Novel Technique Combination Used In IDATLOADER Distribution

By Dave Truman

Published: 2024-06-24 · Archived: 2026-04-06 00:37:02 UTC

Kroll's Managed Detection and Response (MDR) team responded to an incident in which suspected malware was exhibiting strange download behavior. After successfully containing and resolving the incident, Kroll's Cyber Threat Intelligence (CTI) team investigated further.

The investigation uncovered a complex infection chain involving many layers of obfuscation being used to deliver IDATLOADER. Ultimately this would result in the deployment of information stealing malware. The infection hinged around utilizing Microsoft's mshta.exe to execute code buried deep within a specially crafted file masquerading as a PGP Secret Key. The campaign made used of novel adaptations of common techniques and heavy obfuscation to hide the malicious code from detection the extent of which is described below.

This incident involved a victim accessing a Bollywood pirate movie download site. When attempting to download a video, the victim was directed to a page hosted on Bunny CDN that provided a bit[.]ly link that ultimately download a ZIP file.

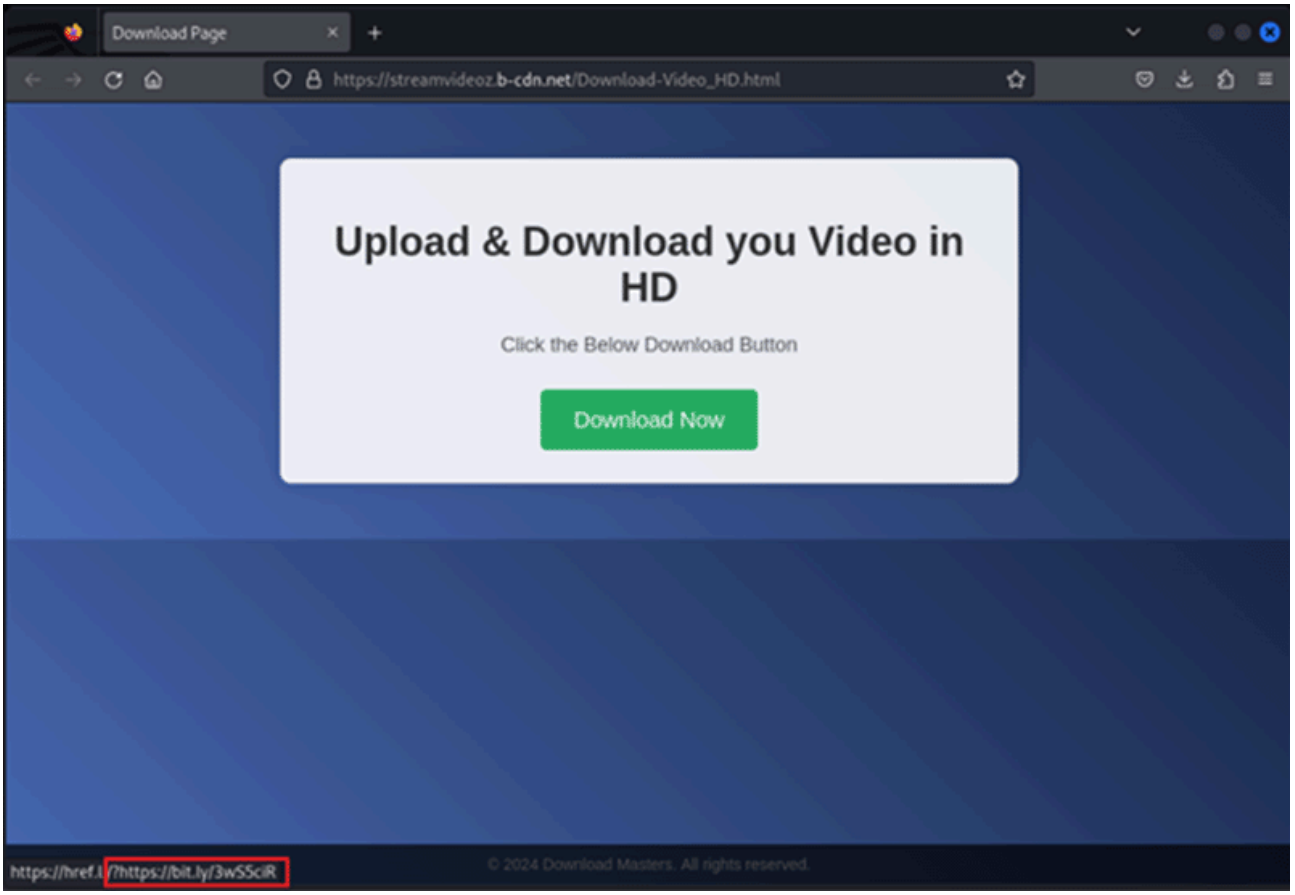

Figure 1 – Download page providing link to zip file

The downloaded ZIP file contained another ZIP file, which was password protected, along with a text file containing the password. The nested ZIP file contained a 192 MB LNK file along with a decoy “trailer” video file.

The LNK file triggered the first element of the novel technique used in this infection chain for distributing IDATLOADER. The LNK file was using mshta.exe to execute what appeared to be a “PGP Secret Key,” hosted again hosted on Bunny CDN. The Microsoft binary mshta.exe is used to execute “HTML Application” files, which contain HTML markup and web technology scripting languages such as JavaScript. HTML applications should follow HTML standards including the use of supported character sets, since HTML is primarily text based. However, looking at the file being downloaded in a hex editor, it clearly contains a large amount of binary data.

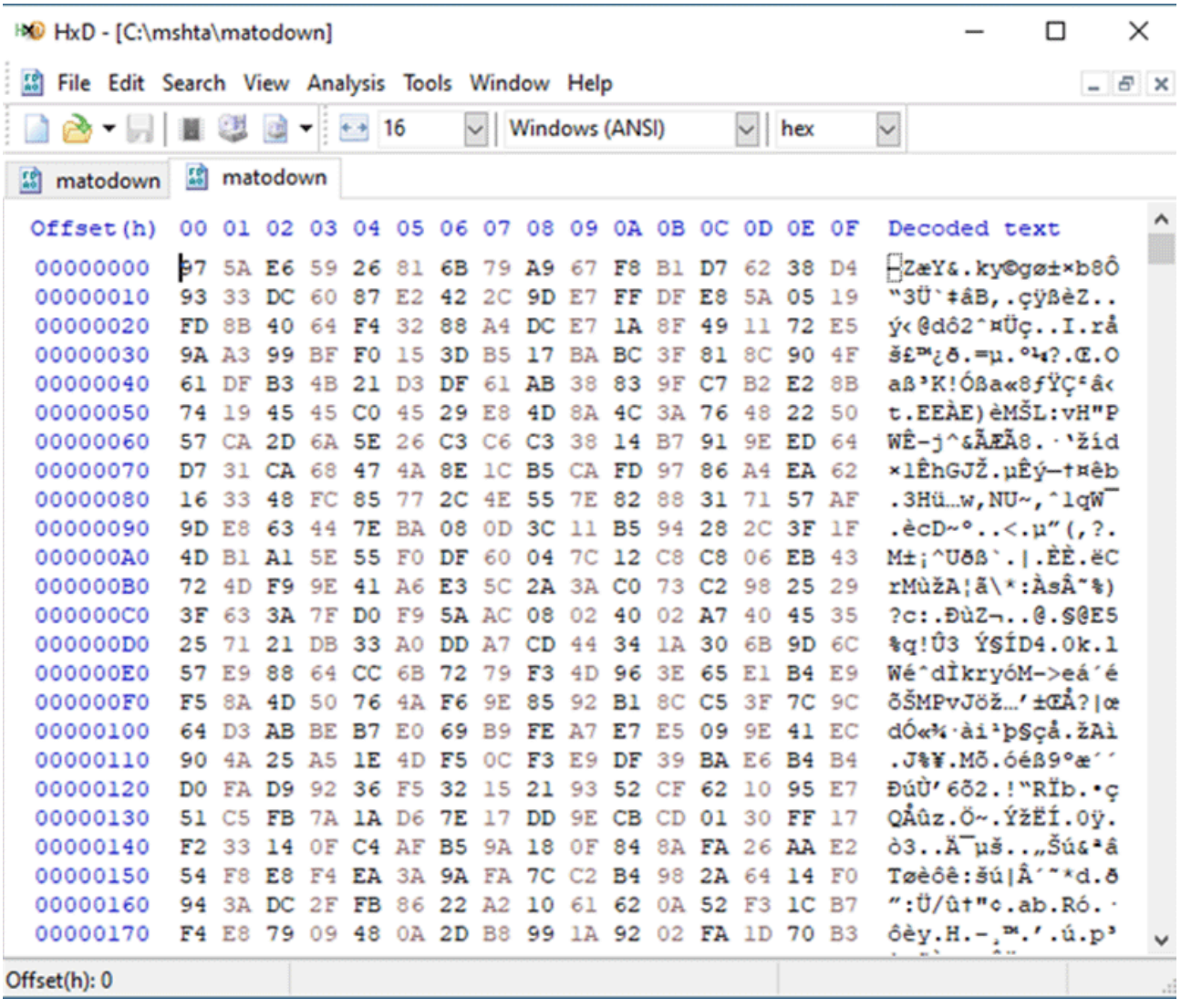

Figure 2– Downloaded “PGP Secret Key” clearly containing binary data

Static analysis indicates that the file is not a legitimate PGP Secret Key, but an amalgamation of a large set of junk bytes, an embedded HTA file and an embedded EXE file.

It is worth noting that junk non-printable bytes can be seen even inside the embedded HTA file. The reason the file is being interpreted by tooling as a PGP key is simply because the first two bytes of the file are the magic bytes for a “PGP Secret Sub-key”. The embedded EXE file is the legitimate calc.exe supplied with the Windows operating system, likely to add known good indicators for bypassing AI/ML detections.

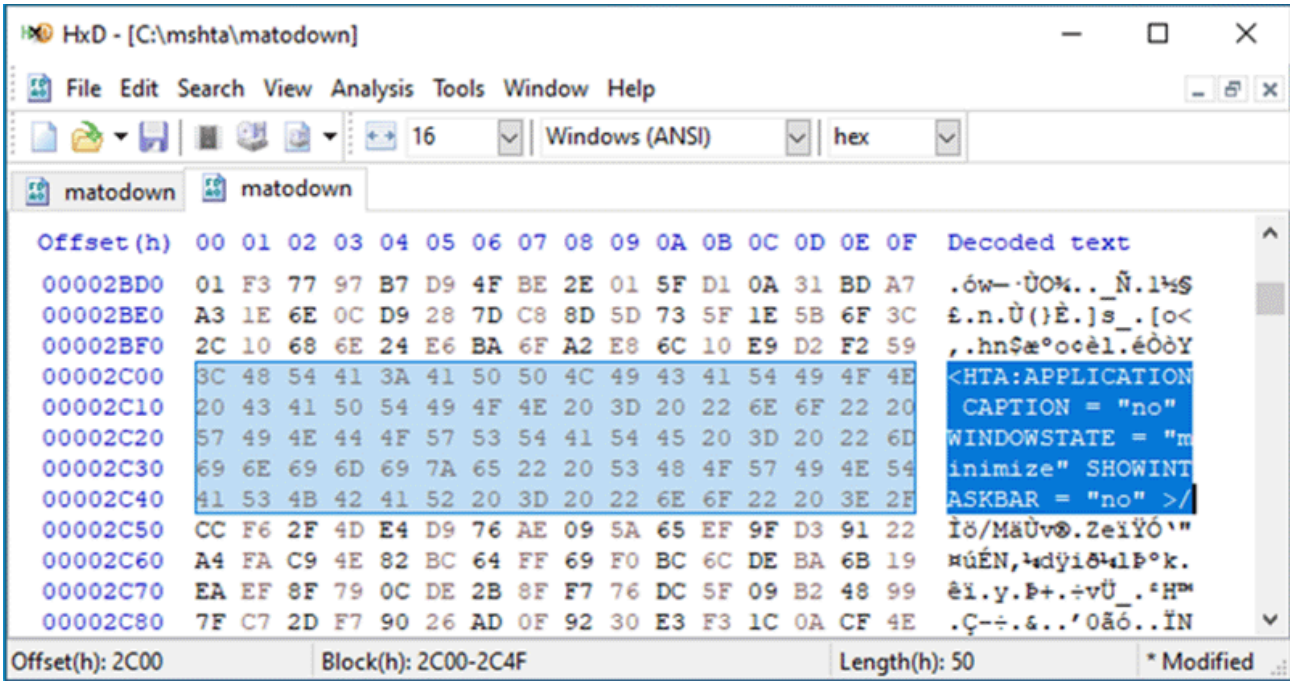

Figure 3 – Embedded HTA within suspicious binary download

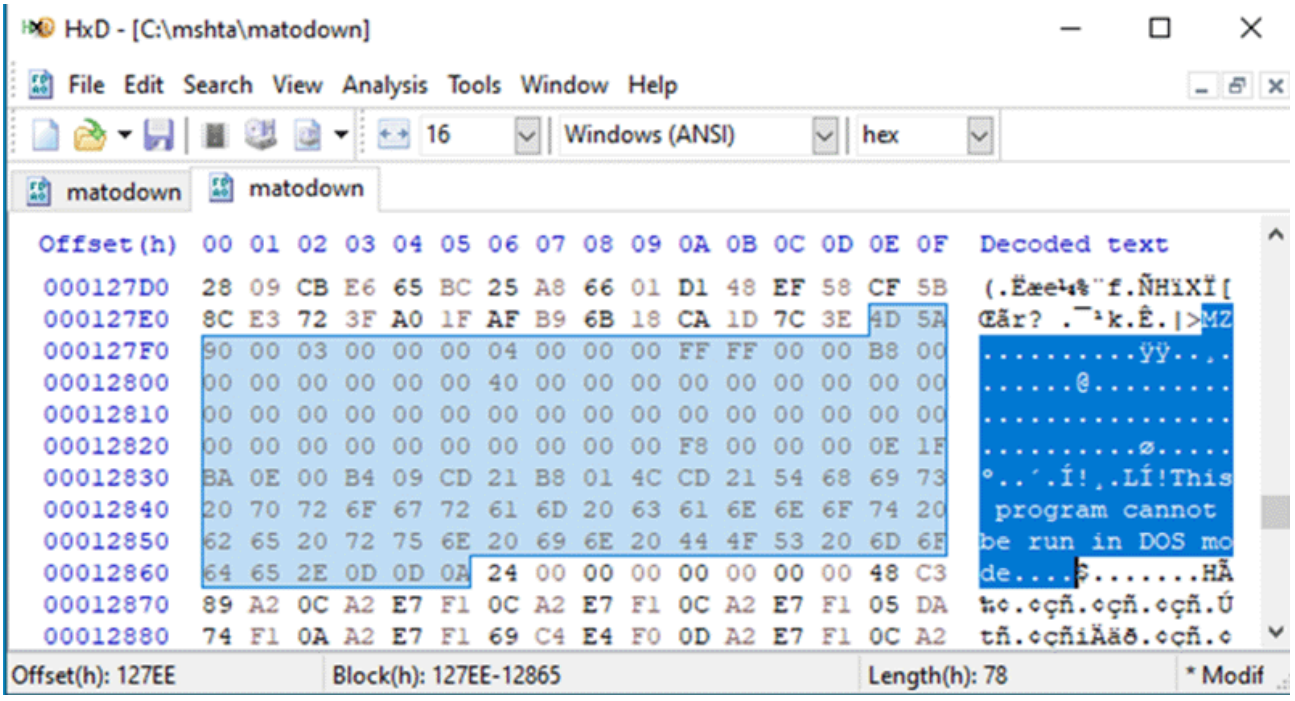

Figure 4 – Embedded calc.exe within suspicious binary download

The file itself has an extremely low detection ratio on VirusTotal with only one out of 70 anti-virus engines detecting it as malicious.

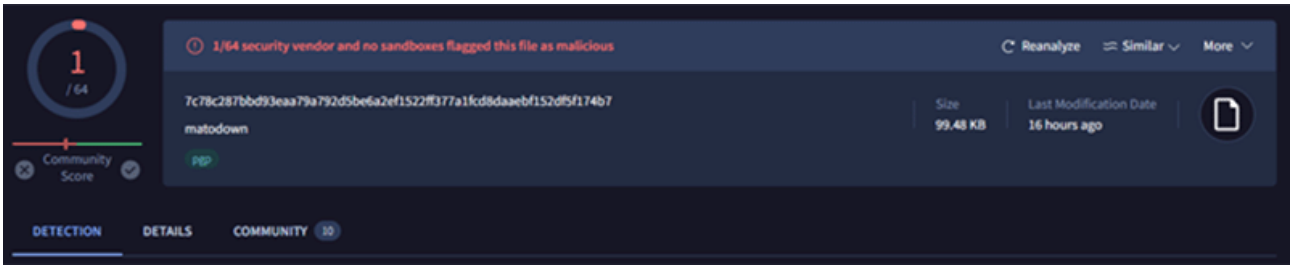

Figure 5 – Extremely low detections for the suspicious binary download

From this it appears that mshta.exe will execute the HTA code hidden within this file even through the file itself is not a legitimate HTA containing HTML that meets the standard. The Kroll CTI team ran the file through the World Wide Web Consortium (W3C) HTML validator, which gave up validating when the number of parsing errors exceeded 1,000. Some example HTML errors that occurred were:

- Forbidden code point
- Malformed byte sequences
- Non-space characters found without seeing a DOCTYPE first
- Bad character after <
- & did not start a character reference
- A slash was not immediately followed by >
- < in attribute name
- Quote ' in attribute name

It is common for web browsers to try to render an HTML page even if there are errors. This is because of the number of inconsistences between different web browsers, and poor coding practices or lack of testing by developers for the millions of websites on the internet. Microsoft's mshta.exe continues this practice. But there is an important difference between a web browser and mshta.exe: Web browsers are usually sandboxed and do not allow the scripts to interact directly with the operating system, while mshta.exe scripts can interact with Windows without these restrictions. The technique used here allows a malicious script to potentially mimic hundreds of possible file types, many of which will be treated differently by various security tools depending on what file type they mimic, allowing for easy bypasses. The threat actor takes advantage of this behavior that the actor is taking advantage of to deploy IDATLOADER.

The Kroll CTI team performed some testing to demonstrate this technique. We appended an HTA file that contained code to launch notepad.exe to a copy of calc.exe. The operating system detects the resulting file as a PE file.

We then ran the resulting EXE file with mshta.exe and then directly from cmd.exe. When launched with mshta.exe. Notepad was launched without warnings or errors.

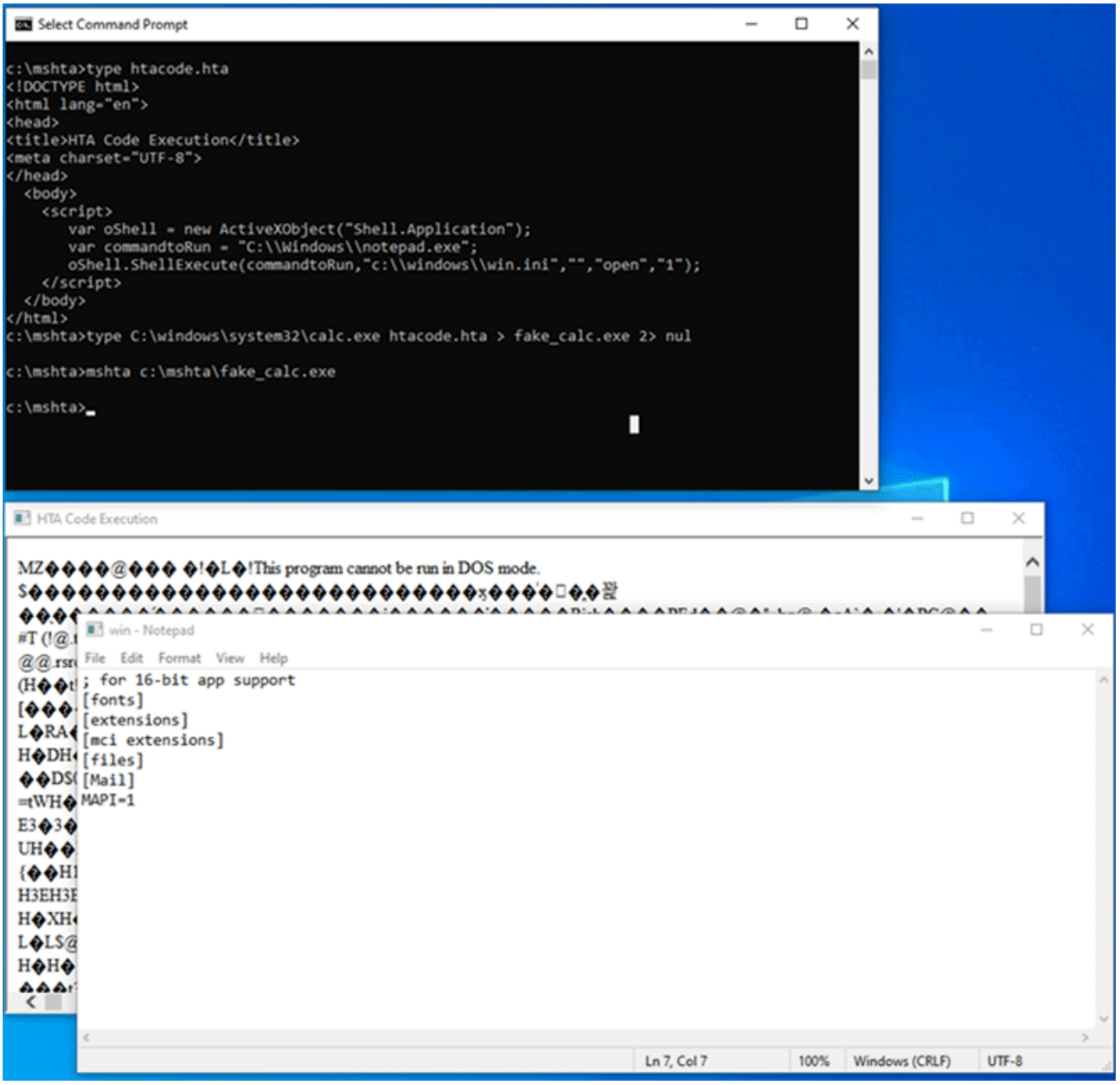

Figure 6 – HTA code executing in mshta.exe regardless of file type and HTML standards nonconformity

When the Kroll CTI team ran a generated test file directly from the command line without mshta.exe, the calc.exe image is started as a process.

We then generated an HTA file with a series of deliberate bad bytes in the middle of the code and tested this. In this case, mshta.exe still ran the code to launch notepad.exe.

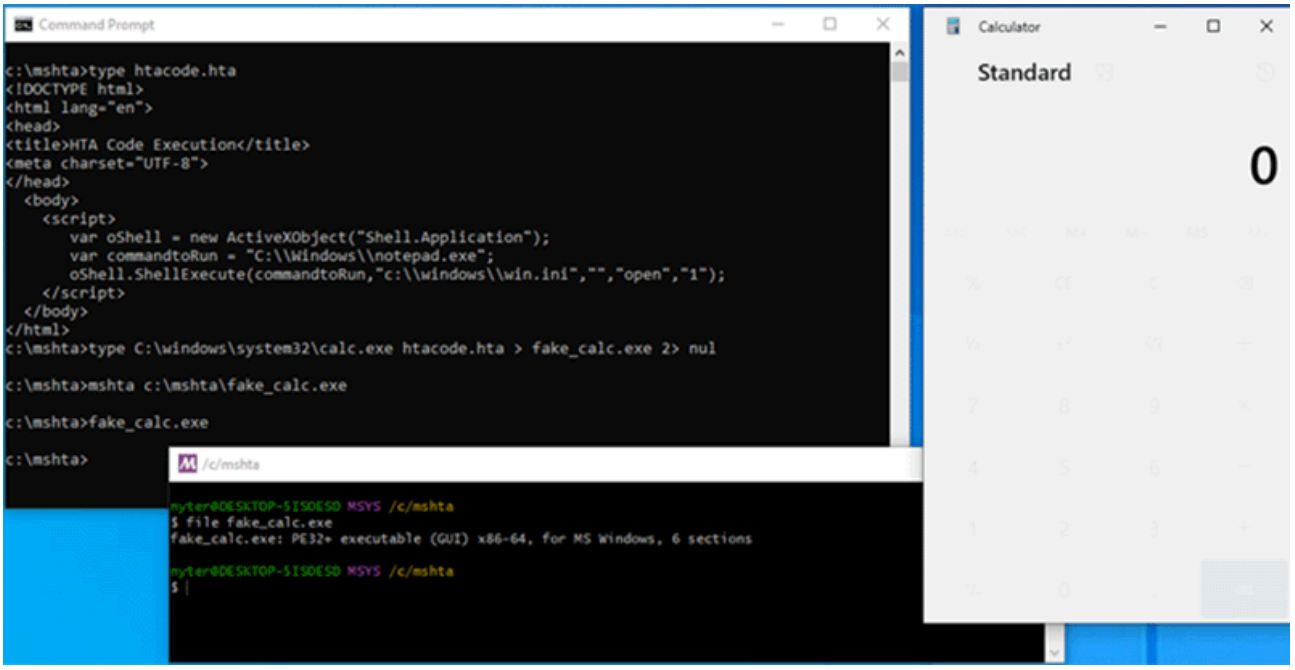

Figure 7 – Test file appears as an EXE file when not opened with mshta.exe

When we extracted the HTA code from the original fake PGP key file, the code was heavily obfuscated and even inside the HTA code there were random non-printable character sequences, making the code invalid for HTML.

Figure 8 – HTA Code extracted from fake PGP Key File

There were four layers of obfuscation. The first three were total obfuscation of the next stage in the obfuscation chain. The fourth stage had readable code, but certain variables had obfuscated content.

Figure 9 – Stages 2, 3 and 4 of de-obfuscation

Figure 10 – Stage 4 modified to make readable

Once fully deobfuscated, the code can be seen to download two separate ZIP archives. The script uses an unzip function with interesting functionality: it will unzip the archive in %AppData% and try to use the ZIP file content as a command to execute. In the case of a ZIP file with lots of files, or with a file that is not executable, this will not work. However, if the ZIP archive contains only one executable file that file gets executed.

This is where the second of the novel combination of techniques occurs. The first of the two ZIP archives, “K1.zip,” contained a large set of files while the second, “K2.zip,” contained a single EXE file.

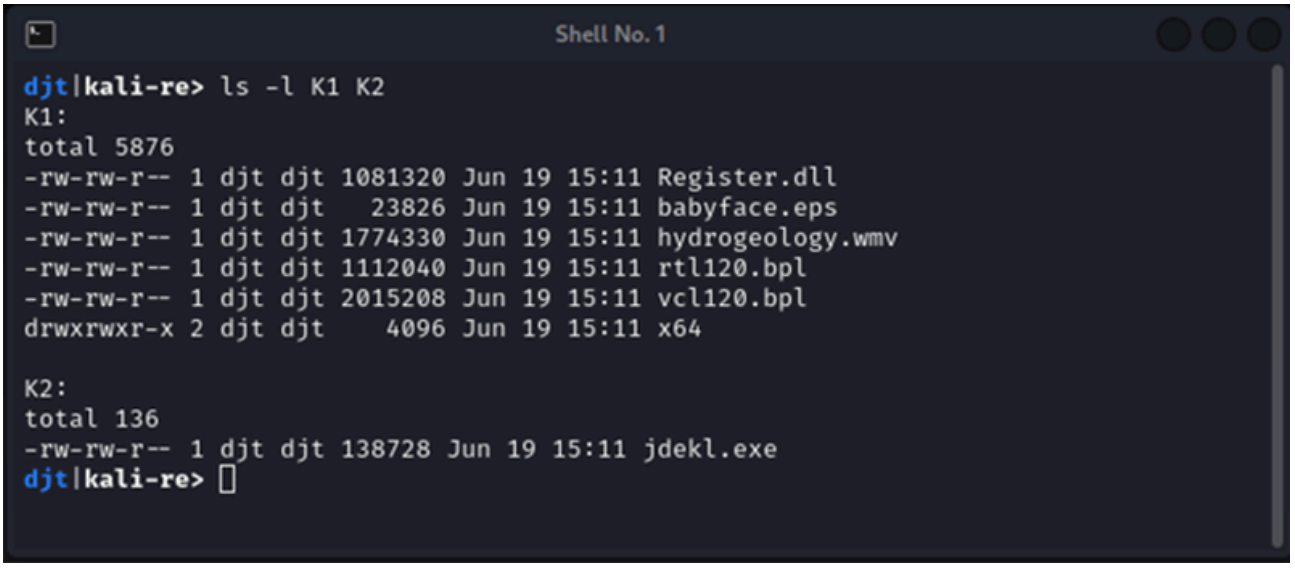

Figure 11 – Contents of downloaded zip files

The file ‘jdekl.exe’ in K2.zip is the renamed legitimate binary RttHlp.exe from IOBit.

The file “hydrogeology” looks like it’s an encrypted payload that gets decrypted and deployed by IDATLOADER, based on the presence of IDATLOADER marker bytes within the file.

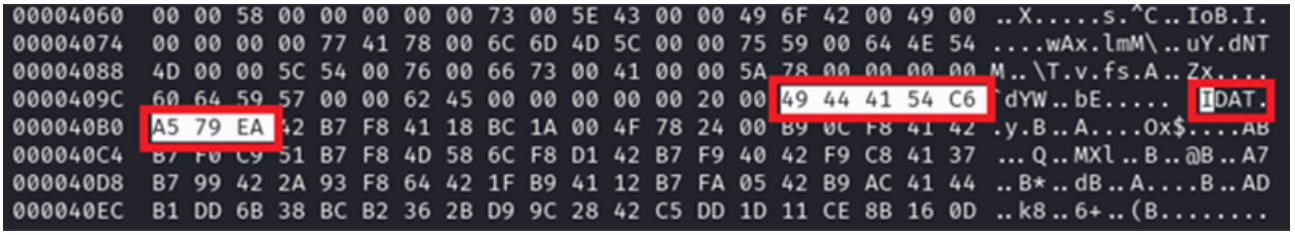

Figure 12 – IDAT Marker bytes within hydrogeology.wav

Initially this appeared to be straightforward DLL sideloading a malicious Register.dll file. However on closer inspection this was not the case.

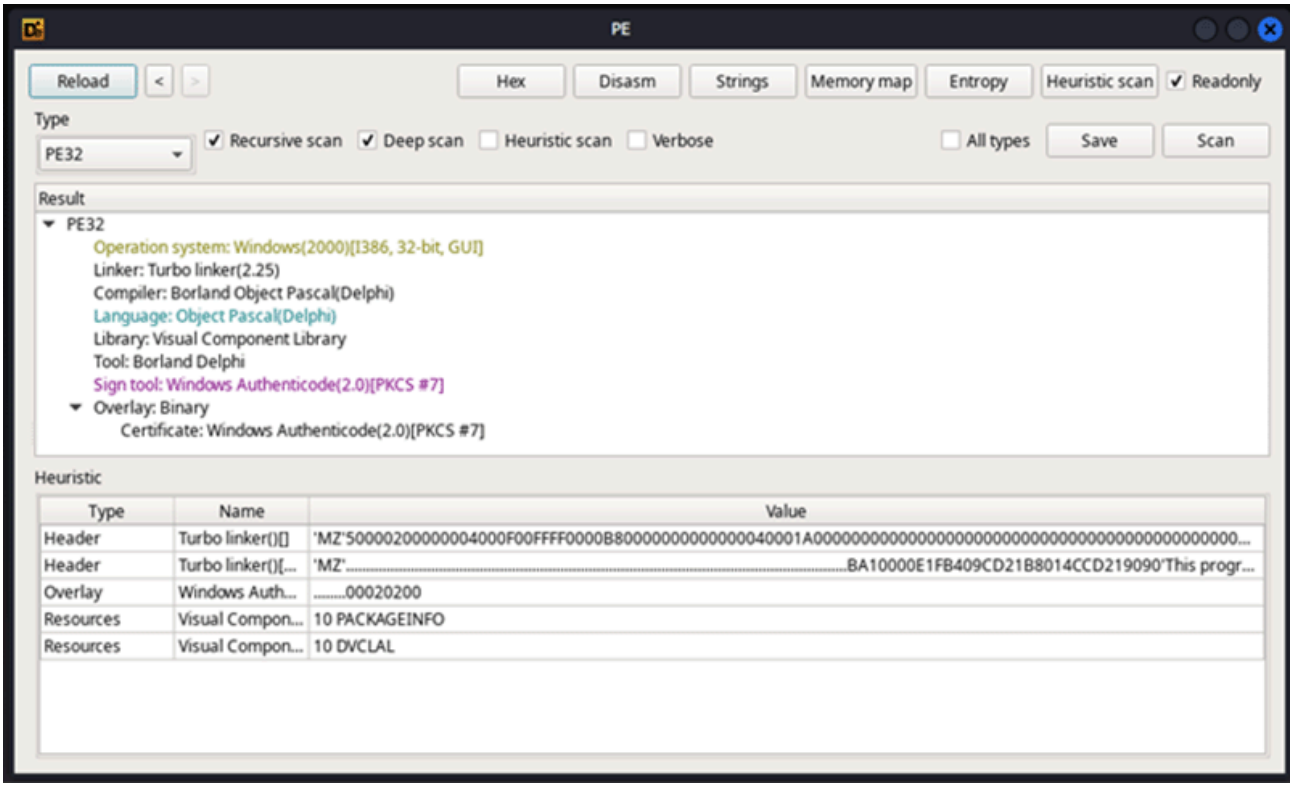

Figure 13 – Executable being detected as compiled with Delphi

The file itself appears to have been written and compiled in Delphi and lLooking at the imports for the EXE it is clear that this EXE does not import Register.dll; however, it does import VCL120.BPL. Instead of being a regular DLL, this is a Borland Package Library (BPL) file, which is a DLL-like file created by Borland for use with their suite of compilation tools (notably including Delphi). So, instead of traditional DLL sideloading, this is a case of BPL sideloading. At time of writing there is no MITRE sub-technique for BPL sideloading, we have raised a request for a new sub-technique to be added.

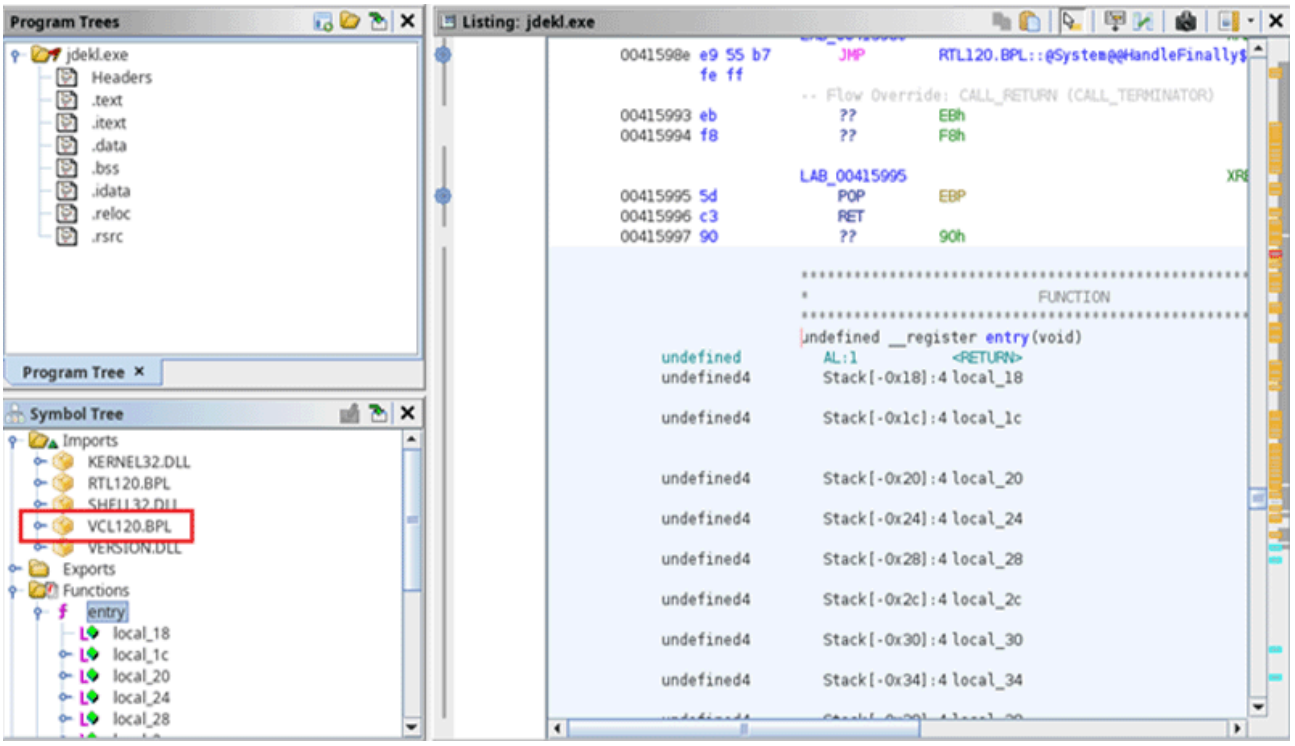

Figure 14 – Screenshot showing BPL file being imported into EXE file

Within the VCL120.BPL there exists code accessing the encrypted data file hydrogeology.wav, indicating this is the file containing the malicious IDATLOADER code.

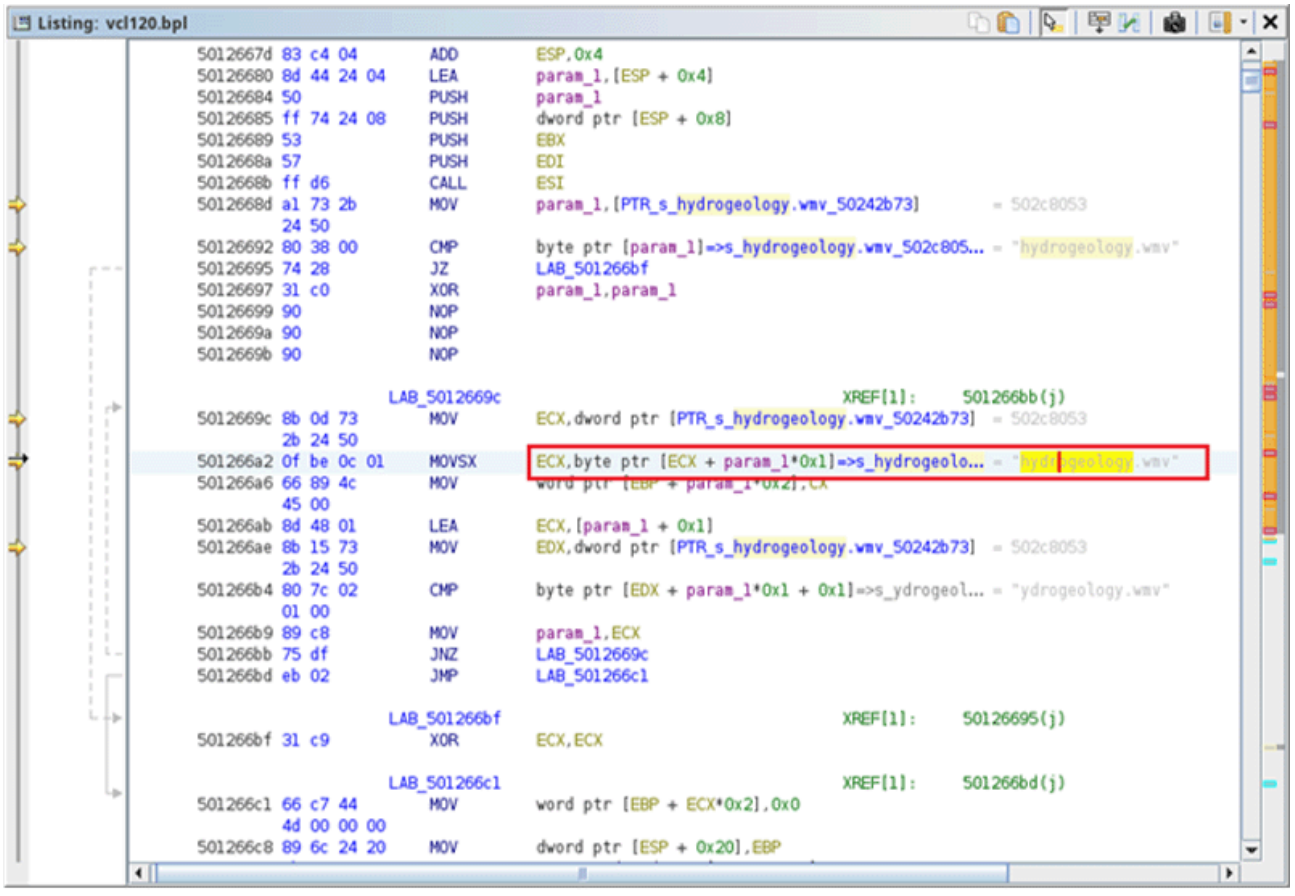

Figure 15 – Code within the malicious BPL accessing IDATLOADER encrypted file

During testing, the Kroll CTI team has seen the chain starting with the fake PGP key file deploying LUMMASTEALER and another currently unidentified generic password stealer, which is currently being analyzed.

Analysis

This IDATLOADER campaign is using a complex infection chain containing multiple layers of direct code-based obfuscation alongside innovative tricks to further hide the maliciousness of the code. This all resulted in a low detection ratio for the initial file. Tools that look at behavior are likely to have an easier time detecting this malware, as opposed to tools that rely heavily on signature-based technologies. As always it's important to have defense in depth so, though a crafted and heavily obfuscated HTA file masquerading as an innocuous file might make it through perimeter scanning, it can still be detected by endpoint detection and response (EDR) or other technologies.

Detection Methods

Threat actors continue to abuse the legitimate and trusted mshta.exe binary in attack chains, detecting this behavior was key in detecting both this infection and a previous incidents involving the TODDLERSHARK malware. Abnormal mshta.exe behavior is a high confidence indicator that malicious activity is taking place. System administrators may consider blocking execution or removing MSHTA altogether as its functionality is tied to older versions of Internet Explorer.
