Practical Malware Analysis & Triage

Malware Analysis Report

Malware.cryptlib64.dll

Jan 2024 | NightNinja | v1.0

Table of Contents

Executive Summary

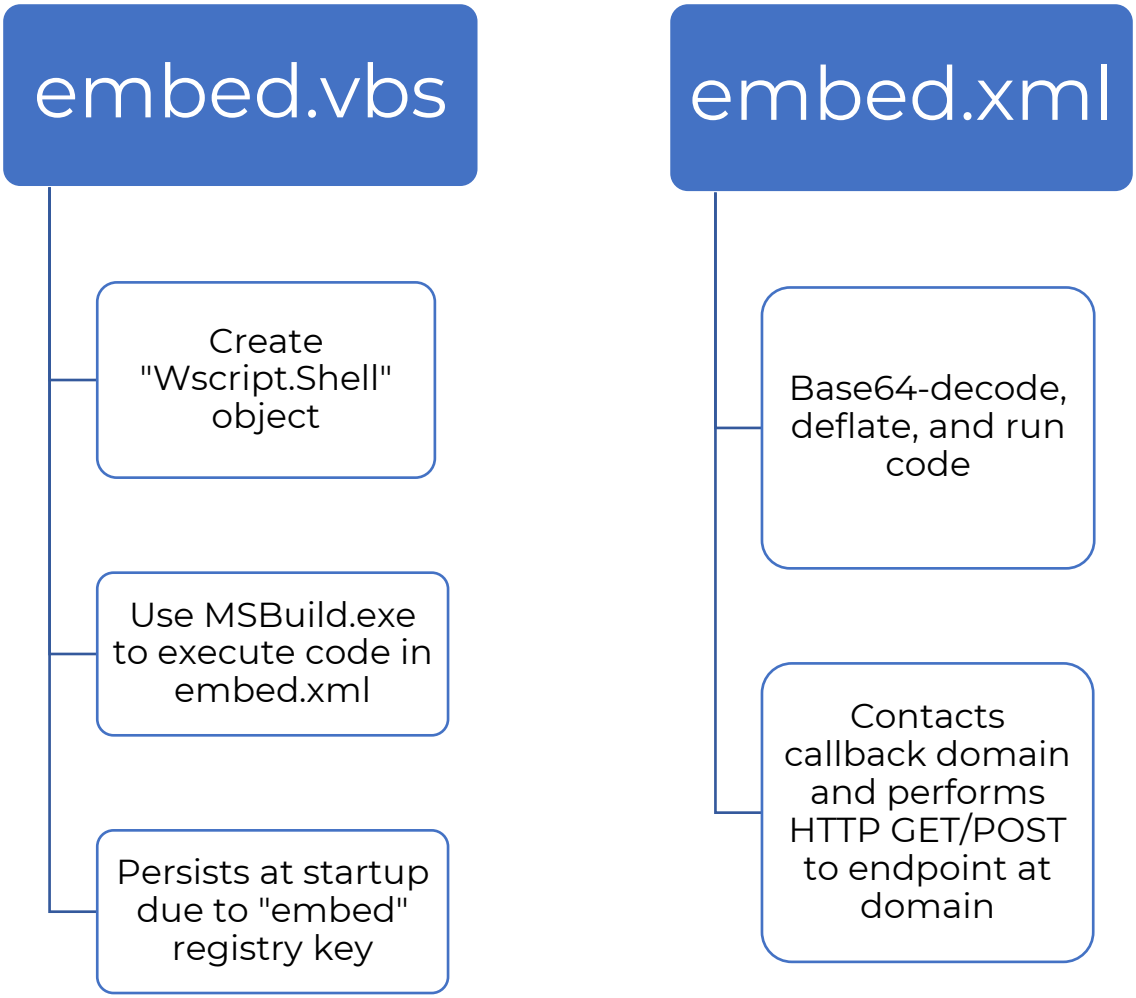
SHA256 hash	732f235784cd2a40c82847b4700fb73175221c6ae6c5f7200a3f43f209989387

Cryptlib64 is a 64-bit Windows malware sample compiled on Sun Oct 10 11:14:49 2021 (UTC-8). It is a C#-compiled dropper that runs on the x64 Windows operating system. It consists of two payloads that are executed in succession following a successful user login. Symptoms of infection include DNS callback and GET/POST requests to the URL listed in Appendix B, and files named “embed.xml” and “embed.vbs” appearing in the %public% directory.

YARA signature rules are attached in Appendix A. The malware sample and hashes have been submitted to VirusTotal for further examination.

High-Level Technical Summary

Cryptlib64 consists of few parts: upon DLL execution, two “embed” files are created in the %public% directory, and a registry key (named “embed”, value of %public\embed.vbs%) is also created. The embed.VBS file is used to create a shell object to invoke code in embed.XML (code loaded as reflection assembly) via MSBuild.exe. It first attempts to contact the callback URL of hxxps://srv[.]masterchiefsgruntemporium[.]local If successful, there’s an HTTP GET then POST to an endpoint at hxxps://srv[.]masterchiefsgruntemporium[.]local, which looked like a connection to a remote application (HTTP request have “Cookie: ASPSESSIONID=; SESSIONID=1552332971750” in header.)



Malware Composition

Cryptlib64 consists of the following components:

File Name	SHA256 Hash
embed.vbs	66fd543f31545082cf8fcc45a6ab1094bc118c45634f2be450f84f4e5745b291
embed.xml	f1548cd02784606c8abac865abf5ed6220d34eea88c7a5715e0183d7f050f4ab

embed.vbs:

This file performs code execution after a successful user login after initial infection. The file creates a “Wscript.Shell” object to call to “MSBuild.exe” to run “embed.xml”. Since “embed.vbs” is referenced in a registry key called ‘embed’ at HKCU\Software\Microsoft\Windows\CurrentVersion\Run, the infection will persist for every subsequent login until removed. Shell runs in a hidden prompt window.

embed.xml:

A Base64-encoded and compressed file containing the code to be run (loaded as reflection assembly) by MSBuild.exe. It attempts to contact the callback URL of hxxps://srv[.]masterchiefsgruntemporium[.]local If successful, there’s an HTTP GET then POST to an endpoint at hxxps://srv[.]masterchiefsgruntemporium[.]local, which looked like a connection to a remote application (HTTP request have “Cookie: ASPSESSIONID=; SESSIONID=1552332971750” in header.)

Basic Static Analysis

Compilation date: Sun Oct 10 11:14:49 2021 UTC-8
SHA256: 732f235784cd2a40c82847b4700fb73175221c6ae6c5f7200a3f43f209989387
MD5: 361e6edb47e711a72c7f8ee3c0c1632b
VT: Potentially malicious
First byte MZ (PE file)
64-bit architecture
Floss strings:

- CreateEncryptor/CreateDecryptor
- AES_Encrypt/AES_Decrypt
- mscorlib
- EmbedDLL
- p0w3r0verwh3lm1ng!
- Large blocks of Base64 text
- Registry stuff: RegistryKey, RegistryHive, RegistryView, OpenBaseKey, OpenSubKey
- System.Security.Cryptography (.NET namespace flagged)
- mscoree.dll (.NET Runtime Execution Engine)
- \EmbedDLL.dll
- C:\Users\Public\Documents\embed.vbs
- public, \embed.xml (on two seperate lines, maybe a file path?)
- Software\Microsoft\Windows\CurrentVersion\Run (registry entry?)

"Embed" files:

- embed.xml - SHA256: f1548cd02784606c8abac865abf5ed6220d34eea88c7a5715e0183d7f050f4ab, MD5: 000ff2048f094552db03e446b25d4441
- embed.vbs - SHA256: 66fd543f31545082cf8fcc45a6ab1094bc118c45634f2be450f84f4e5745b291, MD5: 2c7cae7ea80f8ad5eb4412df76615bc1

indicator (17)	detail	level
imports > flag > count	16	1
.NET > namespace > flag	System.Security.Cryptography	1
string > size > suspicious	10496 bytes	2
groups > API	cryptography, obfuscation, reconnaissance, file, memory, registry	2
mitre > technique	T1027, T1001, T1055, T1060	2
file > entropy	4.178	3
file > signature	Microsoft .NET	3
file > footprint	732F235784CD2A40C82847B4700FB73175221C6AE6C5F7200A3F43F20998...	3
file > size	29184 bytes	3
security > protection	data-execution-prevention (DEP) > ON	3
security > protection	control-flow-guard (CFG) > OFF	3
security > protection	address-space-layout-randomization (ASLR) > ON	3
file > subsystem	console	3
file-name > version	EmbedDLL.dll	3
imphash > md5	DAE02F32A21E03CE65412F6E56942DAA	3
file-name > exports	\EmbedDLL.dll	3
.NET > module > name	EmbedDLL.dll	3

PEStudio: summary of flagged items

encoding (2)	size (bytes)	location	flag (12)	label (161)	group (6)	technique (4)	value (281)
ascii	22	.text	x	import	reconnaissance	-	GetEnvironmentVariable
ascii	22	0x0CD80FA2	x	import	reconnaissance	-	GetEnvironmentVariable
ascii	16	.text	x	import	obfuscation	T1001 | Data Obfuscation	FromBase64String
ascii	16	0x0CD80F63	x	import	obfuscation	T1001 | Data Obfuscation	FromBase64String
ascii	12	.text	x	import	memory	T1055 | Process Injection	MemoryStream
ascii	12	0x0CD80D68	x	import	memory	T1055 | Process Injection	MemoryStream
ascii	15	.text	x	import	cryptography	T1001 | Data Obfuscation	CreateDecryptor
ascii	11	.text	x	import	cryptography	T1027 | Obfuscated Files or Information	AES_Encrypt
ascii	11	.text	x	import	cryptography	T1027 | Obfuscated Files or Information	AES_Decrypt
ascii	15	0x0CD80EF7	x	import	cryptography	T1001 | Data Obfuscation	CreateDecryptor
ascii	11	0x0CD810C0	x	import	cryptography	T1027 | Obfuscated Files or Information	AES_Encrypt
ascii	11	0x0CD810ED	x	import	cryptography	T1027 | Obfuscated Files or Information	AES_Decrypt

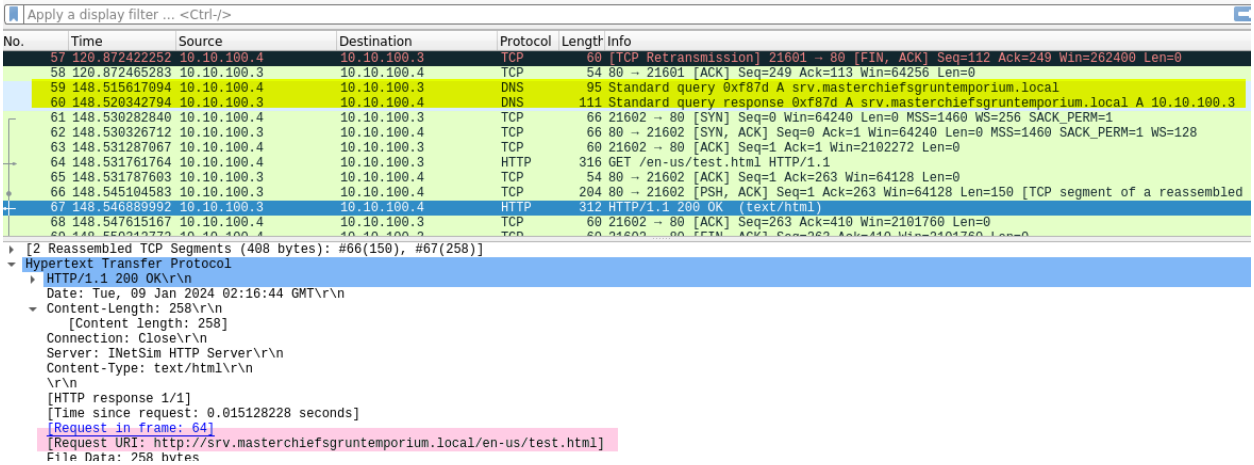
PEStudio: summary of flagged strings

imports (92)	namespace (10)	flag (16)	group (6)	technique (3)	type (2)	ordinal (0)	library (1)
GetEnvironmentVariable	-	x	reconnaissance	-	MemberRef	-	mscoree.dll
MemoryStream	System.IO	x	memory	T1055 | Process Injection	TypeRef	-	mscoree.dll
AES_Encrypt	-	x	cryptography	T1027 | Obfuscated Files or Information	Method	-	mscoree.dll
AES_Decrypt	-	x	cryptography	T1027 | Obfuscated Files or Information	Method	-	mscoree.dll
CreateDecryptor	-	x	cryptography	T1001 | Data Obfuscation	MemberRef	-	mscoree.dll
RijndaelManaged	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
Rfc2898DeriveBytes	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
CryptoStream	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
SymmetricAlgorithm	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
DeriveBytes	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
CipherMode	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
ICryptoTransform	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
CryptoStreamMode	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
PaddingMode	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
SHA256	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll
HashAlgorithm	System.Security.Cryptograp...	x	cryptography	T1001 | Data Obfuscation	TypeRef	-	mscoree.dll

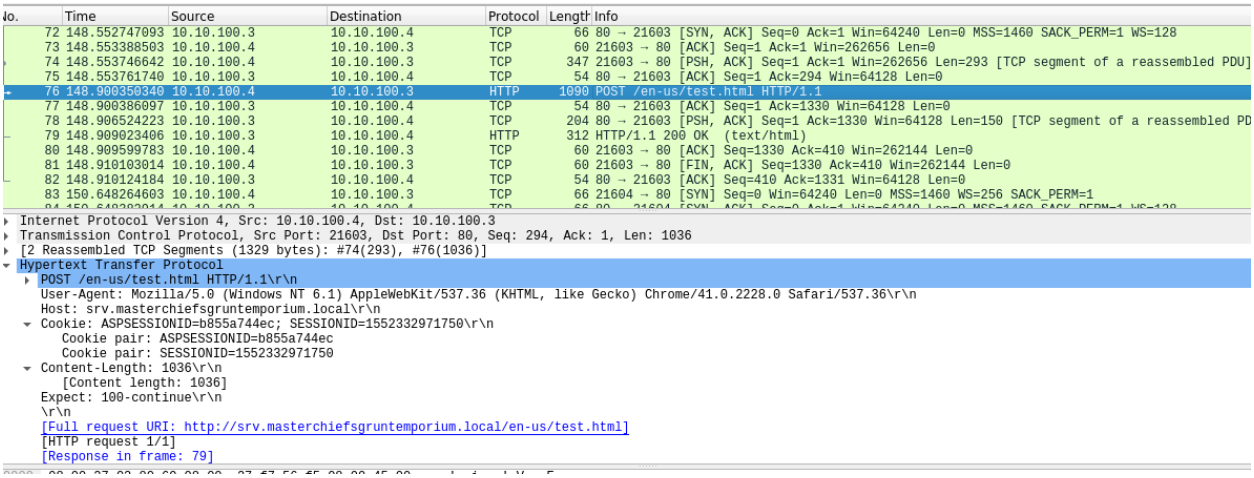
PEStudio: summary of flagged C# imports

Basic Dynamic Analysis

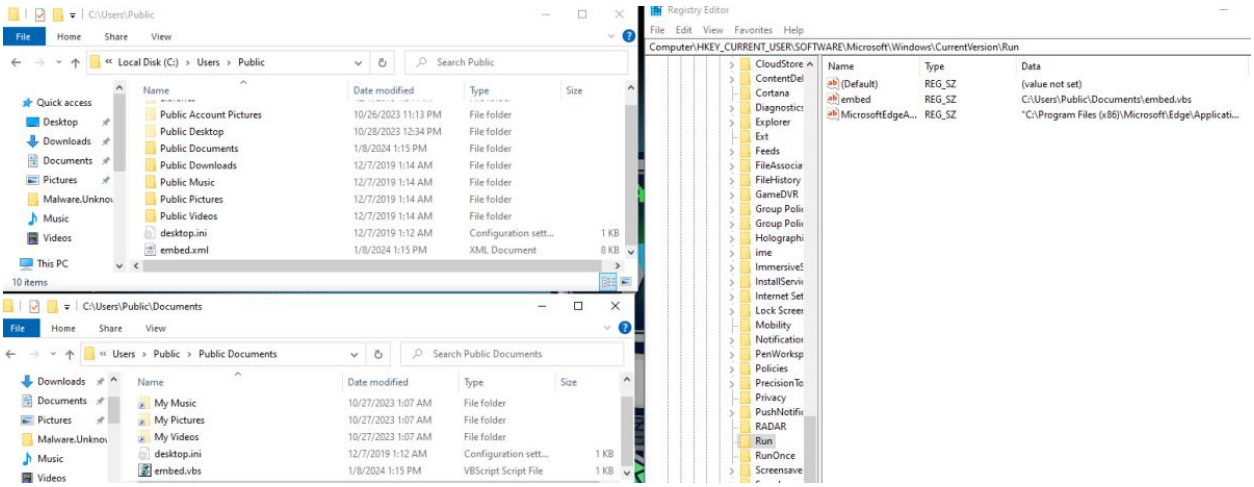
Detonated sample. After "embed" files and registry key are created, I executed “embed.vbs” to simulate user login (when this file is executed normally). There is a DNS record for hxxp://srv[.]masterchiefsgruntemporium[.]local/en-us/index[.]html and HTTP GET to [domain]/en-US/test.html, then later POST to [domain]/en-US/test.html. The HTTP requests might be insignificant, a function of INetSim, or could be communication with a remote app on a server.



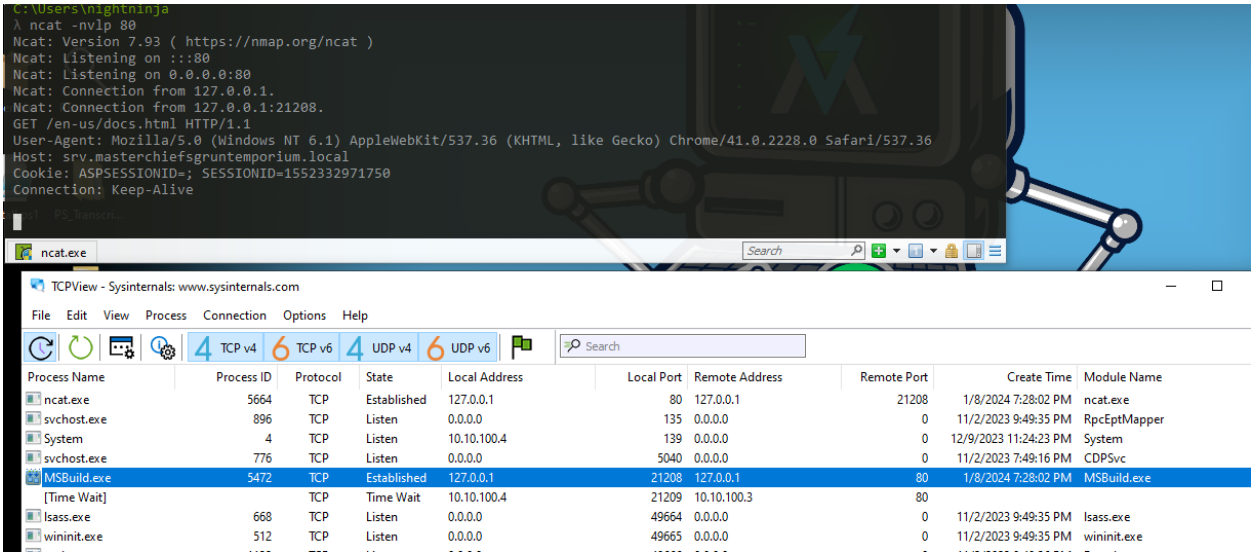
DNS domain callback (highlighted yellow) and HTTP GET to same domain (highlighted pink)



HTTP POST to malicious domain



"Embed" files and registry entry



Using netcat in attempt to emulate remote server for C2 purposes

Advanced Static Analysis

Loaded malware DLL into dnSpy. Looking at the Program class (Cryptor class is just encryption/decryption code): private static void embed() is the main function. The function private static void Main(string[] **args**) is empty.

Start with a bytes array (called "array") that creates a SHA256 hash of the string "p0w3r0verwh3lm1ng!":

```
byte[] array = SHA256.Create().ComputeHash(Encoding.UTF8.GetBytes("p0w3r0verwh3lm1ng!"));
```

Then a string (named "text") is created and writes AES-decrypted/Base64-decoded bytes to a memory stream to be read from memory later. It takes the "array" variable as an argument (as it contains the decryption hash):

```
string text = new StreamReader(new MemoryStream(Cryptor.AES_Decrypt(Convert.FromBase64String(“wall o' text”), array))).ReadToEnd();
```

Next everything in “text“ variable is written to a file called "embed.xml" in the "public" directory:

```
File.WriteAllText(Environment.GetEnvironmentVariable("public") + "\\embed.xml", text);
```

Another string variable called "text2" is created and writes Base64-decoded bytes to a memory stream to be read from memory later:

```
string text2 = new StreamReader(new MemoryStream(Convert.FromBase64String("Text"))).ReadToEnd();
```

Then "text2" is written to folder C:\Users\Public\Documents in a file called "embed.vbs":

```
File.WriteAllText("C:\\Users\\Public\\Documents\\embed.vbs", text2);
```

There's a "try/catch" block. It "tries" to create a new Registry key under CURRENTUSER, subkey "Software\Microsoft\Windows\CurrentVersion\Run", values "embed" (key name) and "C:\Users\Public\Documents\embed.vbs (bet this is for persistence). If this fails, the program "catches" to error message and prints it to the standard output stream:

```
try{RegistryKey.OpenBaseKey(RegistryHive.CurrentUser, RegistryView.Registry64).OpenSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Run", true).SetValue("embed", "C:\\Users\\Public\\Documents\\embed.vbs");} catch (Exception ex){Console.WriteLine(ex.Message);}
```

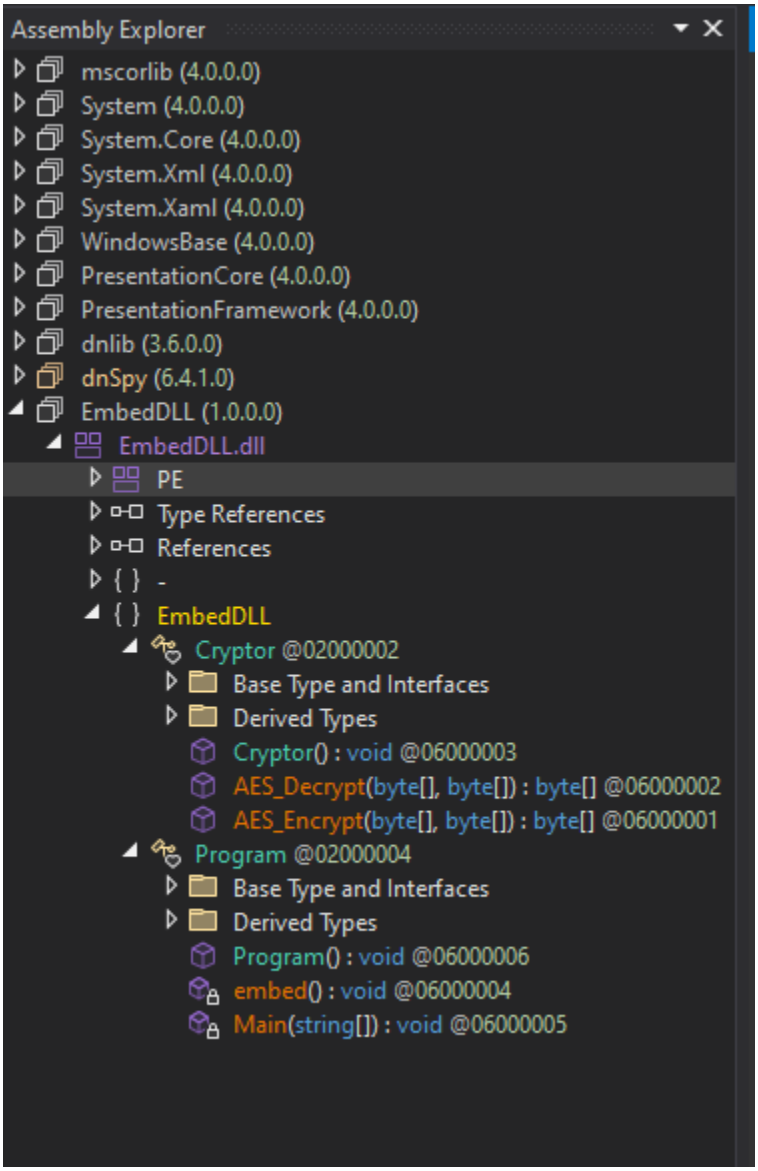
Here's a breakdown of the two “embed” files:

embed.vbs:

This file performs code execution after a successful user login after initial infection. The file creates a “Wscript.Shell” object to call to “MSBuild.exe” to run “embed.xml”. Since “embed.vbs” is referenced in a registry key called ‘embed’ at HKCU\Software\Microsoft\Windows\CurrentVersion\Run, the infection will persist for every subsequent login until removed. Shell runs in a hidden prompt window.

embed.xml:

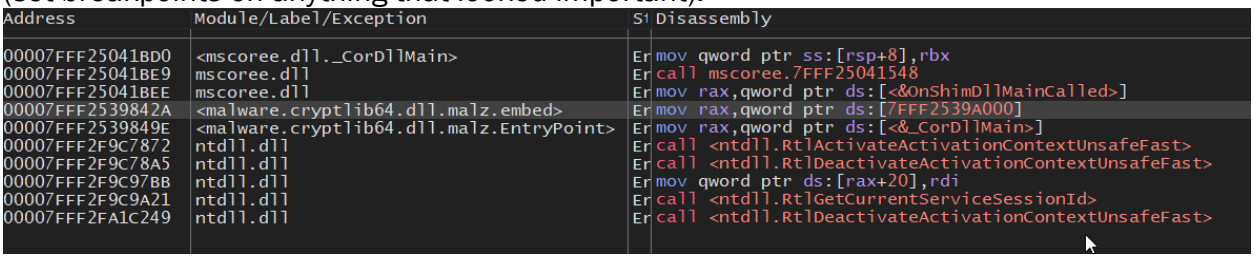
A Base64-encoded and compressed file containing the code to be run (loaded as reflection assembly) by MSBuild.exe. It attempts to contact the callback URL of hxxps://srv[.]masterchiefsgruntemporium[.]local If successful, there’s an HTTP GET then POST to an endpoint at hxxps://srv[.]masterchiefsgruntemporium[.]local, which looked like a connection to a remote application (HTTP request have “Cookie: ASPSESSIONID=; SESSIONID=1552332971750” in header.)



Cryptlib64 loaded into dnSpy

Advanced Dynamic Analysis

Dynamic analysis seems to be different for this C# sample since the IL is what we're actually analyzing (see the **Advanced Static Analysis** section). I'm seeing some unfamiliar API(?) calls (set breakpoints on anything that looked important):

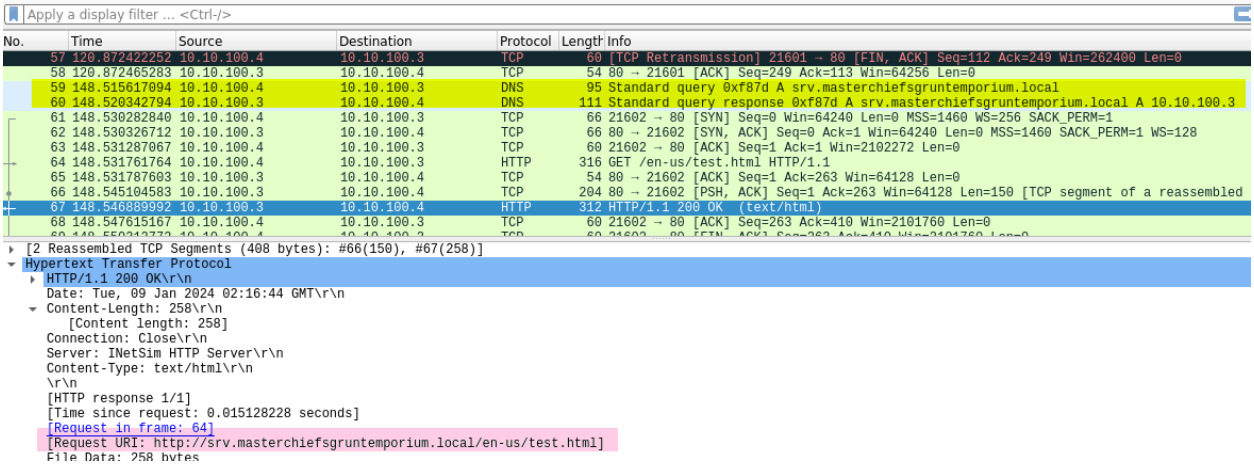


Still working on this section, but I feel the Advanced Static Analysis section has more benefit and information

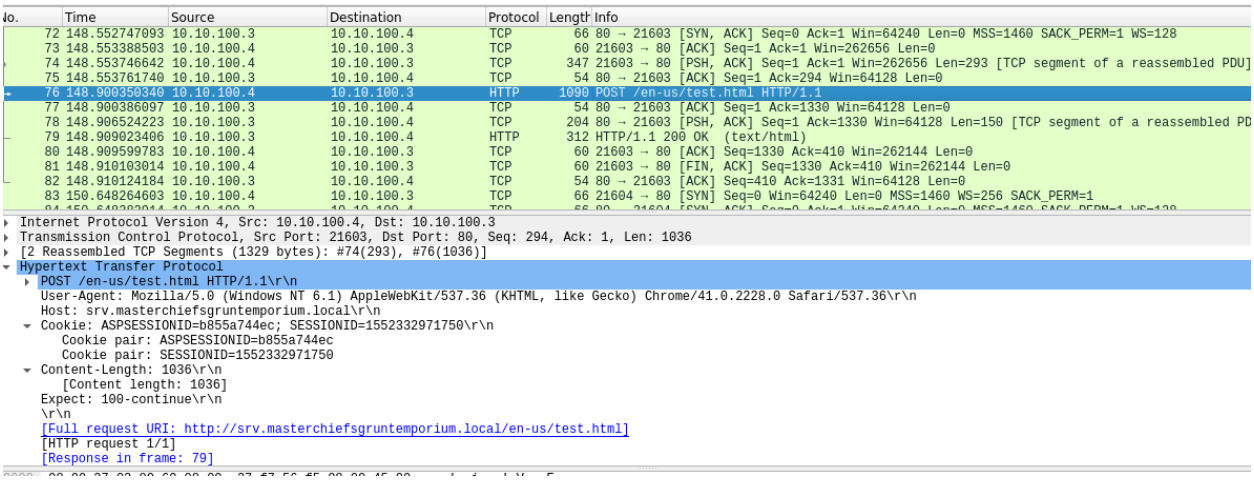
Indicators of Compromise

The full list of IOCs can be found in the Appendices.

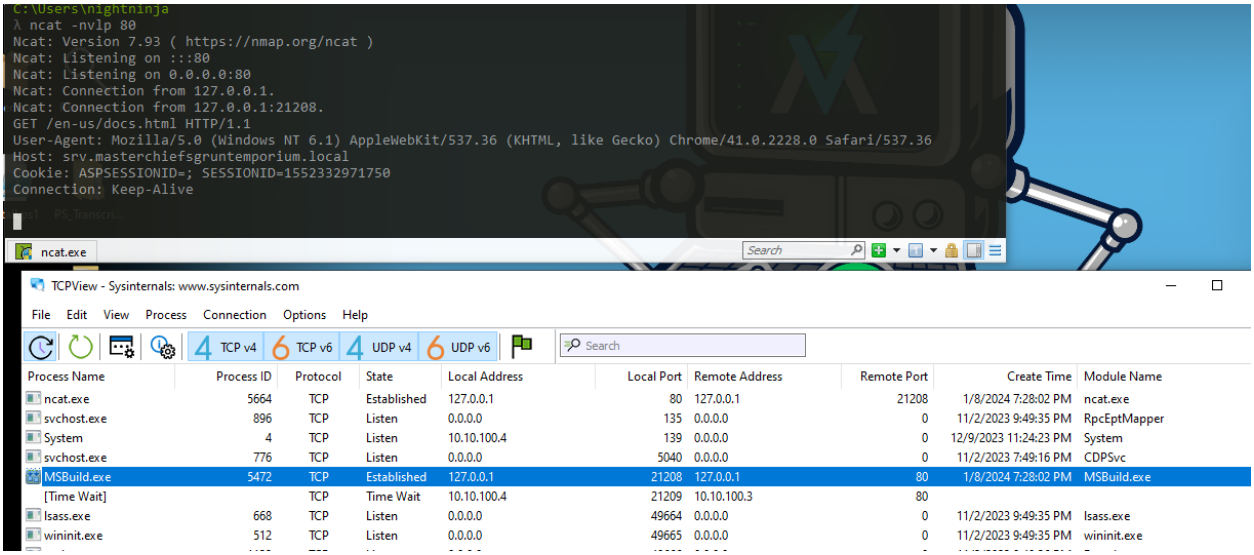
Network Indicators



DNS domain callback (highlighted yellow) and HTTP GET to same domain (highlighted pink)

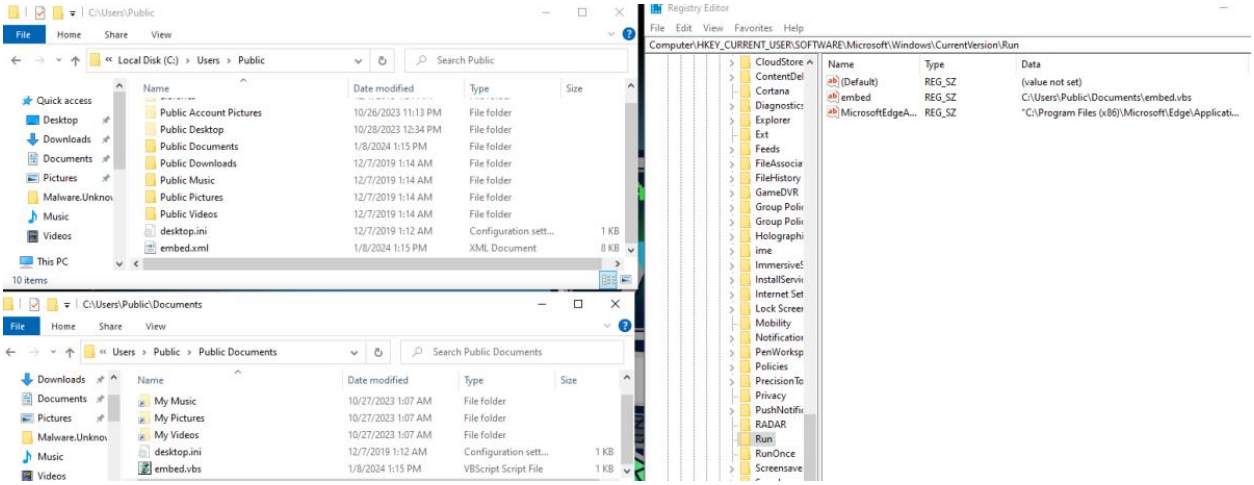


HTTP POST to malicious domain



Using netcat in attempt to emulate remote server for C2 purposes

Host-based Indicators



"Embed" files and registry entry

Rules & Signatures

- Rule for first byte of file being “MZ”
- Rule for “mscorlib” and “mscoree”
- Rule for .NET namespace “System.Security.Cryptography”
- Rule for bytecode that correlates to the “embed” registry key
- Rule for bytecode that correlates to the “embed” files

Yara Rules

Full Yara repository located at: https://github.com/darknessfalls/malware-analysis

```
rule cryptlib64 {

    meta:
        last_updated = "2024-01-08"
        author = "Jarrett Sams"
        description = "My Yara rule for the PMAT final (cryptlib64.exe)"

    strings:
        // Fill out identifying strings and other criteria
        $namespace = "System.Security.Cryptography"
        $embed_files_bytecode = { 72 3? ?? ?? ?? }
        $embed_reg_key_bytecode = { 72 83 ?? ?? ?? }
        $dll1 = "mscoree"
        $dll2 = "mscorlib"
        $PE_magic_byte = "MZ"

    condition:
        // Fill out the conditions that must be met to identify the binary
        $PE_magic_byte at 0 and
        ($embed_files_bytecode and $embed_reg_key_bytecode and $dll1 or $dll2) or
        ($namespace and $dll1 or $dll2)
}
```

```
C:\Users\nightninja\Desktop
λ yara64.exe yara_template.yara Malware.cryptlib64.dll -s -w -p 32
cryptlib64 Malware.cryptlib64.dll
0xd75:$namespace: System.Security.Cryptography
0x6b8:$embed_files_bytecode: 72 3A 52 00 70
0x6e2:$embed_files_bytecode: 72 3B 54 00 70
0x70c:$embed_files_bytecode: 72 3B 54 00 70
0x6fc:$embed_reg_key_bytecode: 72 83 54 00 70
0x688e:$dll1: mscoree
0x107f:$dll2: mscorlib
0x0:$PE_magic_byte: MZ
```

Checking YARA rule against the sample