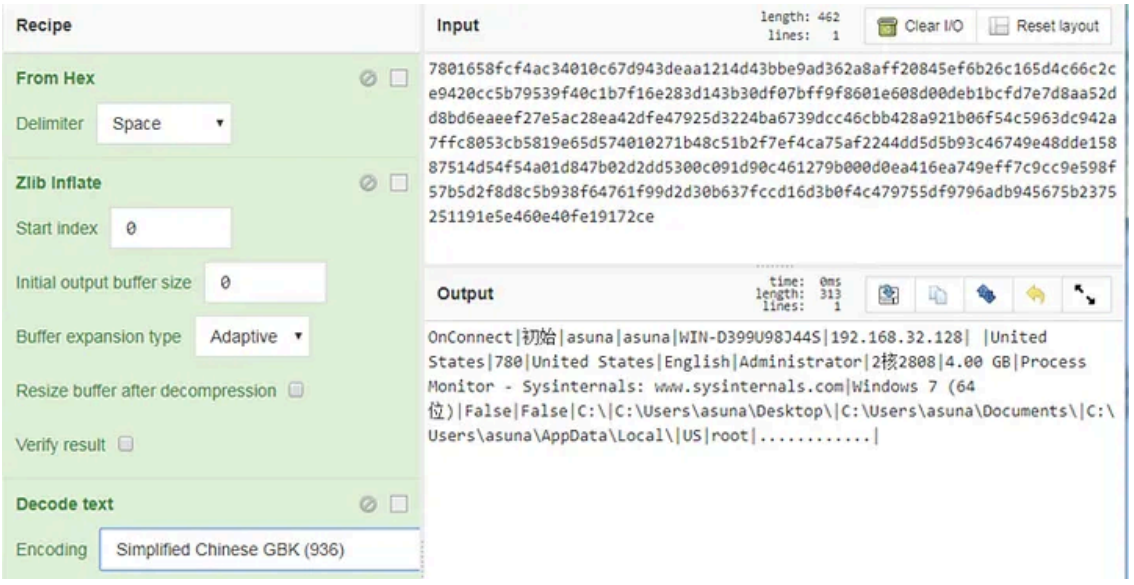
The network callback looks like this:

Press enter or click to view image in full size



We already know that's a ZLIB header, so let's use CyberChef to view the inflated data:

Press enter or click to view image in full size



Interesting. The data structure and keywords are identical. Now I can say that the sample (SHA256: A4DA694DED531EC60CA5A242C554B6A7062E12FF633D34656C4CA9DF86E42DD5) and our sample

(inte.exe, SHA256: F46520C2284E20C42AFA6E9B90E380735BFDF29817828369D5F1270A887E6979) belong to the same family. Is it really *BlackNix* though?

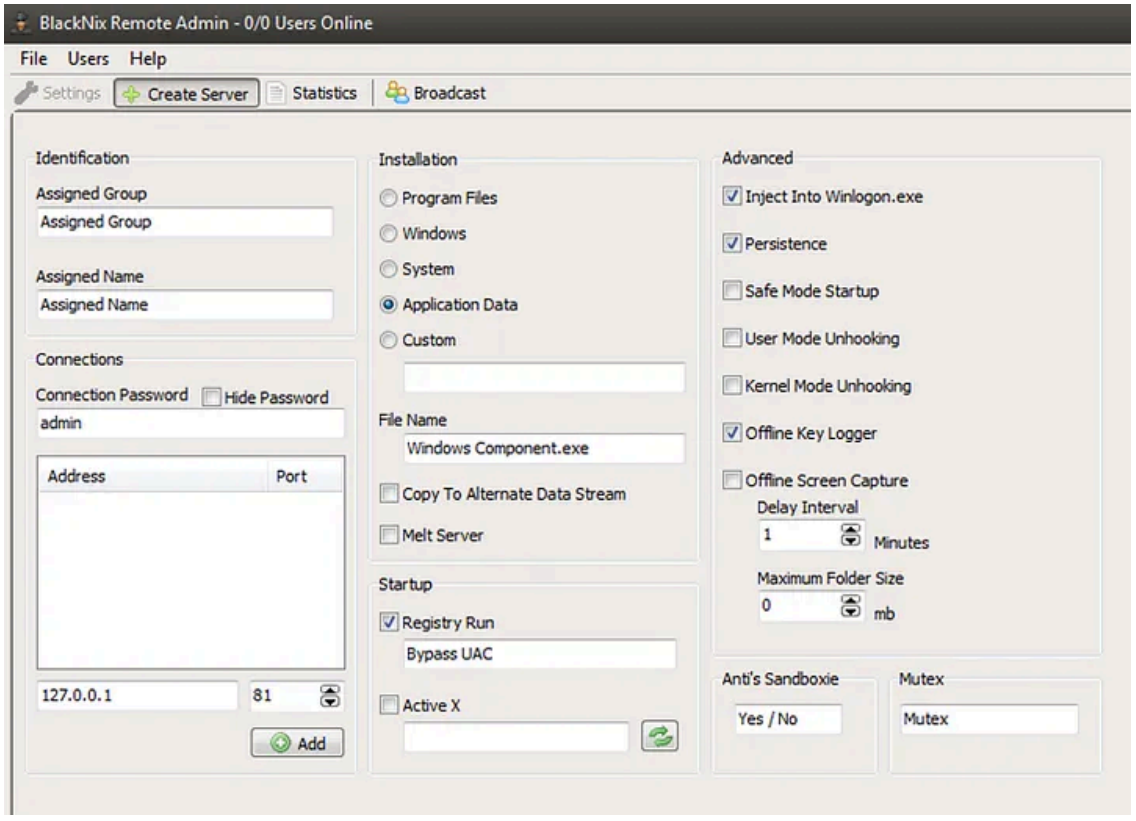
With the C2, everything is easier

With some help from Google search, I found a copy of a *BlackNix* C2 component :D

The C2 executable comes with the ability to generate the "server" component. This naming convention is common in RATs, where the malware client is typically referred to as the "server", and the C2 is the "client".

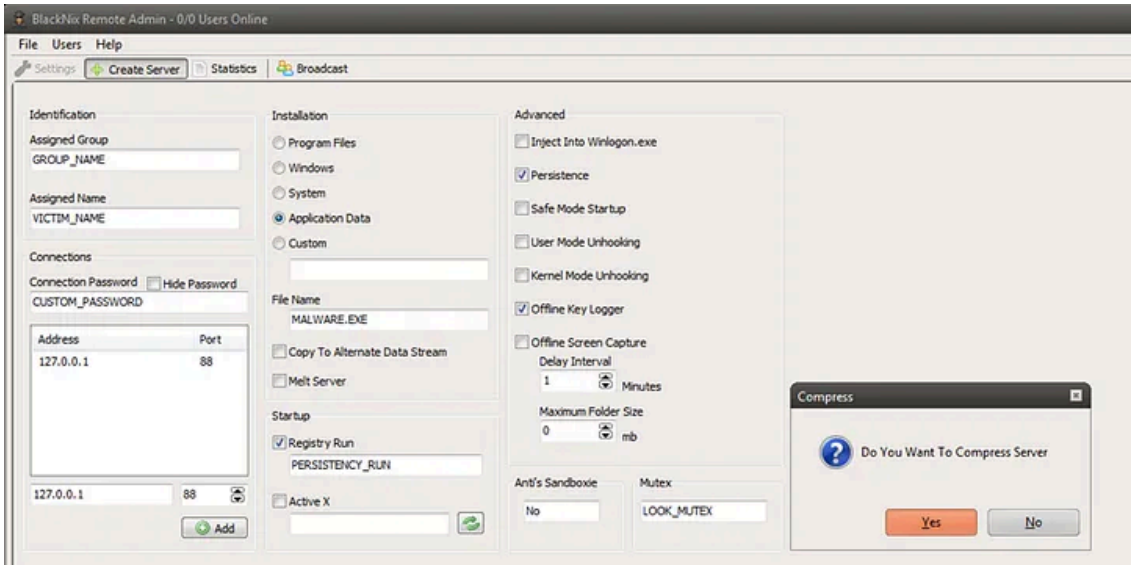
The following is one of the default profiles loaded with the C2:

Press enter or click to view image in full size

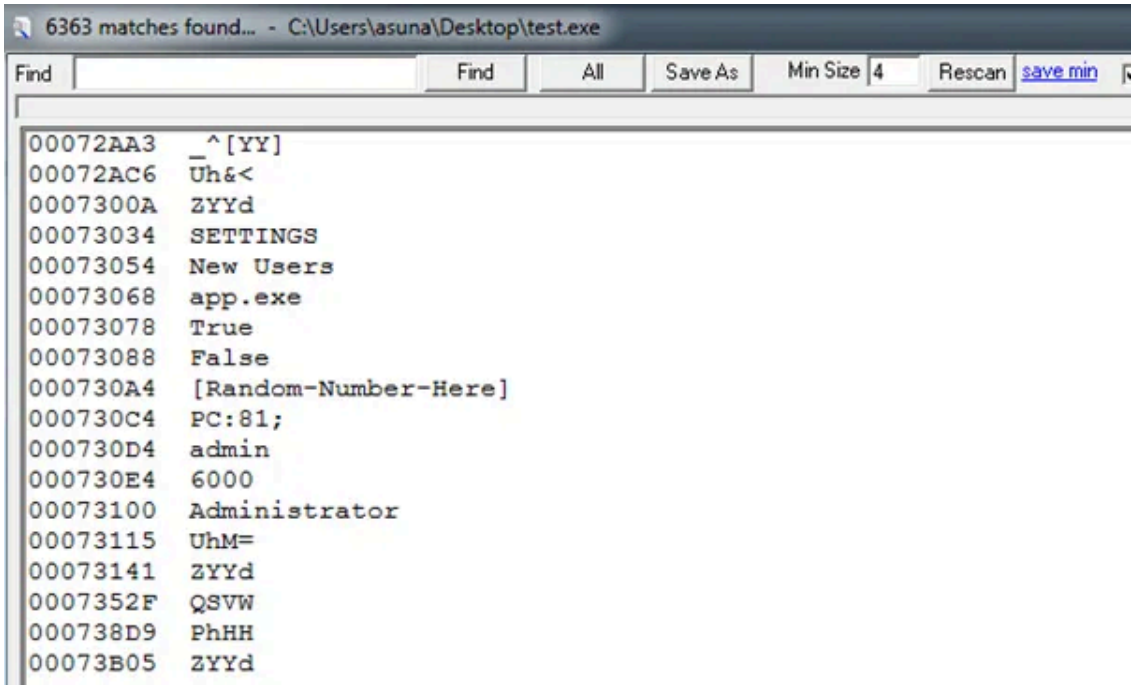


For the ease of testing, I changed some of the values when generating our test binary. The generator even comes with the option of UPX-packing the generated binary if the user wishes.

Press enter or click to view image in full size



Inspecting the strings within the generated binary, we can quickly identify some familiar keywords.



I've written a quick script to read the strings from the default settings. This will come useful later, when comparing these strings across different samples.

Press enter or click to view image in full size

```python
from binascii import unhexlify
from struct import unpack
import argparse

def extract_default(filename):
    with open(filename,"rb") as blacknix_input:
        file_contents = blacknix_input.read()
        setting_offset = file_contents.find("SETTINGS")
        print "SETTINGS found at {}".format(setting_offset)
        setting_offset +=12
        seek_pt = file_contents[setting_offset:setting_offset+256]
        next_offset = seek_pt.find(unhexlify(r"ffffffff"))
        while next_offset >= 0:
            next_offset += 4
            str_size = unpack("i",seek_pt[next_offset:next_offset+4])[0]
            next_offset += 4
            string = unpack(str(str_size)+"s", seek_pt[next_offset:next_offset+str_size])[0]
            next_offset += str_size
            print "{}".format(string)
            seek_pt = seek_pt[next_offset:]
            next_offset = seek_pt.find(unhexlify(r"ffffffff"))

def main():
    parser = argparse.ArgumentParser(description='Extract BlackNix RAT Default Settings')
    parser.add_argument('filename')
    args = parser.parse_args()
    extract_default(args.filename)


if __name__ == "__main__":
    main()
```

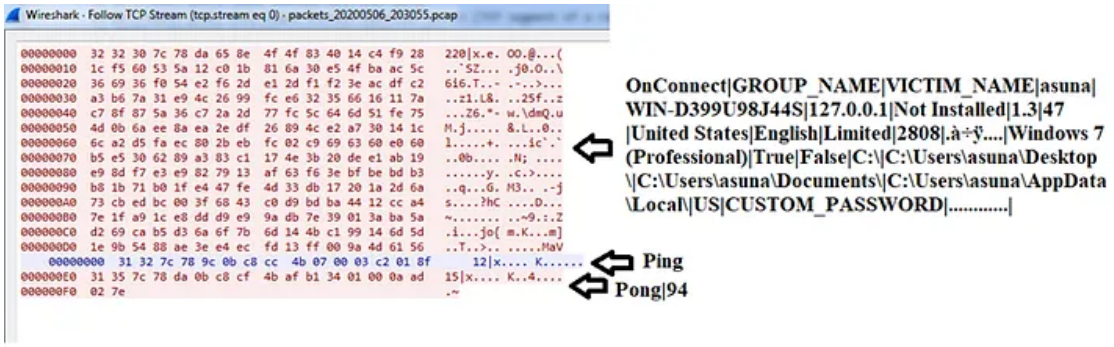
Here’s the output of running the script:

```
C:\Users\asuna\Desktop
λ python "blacknix_extract_default_settings.py" test.exe
SETTINGS found at 471092
|
New Users
app.exe
True
False
app
[Random-Number-Here]
PC:81;
admin
6000
0
Administrator
```

It appears that these SETTINGS strings have nothing to do with the configuration set when generating the binary. The default connection password within the C2 is “admin” and notice that even if I changed the password when generating the binary, the new password does not get inserted into this SETTINGS data. These may be part of a “stub” that comes with the C2 executable and inserted into every generated binary. I think this may be a helpful piece of information when trying to identify if a set of *BlackNix RAT*s is communicating with the same C2 executable (or at least the same version).

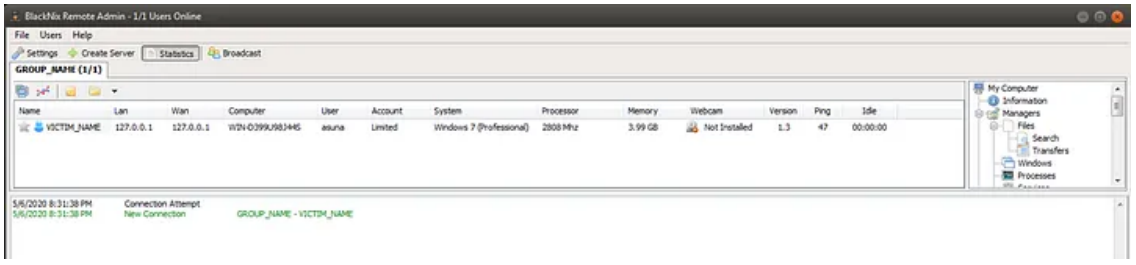
Take a look at the network communications.

Press enter or click to view image in full size



First thing that I noticed was the difference in the way the “Processor” information is being formatted. Remember there were some Chinese words (2核2808) that I thought refers to Processor Cores? In the data sent from this test binary, the processor information was simply a “2808” (referring to 2808 Mhz, which is indeed the setup of my VM).

Press enter or click to view image in full size



Connected victim on C2 dashboard

Let’s get back to the sample we have at hand.

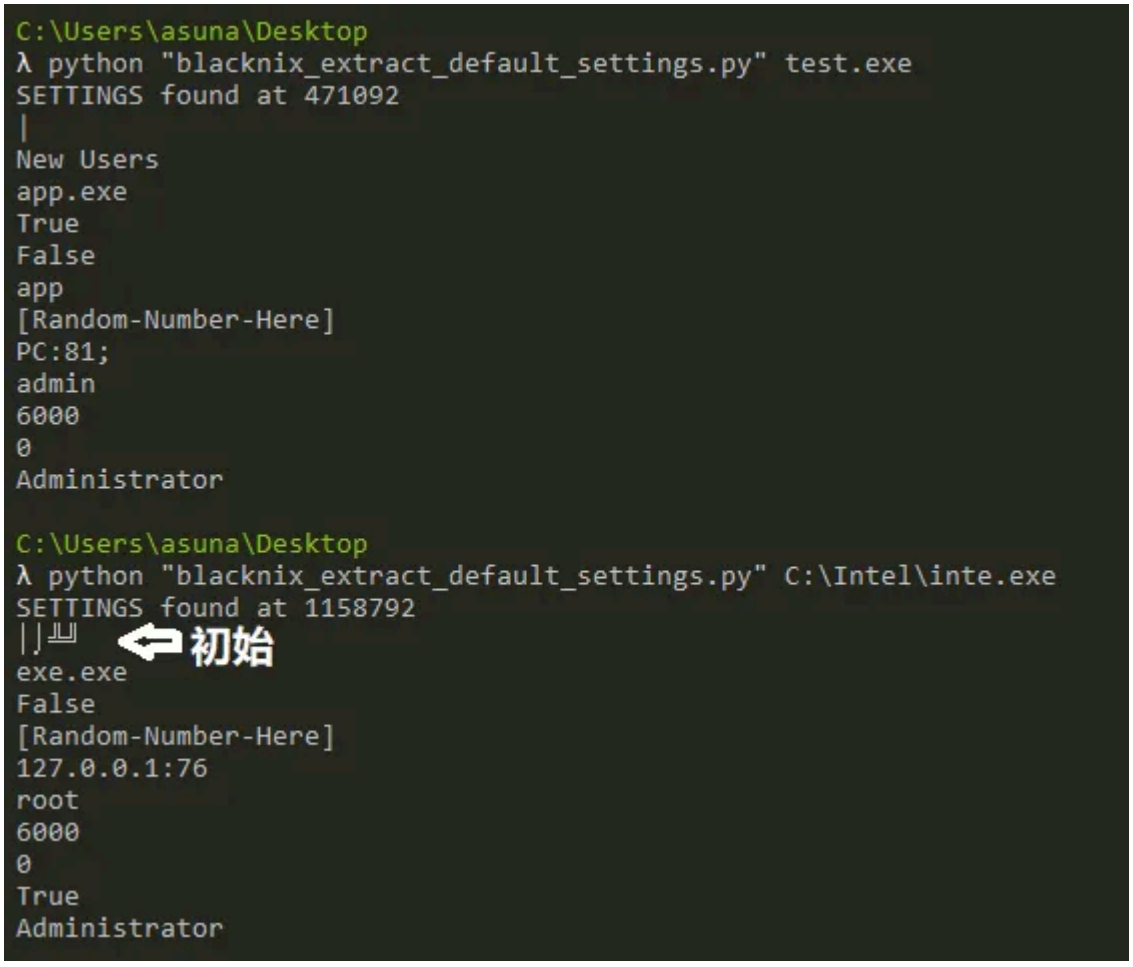
I did an in-depth analysis of how the first beacon’s data structure is formed within inte.exe.

Earlier I mentioned some SETTINGS strings. These are the strings that are populated into a data structure and some of these values are later copied into the first beacon data.

```
0051B8B5 BA 9C BA 51 00          mov     edx, offset a3oe ; 初始
0051B8BA E8 F5 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B8BF 8D 45 9C                lea     eax, [ebp+var_64]
0051B8C2 E8 15 66 F6 FF          call    getusername_481EDC
0051B8C7 8B 55 9C                mov     edx, [ebp+var_64]
0051B8CA 8D 43 04                lea     eax, [ebx+4]
0051B8CD E8 E2 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B8D2 8D 45 98                lea     eax, [ebp+var_68]
0051B8D5 E8 AA FB FF FF          call    sub_51B484
0051B8DA 8B 55 98                mov     edx, [ebp+var_68]
0051B8DD 8D 43 08                lea     eax, [ebx+8]
0051B8E0 E8 CF 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B8E5 8D 43 0C                lea     eax, [ebx+0Ch]
0051B8E8 BA AC BA 51 00          mov     edx, offset aExeExe ; "exe.exe"
0051B8ED E8 C2 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B8F2 8D 43 34                lea     eax, [ebx+34h]
0051B8F5 BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B8FA E8 B5 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B8FF 8D 43 38                lea     eax, [ebx+38h]
0051B902 BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B907 E8 A8 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B90C 8D 43 14                lea     eax, [ebx+14h]
0051B90F BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B914 E8 9B 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B919 8D 43 18                lea     eax, [ebx+18h]
0051B91C BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B921 E8 8E 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B926 8D 43 1C                lea     eax, [ebx+1Ch]
0051B929 BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B92E E8 81 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B933 8D 43 24                lea     eax, [ebx+24h]
0051B936 BA CC BA 51 00          mov     edx, offset aRandomNumberHe ; "[Random-Number-Here]"
0051B93B E8 74 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B940 8D 43 10                lea     eax, [ebx+10h]
0051B943 BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B948 E8 67 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B94D 8D 43 30                lea     eax, [ebx+30h]
0051B950 BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B955 E8 5A 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B95A 8D 43 28                lea     eax, [ebx+28h]
0051B95D BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B962 E8 4D 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B967 8D 43 3C                lea     eax, [ebx+3Ch]
0051B96A BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B96F E8 40 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B974 8D 43 40                lea     eax, [ebx+40h]
0051B977 BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B97C E8 33 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B981 8D 43 44                lea     eax, [ebx+44h]
0051B984 BA BC BA 51 00          mov     edx, offset aFalse ; "False"
0051B989 E8 26 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B98E 8D 43 2C                lea     eax, [ebx+2Ch]
0051B991 BA EC BA 51 00          mov     edx, offset a12700176 ; "127.0.0.1:76"
0051B996 E8 19 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
0051B99B 8D 43 48                lea     eax, [ebx+48h]
0051B99E BA 04 BB 51 00          mov     edx, offset aRoot ; "root"
0051B9A3 E8 0C 96 EE FF          call    System::__linkproc__ LStrAsg(void *,void *)
```

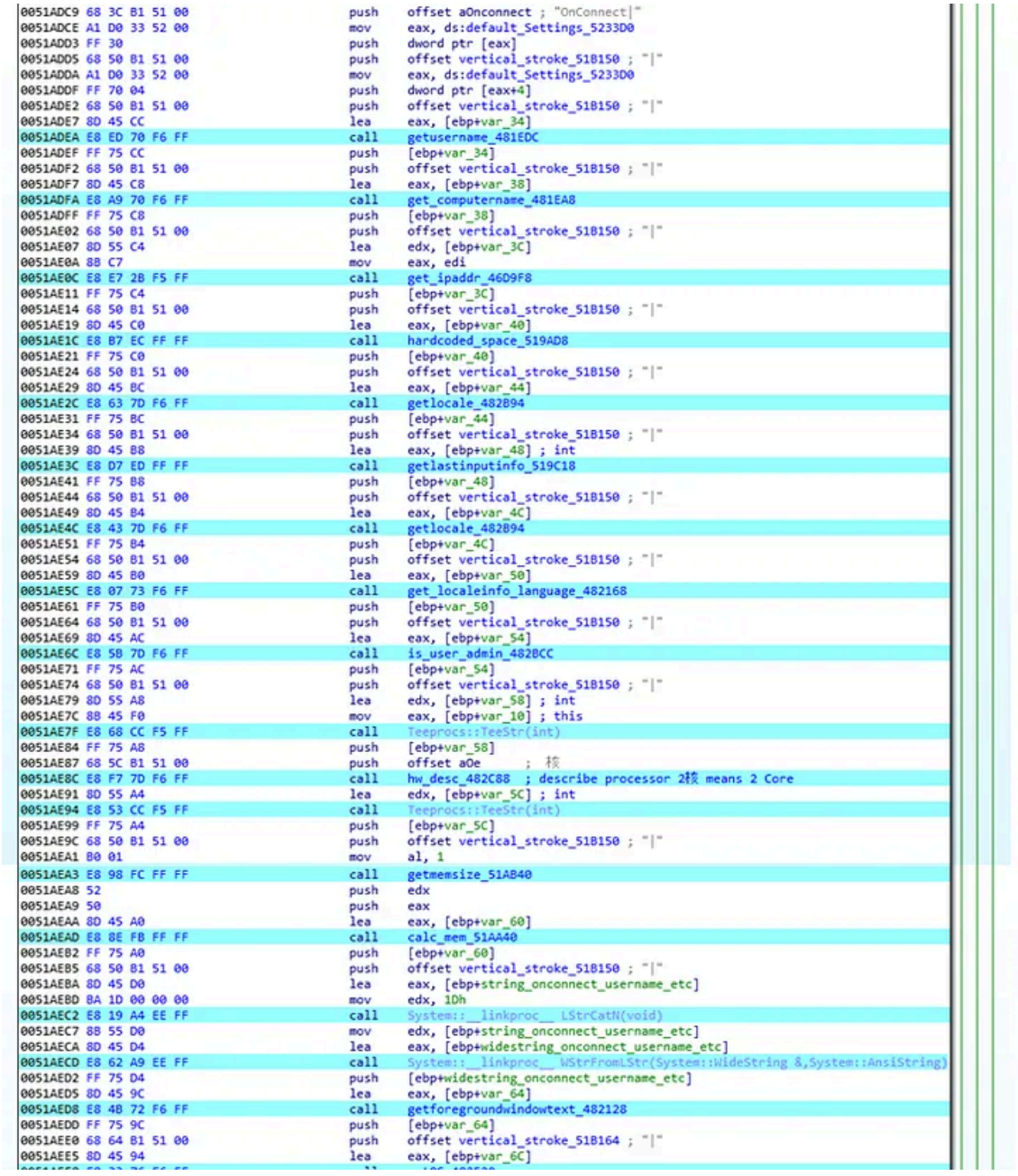
Reading strings from SETTINGS

Let’s compare the default SETTINGS strings found in the generated *BlackNix* binary and inte.exe.



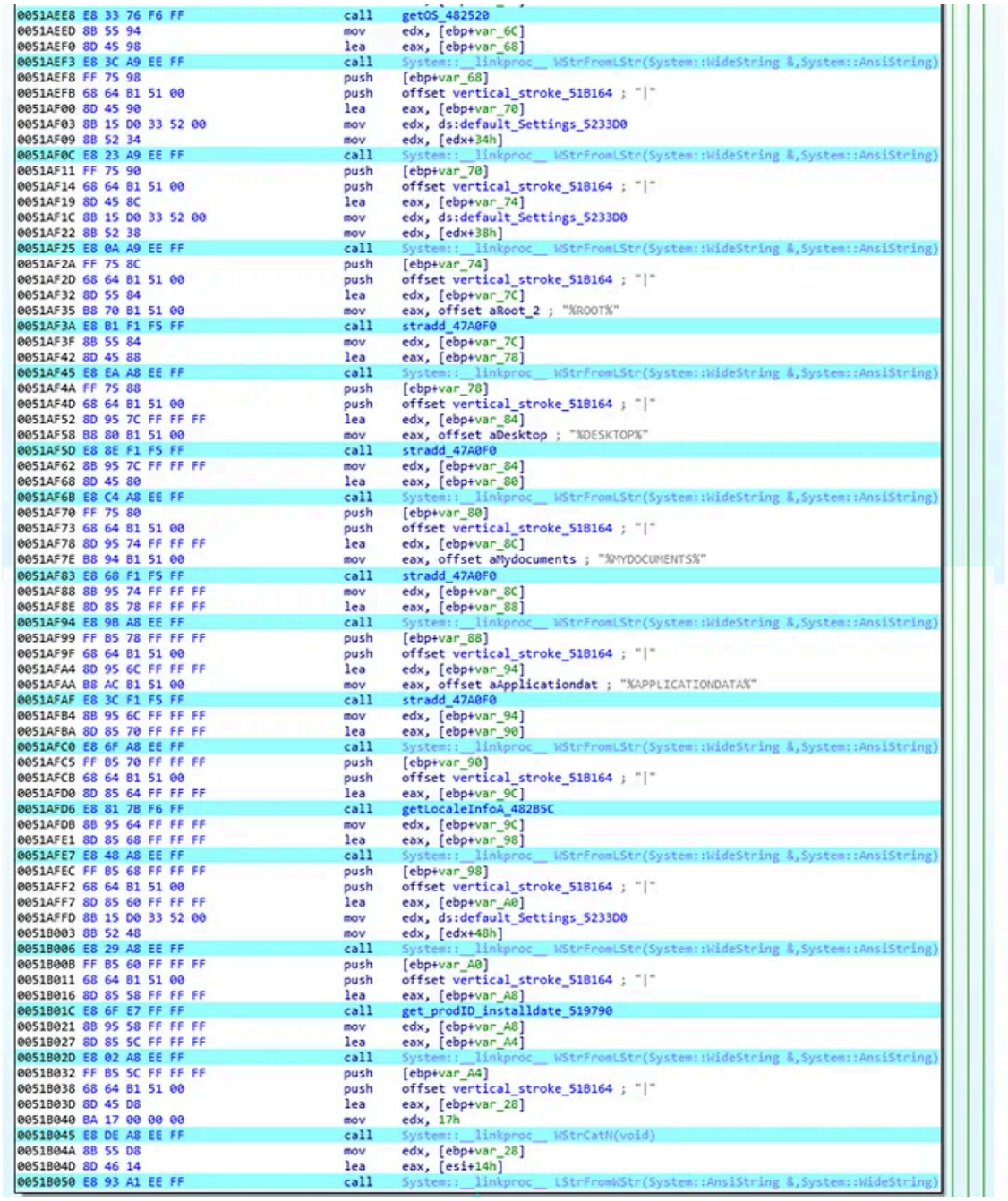
The following code is responsible for building the structure to be sent in the first beacon:

Press enter or click to view image in full size



```
0051ADC9 68 3C B1 51 00          push    offset aOnconnect ; "OnConnect|"
0051ADCE A1 D0 33 52 00          mov     eax, ds:default_Settings_5233D0
0051ADD3 FF 30                   push    dword ptr [eax]
0051ADD5 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051ADDA A1 D0 33 52 00          mov     eax, ds:default_Settings_5233D0
0051ADDF FF 70 04                push    dword ptr [eax+4]
0051ADE2 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051ADE7 8D 45 CC                lea     eax, [ebp+var_34]
0051ADEA E8 ED 70 F6 FF          call    getusername_481EDC
0051ADEF FF 75 CC                push    [ebp+var_34]
0051ADF2 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051ADF7 8D 45 C8                lea     eax, [ebp+var_38]
0051ADFA E8 A9 70 F6 FF          call    get_computername_481EA8
0051ADFF FF 75 C8                push    [ebp+var_38]
0051AE02 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AE07 8D 55 C4                lea     edx, [ebp+var_3C]
0051AE0A 8B C7                   mov     eax, edi
0051AE0C E8 E7 2B F5 FF          call    get_ipaddr_46D9F8
0051AE11 FF 75 C4                push    [ebp+var_3C]
0051AE14 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AE19 8D 45 C0                lea     eax, [ebp+var_40]
0051AE1C E8 B7 EC FF FF          call    hardcoded_space_519AD8
0051AE21 FF 75 C0                push    [ebp+var_40]
0051AE24 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AE29 8D 45 BC                lea     eax, [ebp+var_44]
0051AE2C E8 63 7D F6 FF          call    getlocale_482B94
0051AE31 FF 75 BC                push    [ebp+var_44]
0051AE34 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AE39 8D 45 B8                lea     eax, [ebp+var_48] ; int
0051AE3C E8 D7 ED FF FF          call    getlastinputinfo_519C18
0051AE41 FF 75 B8                push    [ebp+var_48]
0051AE44 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AE49 8D 45 B4                lea     eax, [ebp+var_4C]
0051AE4C E8 43 7D F6 FF          call    getlocale_482B94
0051AE51 FF 75 B4                push    [ebp+var_4C]
0051AE54 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AE59 8D 45 B0                lea     eax, [ebp+var_50]
0051AE5C E8 07 73 F6 FF          call    get_localeinfo_language_482168
0051AE61 FF 75 B0                push    [ebp+var_50]
0051AE64 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AE69 8D 45 AC                lea     eax, [ebp+var_54]
0051AE6C E8 5B 7D F6 FF          call    is_user_admin_482BCC
0051AE71 FF 75 AC                push    [ebp+var_54]
0051AE74 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AE79 8D 55 A8                lea     edx, [ebp+var_58] ; int
0051AE7C 8B 45 F0                mov     eax, [ebp+var_10] ; this
0051AE7F E8 68 CC F5 FF          call    Teeprocs::TeeStr(int)
0051AE84 FF 75 A8                push    [ebp+var_58]
0051AE87 68 5C B1 51 00          push    offset aOe      ; 核
0051AE8C E8 F7 7D F6 FF          call    hw_desc_482C88  ; describe processor 2核 means 2 Core
0051AE91 8D 55 A4                lea     edx, [ebp+var_5C] ; int
0051AE94 E8 53 CC F5 FF          call    Teeprocs::TeeStr(int)
0051AE99 FF 75 A4                push    [ebp+var_5C]
0051AE9C 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AEA1 B0 01                   mov     al, 1
0051AEA3 E8 98 FC FF FF          call    getmemsize_51AB40
0051AEA8 52                      push    edx
0051AEA9 50                      push    eax
0051AEAA 8D 45 A0                lea     eax, [ebp+var_60]
0051AEAD E8 8E FB FF FF          call    calc_mem_51AA40
0051AEB2 FF 75 A0                push    [ebp+var_60]
0051AEB5 68 50 B1 51 00          push    offset vertical_stroke_51B150 ; "|"
0051AEBA 8D 45 D0                lea     eax, [ebp+string_onconnect_username_etc]
0051AEBD BA 1D 00 00 00          mov     edx, 1Dh
0051AEC2 E8 19 A4 EE FF          call    System::__linkproc__ LStrCatN(void)
0051AEC7 8B 55 D0                mov     edx, [ebp+string_onconnect_username_etc]
0051AECA 8D 45 D4                lea     eax, [ebp+widestring_onconnect_username_etc]
0051AECD E8 62 A9 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051AED2 FF 75 D4                push    [ebp+widestring_onconnect_username_etc]
0051AED5 8D 45 9C                lea     eax, [ebp+var_64]
0051AED8 E8 4B 72 F6 FF          call    getforegroundwindowtext_482128
0051AEDD FF 75 9C                push    [ebp+var_64]
0051AEE0 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051AEE5 8D 45 94                lea     eax, [ebp+var_6C]
```

Press enter or click to view image in full size



```
0051AEE8 E8 33 76 F6 FF          call    getOS_482520
0051AEED 8B 55 94                mov     edx, [ebp+var_6C]
0051AEF0 8D 45 98                lea     eax, [ebp+var_68]
0051AEF3 E8 3C A9 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051AEF8 FF 75 98                push    [ebp+var_68]
0051AEFB 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051AF00 8D 45 90                lea     eax, [ebp+var_70]
0051AF03 8B 15 D0 33 52 00       mov     edx, ds:default_Settings_5233D0
0051AF09 8B 52 34                mov     edx, [edx+34h]
0051AF0C E8 23 A9 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051AF11 FF 75 90                push    [ebp+var_70]
0051AF14 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051AF19 8D 45 8C                lea     eax, [ebp+var_74]
0051AF1C 8B 15 D0 33 52 00       mov     edx, ds:default_Settings_5233D0
0051AF22 8B 52 38                mov     edx, [edx+38h]
0051AF25 E8 0A A9 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051AF2A FF 75 8C                push    [ebp+var_74]
0051AF2D 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051AF32 8D 55 84                lea     edx, [ebp+var_7C]
0051AF35 B8 70 B1 51 00          mov     eax, offset aRoot_2 ; "%ROOT%"
0051AF3A E8 B1 F1 F5 FF          call    stradd_47A0F0
0051AF3F 8B 55 84                mov     edx, [ebp+var_7C]
0051AF42 8D 45 88                lea     eax, [ebp+var_78]
0051AF45 E8 EA A8 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051AF4A FF 75 88                push    [ebp+var_78]
0051AF4D 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051AF52 8D 95 7C FF FF FF       lea     edx, [ebp+var_84]
0051AF58 B8 80 B1 51 00          mov     eax, offset aDesktop ; "%DESKTOP%"
0051AF5D E8 8E F1 F5 FF          call    stradd_47A0F0
0051AF62 8B 95 7C FF FF FF       mov     edx, [ebp+var_84]
0051AF68 8D 45 80                lea     eax, [ebp+var_80]
0051AF6B E8 C4 A8 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051AF70 FF 75 80                push    [ebp+var_80]
0051AF73 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051AF78 8D 95 74 FF FF FF       lea     edx, [ebp+var_8C]
0051AF7E B8 94 B1 51 00          mov     eax, offset aMydocuments ; "%MYDOCUMENTS%"
0051AF83 E8 68 F1 F5 FF          call    stradd_47A0F0
0051AF88 8B 95 74 FF FF FF       mov     edx, [ebp+var_8C]
0051AF8E 8D 85 78 FF FF FF       lea     eax, [ebp+var_88]
0051AF94 E8 9B A8 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051AF99 FF B5 78 FF FF FF       push    [ebp+var_88]
0051AF9F 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051AFA4 8D 95 6C FF FF FF       lea     edx, [ebp+var_94]
0051AFAA B8 AC B1 51 00          mov     eax, offset aApplicationdat ; "%APPLICATIONDATA%"
0051AFAF E8 3C F1 F5 FF          call    stradd_47A0F0
0051AFB4 8B 95 6C FF FF FF       mov     edx, [ebp+var_94]
0051AFBA 8D 85 70 FF FF FF       lea     eax, [ebp+var_90]
0051AFC0 E8 6F A8 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051AFC5 FF B5 70 FF FF FF       push    [ebp+var_90]
0051AFCB 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051AFD0 8D 85 64 FF FF FF       lea     eax, [ebp+var_9C]
0051AFD6 E8 81 7B F6 FF          call    getLocaleInfoA_48285C
0051AFDB 8B 95 64 FF FF FF       mov     edx, [ebp+var_9C]
0051AFE1 8D 85 68 FF FF FF       lea     eax, [ebp+var_98]
0051AFE7 E8 48 A8 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051AFEC FF B5 68 FF FF FF       push    [ebp+var_98]
0051AFF2 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051AFF7 8D 85 60 FF FF FF       lea     eax, [ebp+var_A0]
0051AFFD 8B 15 D0 33 52 00       mov     edx, ds:default_Settings_5233D0
0051B003 8B 52 48                mov     edx, [edx+48h]
0051B006 E8 29 A8 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051B00B FF B5 60 FF FF FF       push    [ebp+var_A0]
0051B011 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051B016 8D 85 58 FF FF FF       lea     eax, [ebp+var_A8]
0051B01C E8 6F E7 FF FF          call    get_prodID_installdate_519790
0051B021 8B 95 58 FF FF FF       mov     edx, [ebp+var_A8]
0051B027 8D 85 5C FF FF FF       lea     eax, [ebp+var_A4]
0051B02D E8 02 A8 EE FF          call    System::__linkproc__ WStrFromLStr(System::WideString &,System::AnsiString)
0051B032 FF B5 5C FF FF FF       push    [ebp+var_A4]
0051B038 68 64 B1 51 00          push    offset vertical_stroke_51B164 ; "|"
0051B03D 8D 45 D8                lea     eax, [ebp+var_28]
0051B040 BA 17 00 00 00          mov     edx, 17h
0051B045 E8 DE A8 EE FF          call    System::__linkproc__ WStrCatN(void)
0051B04A 8B 55 D8                mov     edx, [ebp+var_28]
0051B04D 8D 46 14                lea     eax, [esi+14h]
0051B050 E8 93 A1 EE FF          call    System::__linkproc__ LStrFromWStr(System::AnsiString &,System::WideString)
```

The following is the deduced first beacon's data structure sent from the inte.exe to its C2.

> OnConnect|Default 初始|Username|Username|Computer Name|IP Address|Hardcoded Space|Locale|Is Machine Idle?|Locale|Language|Account Privilege|Processor|Memory|Foreground Window Text|OS|Default False|Default False|%Root%|%Desktop%|%MyDocuments%|%AppData%|Locale|Server authentication password|ProdID, InstallDate|

Now we can play spot-the-differences. We can guess what each of these fields mean by looking at what can be seen on the dashboard, without reverse engineering the binary.

The following is the deduced first beacon's data structure sent from the test *BlackNix* binary.

> OnConnect|Assigned Group|Assigned Name|Username|Computer Name|IP Address|Webcam Installed?|C2 Version|?|Locale|Language|Account Privilege|Processor|Memory|OS|True/False?|True/False?|%Root%|%Desktop%|%MyDocuments%|%AppData%|Locale|Server authentication password|?|

Press enter or click to view image in full size

generated BlackNix binary	inte.exe	
OnConnect	OnConnect	
Assigned Group	初始	
Assigned Name	Username	
Username	Username	
Computer Name	Computer Name	
IP Address	IP Address	
Webcam Installed?	Hardcoded space	<- different data represented in this field
C2 Version?	Locale	<- different data represented in this field
?	Is Machine Idle?	
Locale	Locale	
Language	Language	
Account Privilege	Account Privilege	
Processor	Processor	
Memory	Memory	
-	Foreground Window Text	<- this field only exists for inte.exe
OS	OS	
Boolean	Boolean	
Boolean	Boolean	
%Root%	%Root%	
%Desktop%	%Desktop%	
%MyDocuments%	%MyDocuments%	
%AppData%	%AppData%	
Locale	Locale	
Server auth password	Server auth password	
?	ProdID, InstallDate	

Comparison of fields within data structure sent to C2

Yes! inte.exe is a BlackNix RAT, but has a different/modified C2 component?

What I did above proved that the inte.exe sample is indeed a *BlackNix RAT*, judging from the highly similar data structure within the initial beacon and the similarities found within the executables. However, since some fields in the communicated data are interpreted differently, I'm guessing there is a customised C2 that the adversary is using. I am not even able to tell the version number of the C2 from the sample, perhaps it is not important for the adversary. However, the SETTINGS strings found within the samples could be a way for us to differentiate variants.

So far, I've walked through the analysis of this set of files:

· **1st level droppers** (Project1.exe)
daaa061c88b197fa92d9648306e79875e3a24f392550dacaabd22e5fdba53ebf

75dc821013fe92ef93cefa47d3fe83ad5ce90658e8ef01fcdb0b11652397abec

· **2nd level droppers** (diskshadow.exe)
f0311ede2dd5e752411bf181626e3cdb36737affe67ddeb8af028d0c44355886
c5bab78fca3db0ce5ffff5838a5a4a93d930e715ded1cbd8a5b3caf0cdce803c

· ***BlackNix RAT*** (inte.exe)
f46520c2284e20c42afa6e9b90e380735bfdf29817828369d5f1270a887e6979

I've verified that the sample inte.exe is indeed a *BlackNix RAT* and communicates with a custom *BlackNix* C2 at IP address 112.213.107[.]134.

Related to this IP address, other possible *BlackNix RAT* samples were found on VirusTotal:

Press enter or click to view image in full size

Set 1	7bd71729392ccb2edff561cd3cf65f5c2e4c40c14a309915b28c4a24bf3bf5f4 b77db93ad5561198a106d282a9f1fd36192b5eba00a75afa2b83ee1e1c536c8d ec016cb5d3365381f58529b796e43d61d30bc7ae09b22f4b9242a182df438a40 e9313bb16605dde4798bef57f47977c9f21aeda79c456a0624d4cd32e1f684d8 0b177acb2f53d230cdbf2cb8c3aa6fe315461934596aabd9e25fae037c35a9cf 8fdefc1b8869cc565a55e2e79d7dfc92f861895d80dfc1ff70af247fb969ec15

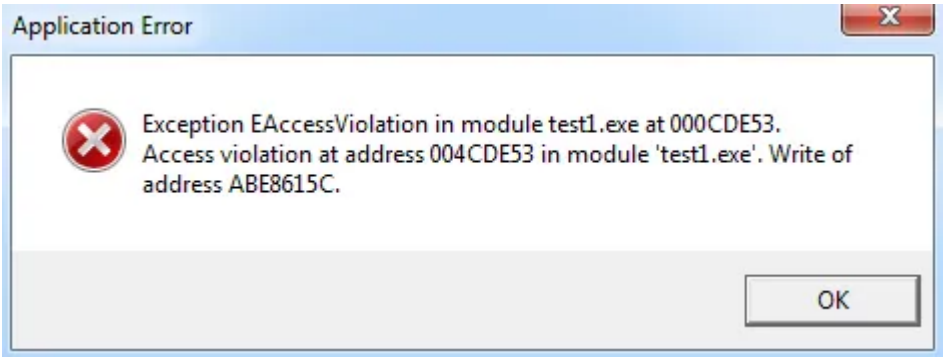
In addition, one other *BlackNix RAT* samples were mentioned by james_inthe_box[2]. Based on SSDEEP similarity, another sample was found.

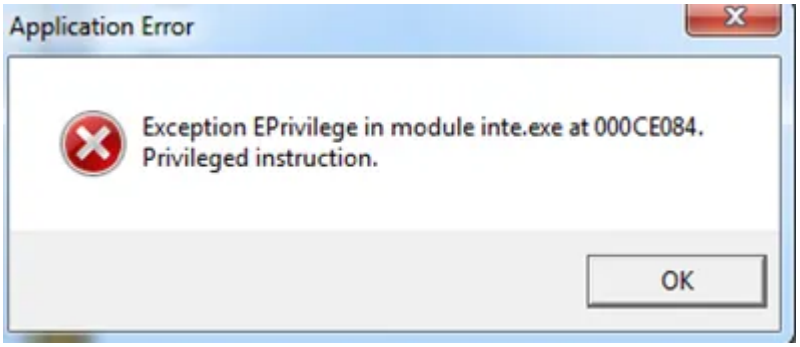
Press enter or click to view image in full size

Set 2	a4da694ded531ec60ca5a242c554b6a7062e12ff633d34656c4ca9df86e42dd5 873dfa94f924d59ceff4efb277fef5a251d7b648605c5239fc2ac0885ba32bd5

Next, we shall see if all these samples send data in the same structure as what we have analyzed previously. If they do, then perhaps these are all related in some way and are not "wild" *BlackNix RAT*s.

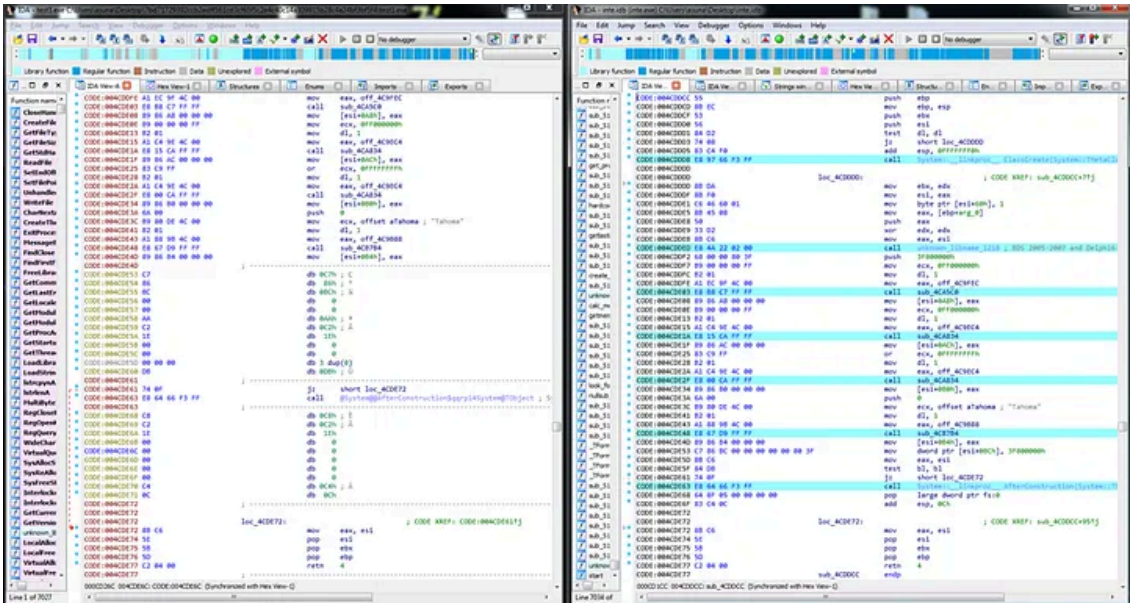
Set 1 binaries





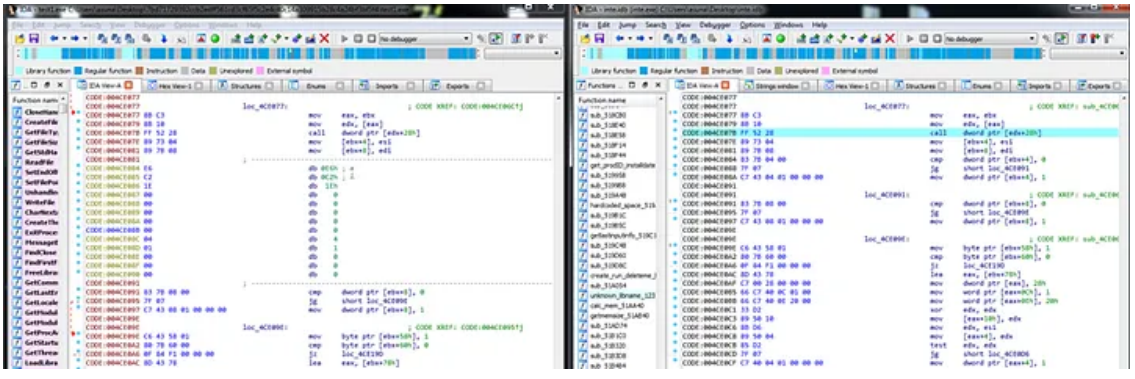
Execution of these binaries will result in errors. Upon closer look, the errors happen because some part of the binary seems to be corrupted. Whether this is a deliberate “disarm” attempt or due to a bug, I can’t tell.

Press enter or click to view image in full size



Cause of error at address 0x4CDE53

Press enter or click to view image in full size



Cause of error at address 0x4CE084

Just patch the areas with the corresponding bytes from inte.exe to fix the problems. The following screenshots show highlighted areas after patching.

```
         0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
C:D220h: 86 AC 00 00 00 83 C9 FF B2 01 A1 C4 9E 4C 00 E8  †¬...fÉÿ².¡ÄžL.è
C:D230h: 00 CA FF FF 89 86 B0 00 00 00 6A 00 B9 80 DE 4C  .Êÿÿ‰†°...j.¹€ÞL
C:D240h: 00 B2 01 A1 88 9B 4C 00 E8 67 D9 FF FF 89 86 B4  .².¡ˆ›L.ègÙÿÿ‰†´
C:D250h: 00 00 00 C7 86 BC 00 00 00 00 00 80 3F 8B C6 84  ...Ç†¼.....€?‹Æ„
C:D260h: DB 74 0F E8 64 66 F3 FF 64 8F 05 00 00 00 00 83  Ût.èdfóÿd.....ƒ
C:D270h: C4 0C 8B C6 5E 5B 5D C2 04 00 00 00 0C 00 00 00  Ä.‹Æ^[]Â........
C:D280h: 54 00 61 00 68 00 6F 00 6D 00 61 00 00 00 00 00  T.a.h.o.m.a.....
C:D290h: 55 8B EC 51 53 56 84 D2 74 08 83 C4 F0 E8 D2 65  U‹ìQSV„Òt.ƒÄðèÒe
```

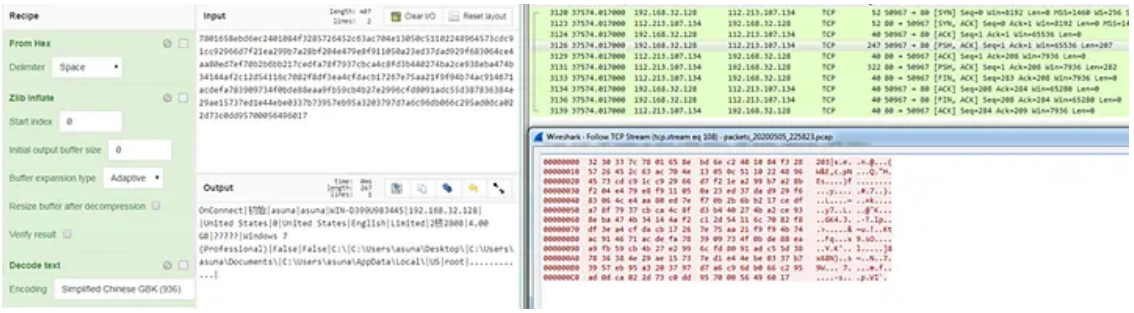
Patch to solve error at 0x4CDE53

```
         0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
C:D460h: 53 56 57 8B F9 8B F2 8B D8 3B 73 04 75 09 3B 7B  SVW‹ù‹ò‹Ø;s.u.;{
C:D470h: 08 0F 84 A2 01 00 00 8B C3 8B 10 FF 52 28 89 73  ..„¢...‹Ã‹.ÿR(‰s
C:D480h: 04 89 7B 08 83 7B 04 00 7F 07 C7 43 04 01 00 00  .‰{.ƒ{....ÇC....
C:D490h: 00 83 7B 08 00 7F 07 C7 43 08 01 00 00 00 C6 43  .ƒ{....ÇC.....ÆC
C:D4A0h: 58 01 80 7B 60 00 0F 84 F1 00 00 00 8D 43 78 C7  X.€{`..„ñ....CxÇ
C:D4B0h: 00 28 00 00 00 66 C7 40 0C 01 00 66 C7 40 0E 20  .(...fÇ@...fÇ@.
```

Patch to solve error at 0x4CE084

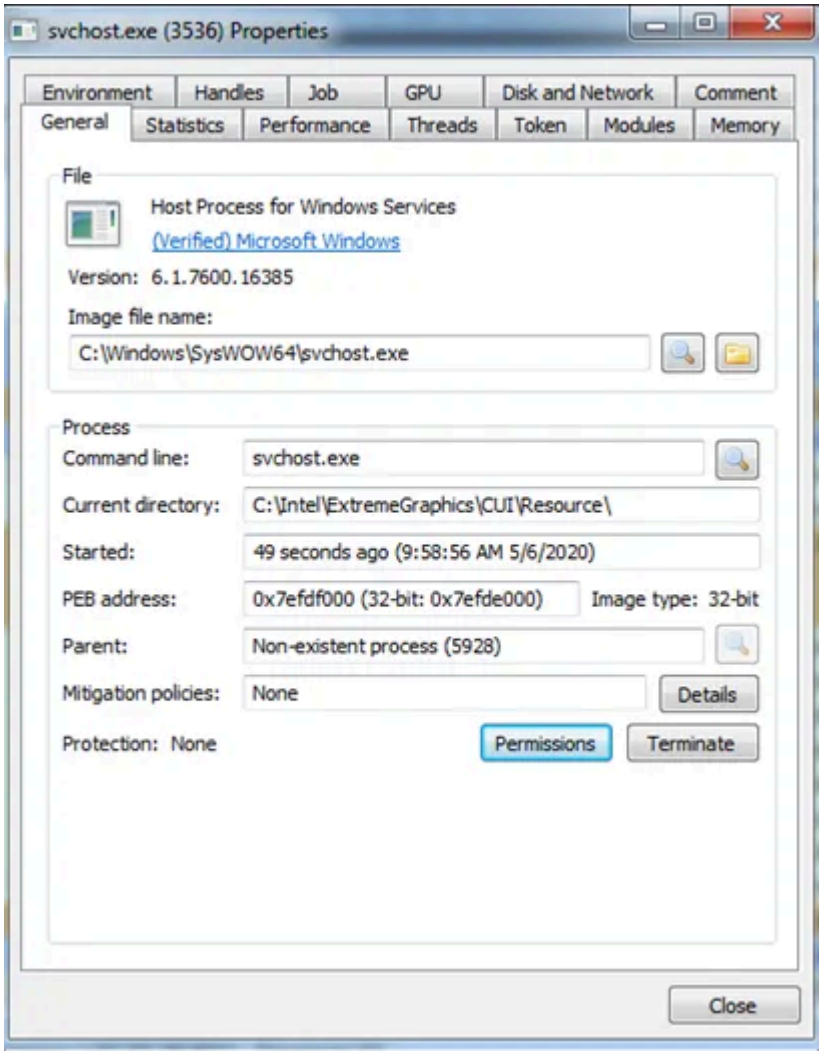
From the network packet, it looks like the data structures are identical to what we saw in inte.exe. This is verified with a comparison of the function that is responsible for building the structure. This is not surprising, as they all call back to the same IP address.

Press enter or click to view image in full size



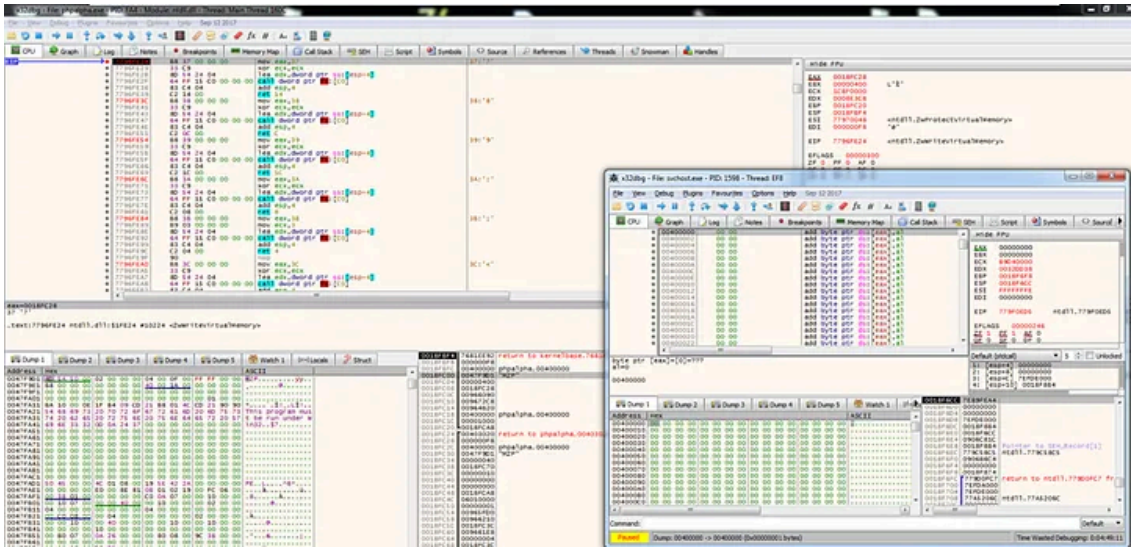
Set 2 binaries

These binaries are different, because they are VMProtect-packed, which means that I cannot simply throw them into IDA Pro and hope to do function comparisons.



Execution of these binaries require them to be executed with administrator privileges, as they will spawn a svchost.exe process for injection. Knowing this behaviour, we can dump the unpacked executable from memory at the moment where the injection happens. A breakpoint at ntdll.dll's NtWriteVirtualMemory will do the trick.

Press enter or click to view image in full size

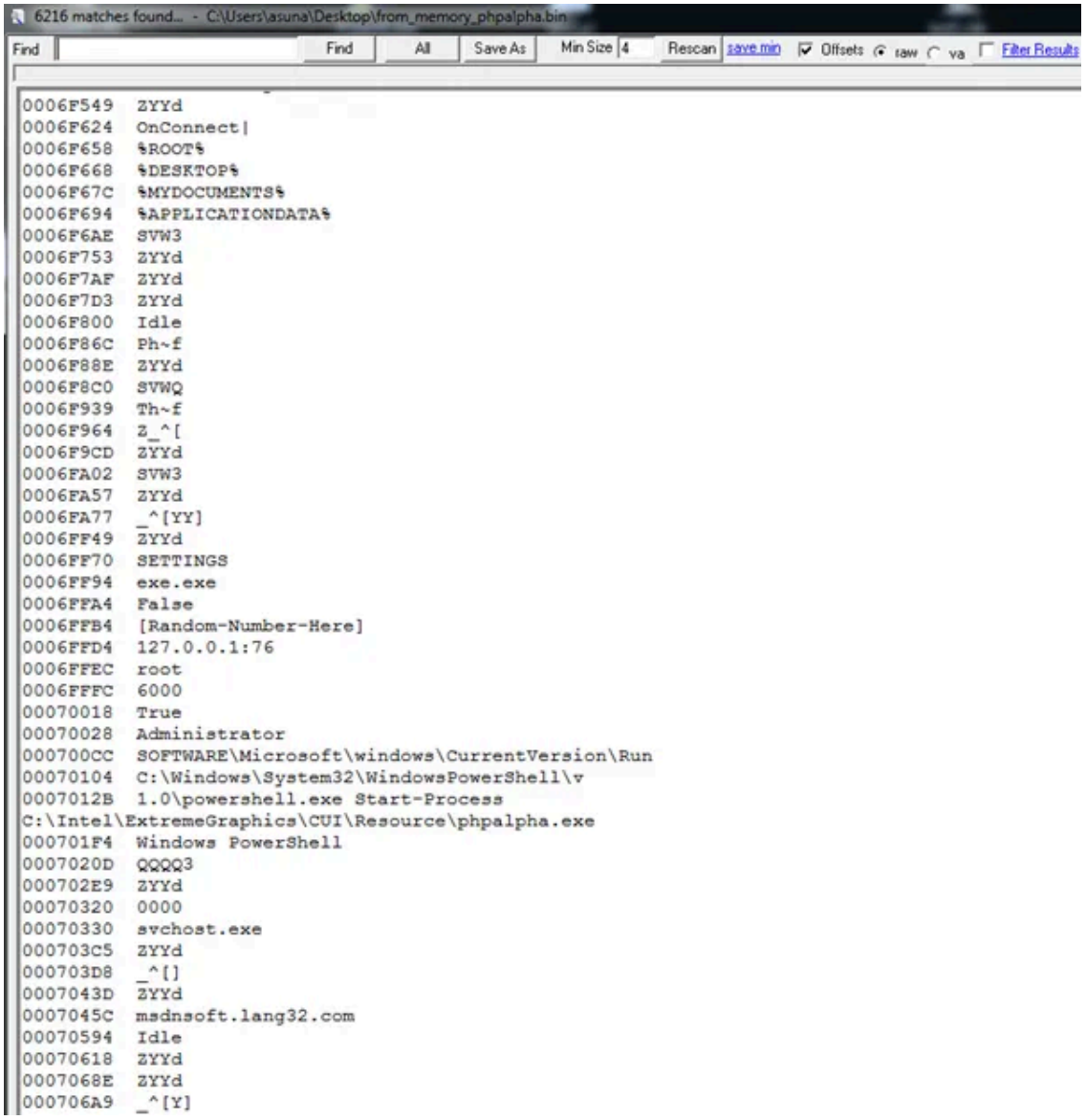


> NtWriteVirtualMemory(
> IN HANDLE ProcessHandle,
> IN PVOID BaseAddress,
> IN PVOID Buffer,
> IN ULONG NumberOfBytesToWrite,
> OUT PULONG NumberOfBytesWritten OPTIONAL);

The idea is to watch for a call to NtWriteVirtualMemory with a handle to svchost.exe, and let it run till all the sections have been copied. We would know it's done when NtResumeThread is called.

After dumping the executable from memory, we would have to fix the section headers' raw addresses before we can use IDA Pro to look at it. I've mentioned how to do this with CFF explorer in one of my earlier posts.

A quick look at strings within this dumped file reveals the tell-tale *BlackNix* strings:

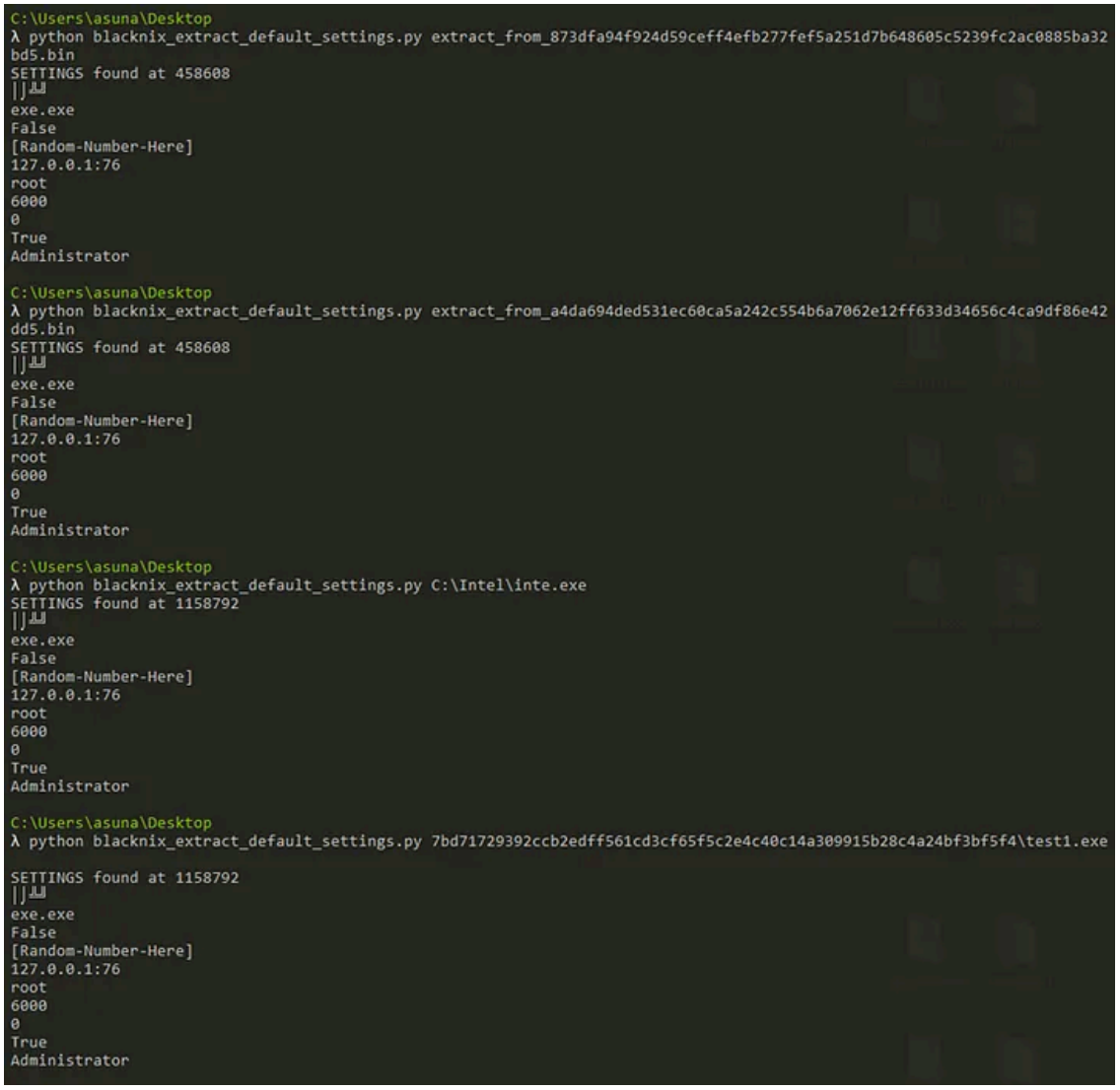


The SETTINGS strings looks identical to what was seen in inte.exe, including the Chinese words 初始 and the server password 'root'. The function that is responsible for reading the SETTINGS strings and building the

callback data structure is identical to inte.exe's as well (and hence the callback data structure is also the same).

I am certain that we are looking at the same variant of *BlackNix RAT* here.

Press enter or click to view image in full size



So, they are all the same BlackNix variant. Now what?

This journey started from some unique mutexes found in a malware (one *BBSRAT*) that calls back to one of known **Winnti Group**'s infrastructure. The same set of mutexes, some overlaps in code logic (in the naming of files and lateral movement using RDP shared drives), as well as close time proximity in compilation timestamps, suggested relationship between that one *BBSRAT* and the set of *BlackNix RAT*s (Project1.exe).

In addition to the mutexes, I noticed other similarities in Project1.exe's execution and the *Trochilus RAT* dropper csres.exe described in Trend Micro's Uncovering DRBControl report[3], specifically in the names of the files and service created and path to malicious binary:

- system.exe
- SESSRV
- c:\ProgramData\Microsoft\DeviceSync

I get reminded of my earlier speculation that system.exe is a generic tool used to deliver/spread the payload (be it *BlackNix* or *Trochilus RAT*). I'll never know for sure till I get my hands on some more samples ;)

Last Words

Is **Winnti Group** also behind the set of *BlackNix RAT*s that were under scrutiny in this post? There might be a good chance this is true. However, one other interesting finding that I came across was that the C2 domain msdnsoft[.]lang32[.]com as well as the corresponding binary (SHA256: 873dfa94f924d59ceff4efb277fef5a251d7b648605c5239fc2ac0885ba32bd5) were linked to an adversary group named "Lang32" by QiAnXin Technology[4]. This adversary group is said to target victims in Southeast Asia. Perhaps I should look into the tools used by this group as well…

But that's a different long story for another time, that's it for now!

References:

[1]: http://www.hexacorn.com/blog/2014/12/05/the-not-so-boring-land-of-borland-executables-part-1/
[2]: https://twitter.com/james_inthe_box/status/1151972438692921344
[3]: "Operation DRBControl: Uncovering a Cyberespionage Campaign Targeting Gambling Companies in Southeast Asia", Trend Micro, 18 Feb 2020
[4]: https://www.secrss.com/articles/12463

~~

Drop me a DM if you would like to share findings or samples ;)
