THREAT HUNTING WITH VELOCIRAPTOR

ANALYSIS OF QUASARRAT EXECUTION AND PERSISTENCE ON WINS 10

AUGUST 14, 2025

PREPARED BY HAN THAO NGUYEN

TABLE OF CONTENTS

I. EXECUTIVE SUMMARY

A targeted threat hunt was conducted to analyze the behavior of a known malicious binary, identified as QuasarRAT, within an isolated lab environment. The analysis leveraged multiple forensic artifacts from Velociraptor, as well as network traffic captured with Wireshark. The investigation successfully confirmed the execution of the malware, its attempt to establish C2 communication, and its use of non-standard file paths to evade detection. While no standard persistence mechanisms were observed, the presence of the known malicious binary and its C2 network activity are sufficient to classify the host as compromised. Key findings include:

1. The presence of two identical binaries, `Jackes.exe` & `mal-sample.exe`, with a **SHA-256** hash of **e015719aad3e56e0d0ea614d0868adf95829164b6f89d2232b6084072b17ab2b**
2. Confirmed execution of Jackes.exe via `Windows.EventLogs.Evtx`.
3. Direct temporal correlation between Jackes.exe execution and outbound network communication to a suspicious remote host (`66.63.187.164:8596`).
4. The absence of standard download evidence (`Windows.Analysis.EvidenceOfDownload`) and persistence mechanisms (Autoruns, ScheduledTasks), highlighting the need for a multi-artifact approach in threat hunting.

II. OVERVIEW

2.1. PURPOSE

The purpose of this report is to document the findings from a forensic analysis of a Windows endpoint suspected of being compromised by QuasarRAT. The report details the steps taken to identify indicators of compromise (IOCs) and the attack timeline.

2.2. SCOPE

The scope of this report is limited to the analysis of data collected from a single Windows client VM within a controlled lab environment. The analysis focuses on Velociraptor artifacts and network traffic captured during a simulated malware execution event.

2.3. THREAT HUNTING TYPE

Threat hunting is the proactive, iterative process of identifying and disrupting threats within an organization's environment [1]. It involves a deeper investigation into data, behavior patterns, and system activities to identify potential threats before they can cause harm. Threat hunting operates on the assumption that an organization is already compromised and is often driven by a hypothesis based on threat intelligence and an understanding of attacker Tactics, Techniques, and Procedures (TTPs), as outlined in frameworks like MITRE ATT&CK [1].

Digital forensics and incident response (DFIR) tools, such as Velociraptor, are essential for successful threat hunting. They provide the ability to collect and analyze forensic artifacts; digital footprints left on a system which serve as evidence of an attacker's actions. These artifacts can include log files, registry keys, network connections, file metadata, and memory dumps. By analyzing these artifacts, a cyber analyst can piece together a timeline of events, understand the scope of a breach, and uncover the TTPs used by the adversary.

In particular, this investigation is classified as a Known Indicator Hunting and Behavioral Analysis exercise. We started with a known malicious sample (**QuasarRAT**) and used a behavioral approach to confirm its execution, network activity, and persistence attempts on the host. This contrasts with a broader, more open-ended Anomaly Hunting exercise.

2.4. CASE SCENARIO

A Windows 10 user, IEUser, executed a suspicious file (**`Jackes.exe`**) from a non-standard directory (**C:\Users\IEUser\AppData\Roaming\Wave Google**). This activity triggered an investigation to determine if the host was compromised, identify the full extent of the malware's activity, and collect forensic artifacts.

2.5. THREAT HUNTING ARTIFACTS AND METHODOLOGY

A structured approach was used for the threat hunt, which involved running two distinct hunts, a pre-infection baseline and a post-infection snapshot. This method allows for a differential analysis to pinpoint changes made by the malware. The following artifacts were selected for collection in both hunts to enable a comprehensive "apples-to-apples" comparison:

Table 1: Artifacts for Baseline Hunt, categorized by their purposes

Artifact	Purpose (Simplified)
Execution & Downloads	
Windows.Analysis.EvidenceOfDownload	Finds files downloaded via browser or app. Useful for tracing initial malware dropper.
Windows.Analysis.EvidenceOfExecution	Tracks executed programs (via prefetch, Amcache, etc). Verifies if a suspicious file ran.
Windows.System.Pslist	Shows all active processes. Helps you catch live malware.
Windows.Attack.ParentProcess	Links child processes to parents. Spot suspicious chains like winword.exe → powershell.exe.
Windows.Attack.UnexpectedImagePath	Flags programs run from odd locations (e.g., %APPDATA%, %TEMP%). Common hiding spots for malware.
Persistence & Autostarts	
Windows.Sysinternals.Autoruns	Lists startup locations (registry, services, scheduled tasks). Reveals persistence techniques.

Windows.EventLogs.ScheduledTasks	Finds scheduled tasks that may auto-run malware on boot.
Windows.Registry.RDP	Checks for changes to RDP settings in registry. Could signal remote access setup.
Malware Detection & Binary Analysis	
Windows.Detection.BinaryHunter	Hunts for suspicious or known-malicious binaries across the system.
Windows.Detection.BinaryRename	Finds executables that have suspicious or misleading names (e.g., svch0st.exe).
Windows.Detection.Impersonation	Detects credential abuse or impersonation attempts by malware.
Windows.Detection.Mutants	Detects named mutex objects used by malware to avoid multiple executions.
Windows.System.UntrustedBinaries	Flags unsigned/untrusted executables — common in malware.
Windows.Memory.PEDump	Dumps executables loaded in memory. Great for catching fileless malware or injected code.
Network Activity & C2 Detection	
Windows.System.DNSCache	Displays resolved domains. Useful for spotting C2 domains or phishing links.
Windows.Network.NetstatEnriched	Lists all TCP/UDP connections with context. Spot unusual or long-lived C2 sessions.
Event Logs (Forensics & Anomaly Detection)	
Windows.EventLogs.Evtx	Collects raw Windows event logs (Security, System, Application). Great for manual log review, timeline building, or exporting for external SIEMs.
Windows.EventLogs.EvtxHunter	Scans event logs for known suspicious patterns, like login failures, service changes, or privilege escalation.

The first hunt established a baseline, and the second captured the system state after executing the QuasarRAT variant (`SHA-256: e015719aad3e56e0d0ea614d0868adf95829164b6f89d2232b6084072b17ab2b`), which was renamed `quasarRAT-sample.exe` for easier tracking.

2.6. LAB SETUP

2.6.1. LAB ENVIRONMENT / NET WORK CONFIGURATION

2.6.1.a. Phase 1 – Isolated Environment (No Internet Connection)

The analysis was performed in a safe, isolated lab environment. A Windows 10 client VM (`192.168.100.20`) was connected via a LAN segment called **InternalLabNet** to an **Ubuntu server** (`192.168.100.10:8000`). The Ubuntu server hosted the Velociraptor server and **INetSim** to simulate internet services and monitor C2 traffic. A simple Python HTTP server was also set up on the Ubuntu machine to facilitate the transfer of the Velociraptor agent to the client VM. Below are the screenshots showing the hardware settings and network adapter configuration, specifically a new LAN segment named "InternalLabNet" to isolate communication between soon-to-be infected client and monitoring server and prevent malware from reaching the internet, retrieving more payloads and affecting a production environment. Later, we perform extra assessments using NAT adapter and enable internet connection on client machine to view the full extent of the malware behaviors (similar to real-life infection). Of course, in any case, all AV and Firewall settings on client machine will be turned off for full capture of malware activities in sandbox environment, so extra precautions are still advised.

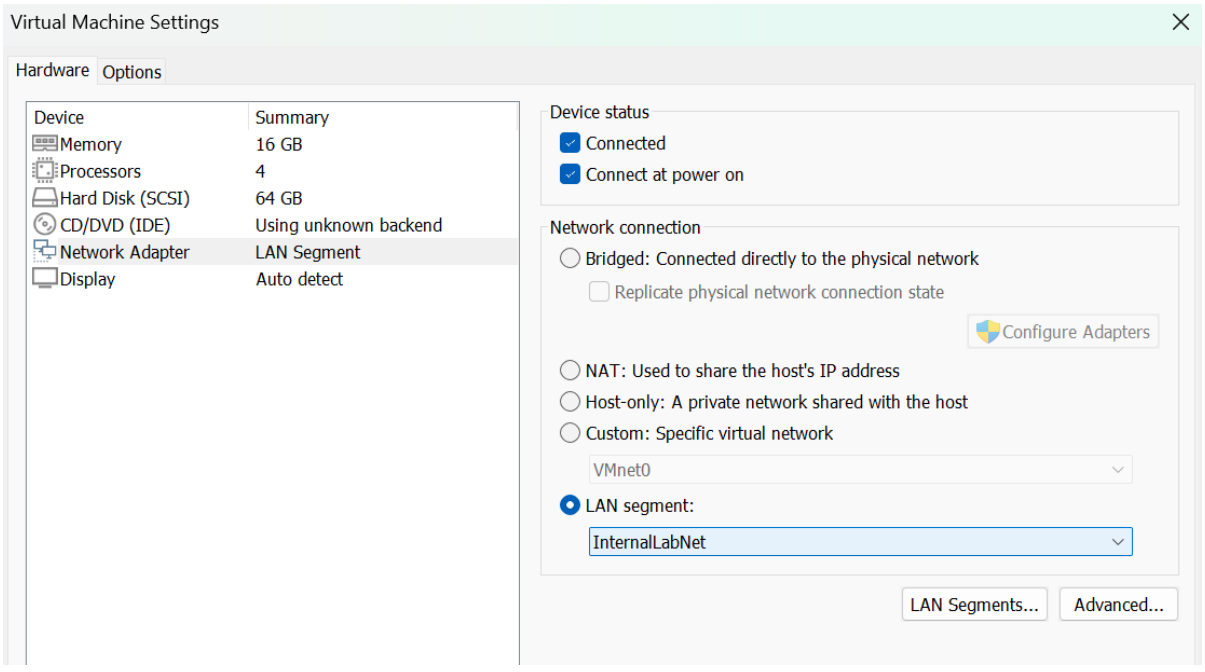


Figure 1.1: Setup private LAN segment between Windows Client VM and Ubuntu Velociraptor Server

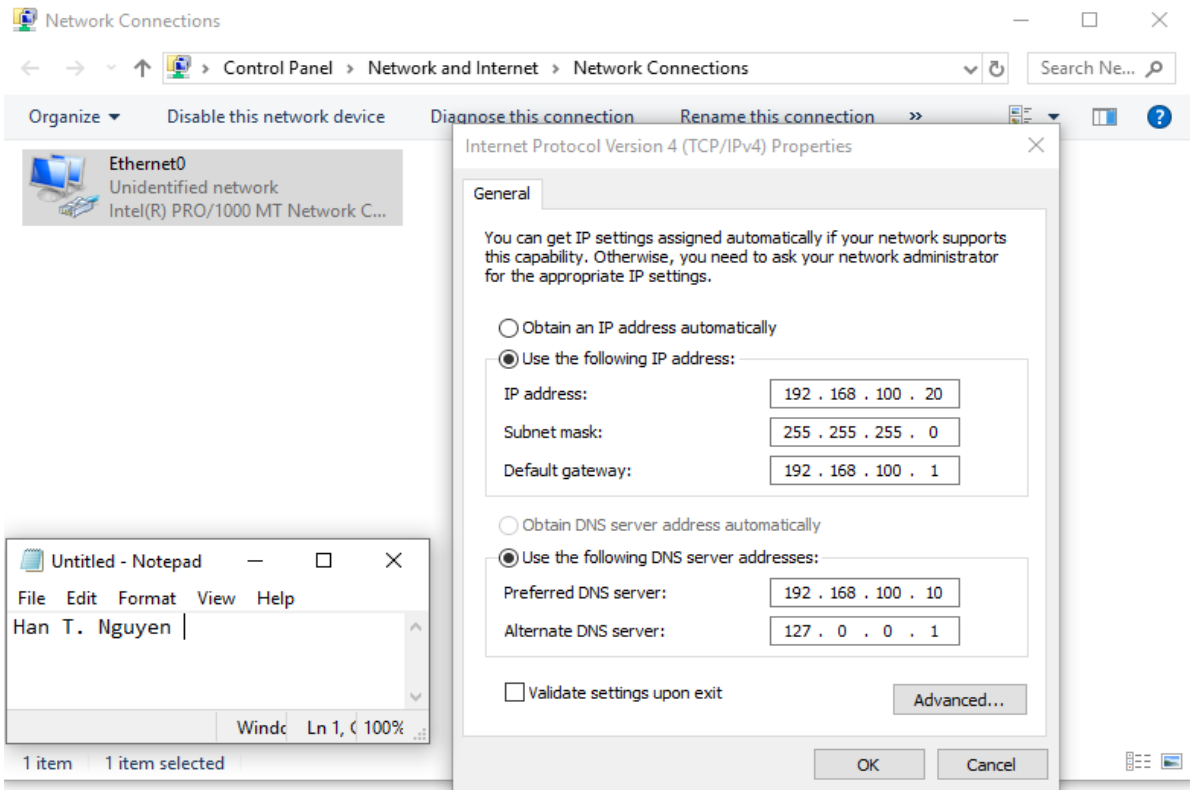


Figure 1.2: Windows Network Connections Menu and IPv4 Properties showing Manual configuration of the Windows client's IP address (192.168.100.20) and DNS server (192.168.100.10).

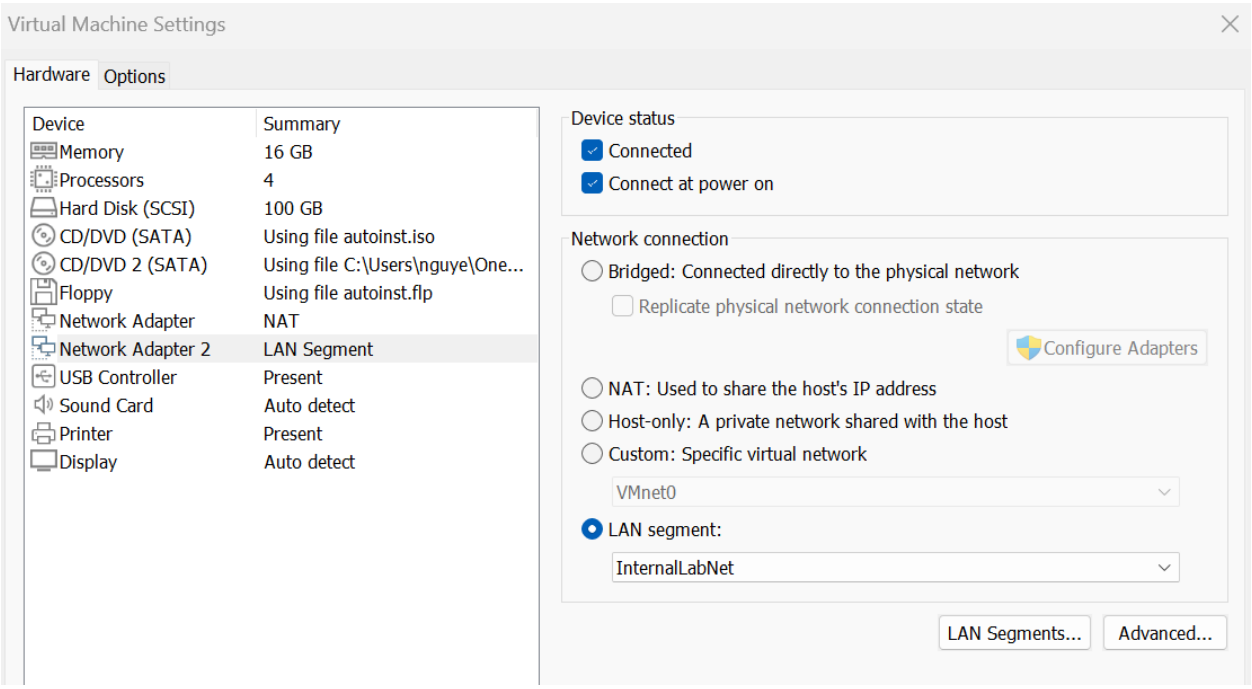


Figure 1.3: VM Settings for an Ubuntu Server, configured to use the "InternalLabNet" LAN segment

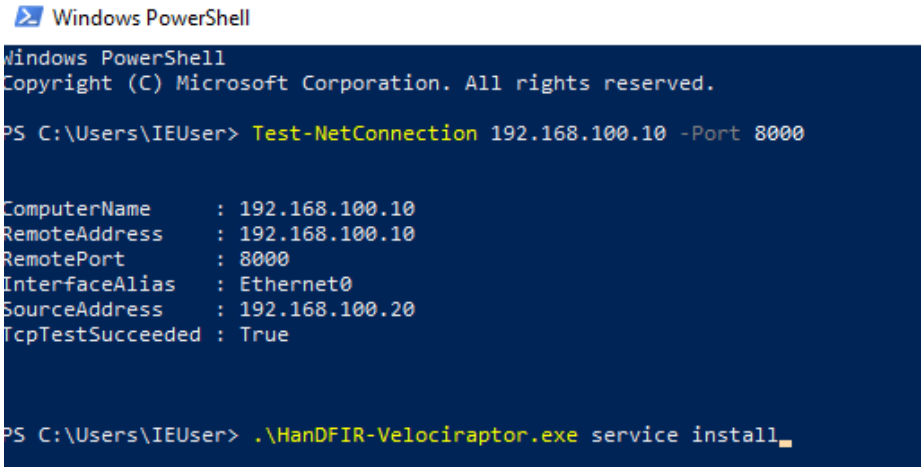


Figure 1.4: Make sure two machine can connect with each other on port 8000 (TcpTestSucceeded: True)

2.6.1.b. Phase 2 – Production Environment (With Internet Connection) – Advanced Lab Setup for Full Malware Behavioral Analysis:

The Windows VM (client) was configured with two network adapters: one connected to the internet (**`ens33, IP 192.168.242.233`**) and a second connected to the isolated InternalLabNet (**`ens37, IP 192.168.100.20`**), allowing it to communicate with the Velociraptor server. Wireshark will run on both subnets to monitor traffic to both the internet and the INetSim service. Therefore, fire it up before running baseline hunt and before executing the malware (post-infection hunt)

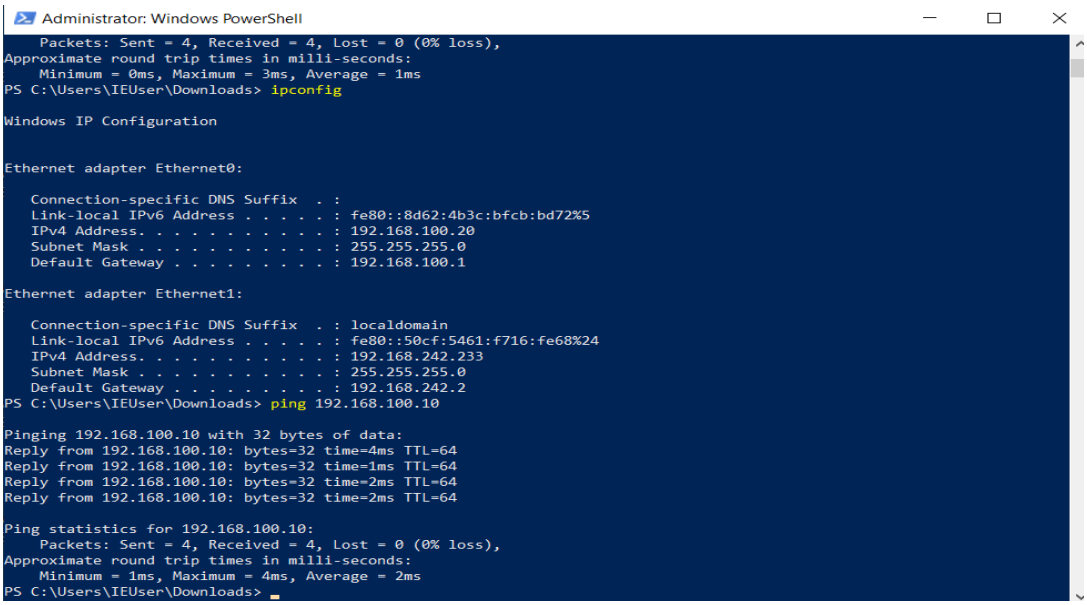


Figure 1.5. Windows Client has 2 net adaptors, one for Internet the other for Ubuntu Velociraptor server

The Ubuntu server was configured with the Uncomplicated Firewall (ufw) to only allow a limited set of INetSim ports from the Windows client, while blocking all other traffic. This ensured that even with internet connectivity, the malware's attempts to communicate with C2 servers were routed through the INetSim service, allowing for a controlled analysis.

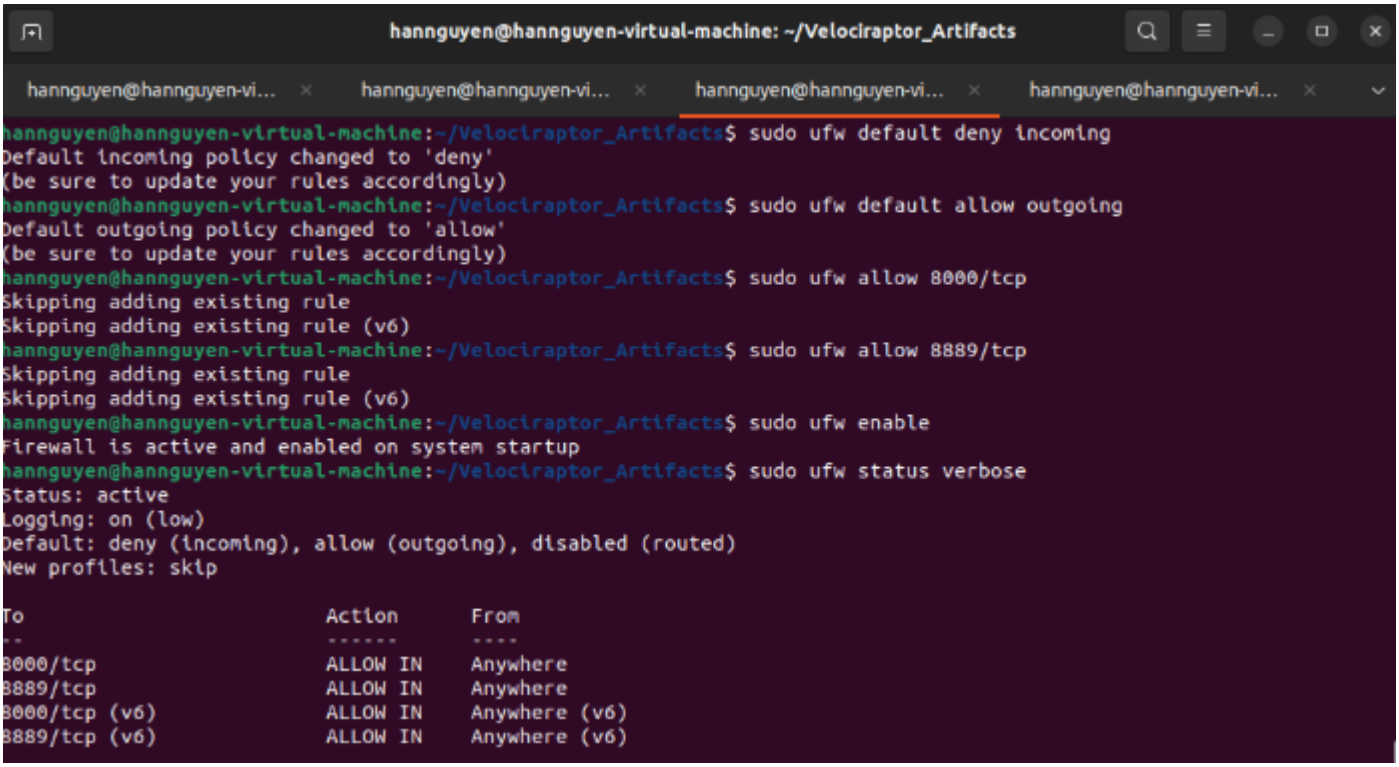


Figure 1.6. UFW Allows Velociraptor, Blocks Everything Else

The extra commands were run to configure Ubuntu's ufw for InetSim

```
# HTTP and HTTPS
sudo ufw allow from 192.168.100.20 to any port 80 proto tcp
sudo ufw allow from 192.168.100.20 to any port 443 proto tcp

# DNS
sudo ufw allow from 192.168.100.20 to any port 53 proto udp

# FTP (some droppers use this)
sudo ufw allow from 192.168.100.20 to any port 21 proto tcp

# SMTP/POP3/IMAP (for mail-related emulation)
sudo ufw allow from 192.168.100.20 to any port 25 proto tcp
sudo ufw allow from 192.168.100.20 to any port 110 proto tcp
sudo ufw allow from 192.168.100.20 to any port 143 proto tcp

# Optional: allow all INetSim ports for dynamic experiments
sudo ufw allow from 192.168.100.20 to any port 1:65535 proto tcp
sudo ufw allow from 192.168.100.20 to any port 1:65535 proto udp
```

```
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw allow from 192.168.100.20 to any port 80 proto tcp
[sudo] password for hannguyen:
Rule added
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw allow from 192.168.100.20 to any port 443 proto tcp
Rule added
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw allow from 192.168.100.20 to any port 53 proto udp
Rule added
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw allow from 192.168.100.20 to any port 21 proto tcp
Rule added
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw allow from 192.168.100.20 to any port 25 proto tcp
Rule added
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw allow from 192.168.100.20 to any port 110 proto tcp
Rule added
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw allow from 192.168.100.20 to any port 143 proto tcp
Rule added
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw allow from 192.168.100.20 to any port 1:65535 proto tcp
Rule added
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw allow from 192.168.100.20 to any port 1:65535 proto udp
Rule added
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
8000/tcp                   ALLOW IN    Anywhere
8889/tcp                   ALLOW IN    Anywhere
8000/tcp                   ALLOW IN    192.168.100.20
80/tcp                     ALLOW IN    192.168.100.20
443/tcp                    ALLOW IN    192.168.100.20
53/udp                     ALLOW IN    192.168.100.20
21/tcp                     ALLOW IN    192.168.100.20
25/tcp                     ALLOW IN    192.168.100.20
110/tcp                    ALLOW IN    192.168.100.20
143/tcp                    ALLOW IN    192.168.100.20
1:65535/tcp                ALLOW IN    192.168.100.20
1:65535/udp                ALLOW IN    192.168.100.20
8000/tcp (v6)              ALLOW IN    Anywhere (v6)
8889/tcp (v6)              ALLOW IN    Anywhere (v6)

hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$
```

Figure 1.7. The final ufw configurations allowing Velociraptor, some INetSim ports, block all else

2.6.2. VELOCIRAPTOR AGENT DEPLOYMENT

To collect data from the Windows 10 client, first we need to install and configure the Velociraptor Server on Ubuntu VM (Basic Velociraptor Server and Client Setup, How-to Guide [7]).

```
Terminal                    Jul 31 15:04
hannguyen@hannguyen-virtual-machine: ~
hannguyen@hannguyen-virtual-machine: ~   hannguyen@hannguyen-virtual-machine: ~   hannguyen@hannguyen-virtual-machine: ~
hannguyen@hannguyen-virtual-machine:~$ sudo systemctl status velociraptor_server
● velociraptor_server.service - Velociraptor server
     Loaded: loaded (/etc/systemd/system/velociraptor_server.service; enabled; vendor preset: ena>
     Active: active (running) since Wed 2025-07-30 22:22:57 EDT; 6s ago
   Main PID: 13290 (velociraptor.bi)
      Tasks: 19 (limit: 19038)
     Memory: 77.3M
        CPU: 6.375s
     CGroup: /system.slice/velociraptor_server.service
             ├─13290 /usr/local/bin/velociraptor.bin --config /etc/velociraptor/server.config.yam>
             └─13299 /usr/local/bin/velociraptor.bin --config /etc/velociraptor/server.config.yam>

Jul 30 22:22:57 hannguyen-virtual-machine systemd[1]: Started Velociraptor server.
```

Figure 2.1: Start the already configured velociraptor server – check status to make sure velociraptor_server.service is "active (running)"

Generate a Velociraptor agent was deployed. The agent was downloaded from a simple Python HTTP server hosted on the Ubuntu machine (at `http://192.168.100.10:9999`) to simulate a common method of file transfer and to capture this activity as a part of the analysis.

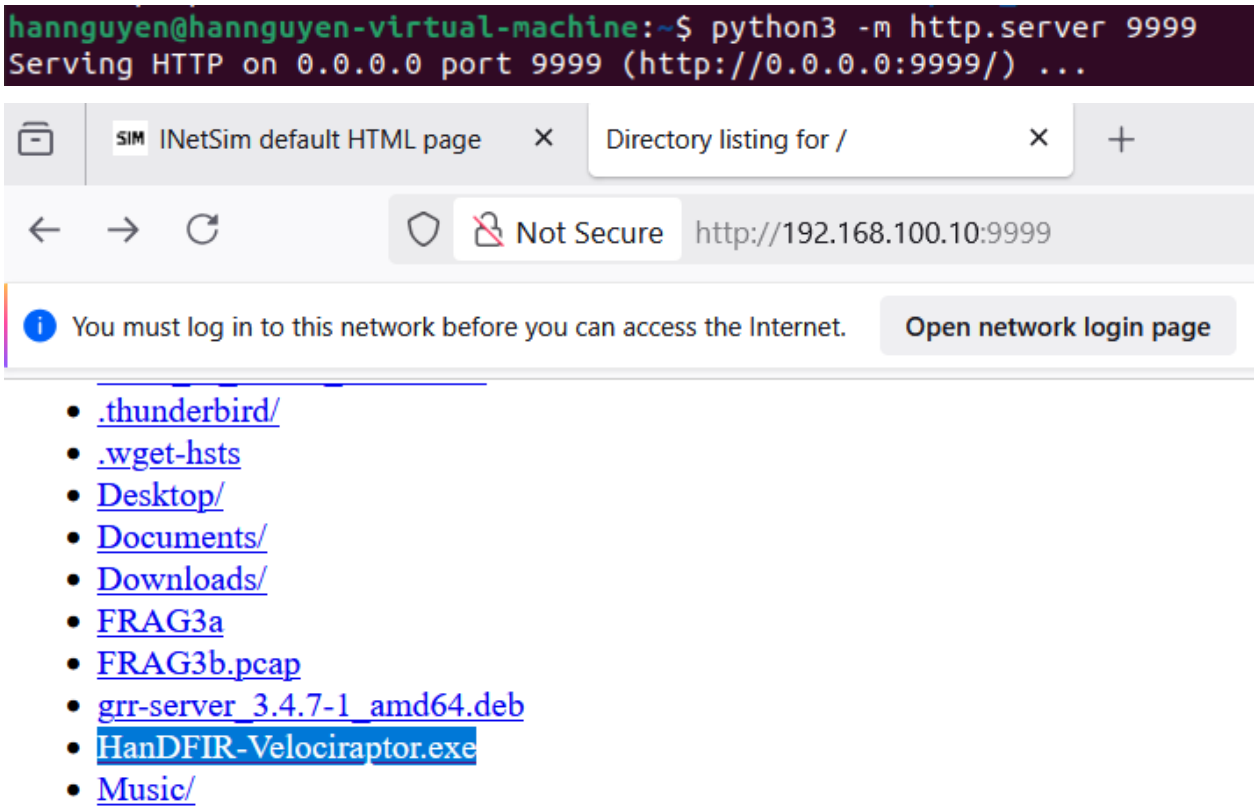


Figure 2.2: Web Browser on Windows VM for Downloading the Velociraptor Agent

The downloaded file, HanDFIR-Velociraptor.exe, was then run with **administrative privileges** on the Windows 10 client, a necessary step for comprehensive artifact collection.

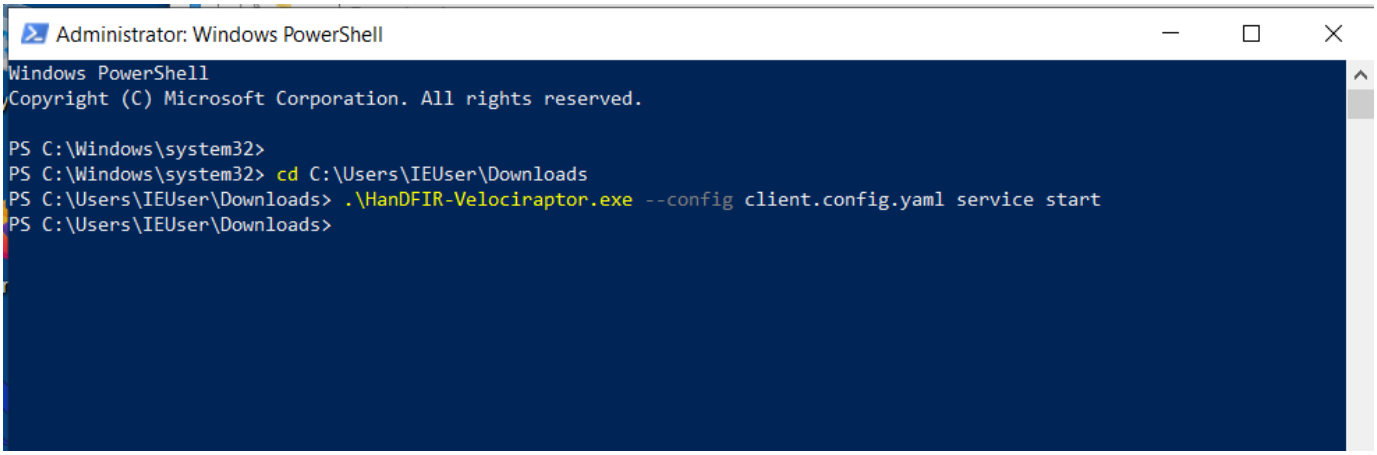


Figure 2.3: PowerShell Commands to Install Velociraptor Client Service. This shows the commands used to navigate to the download directory and start the Velociraptor client as a service.

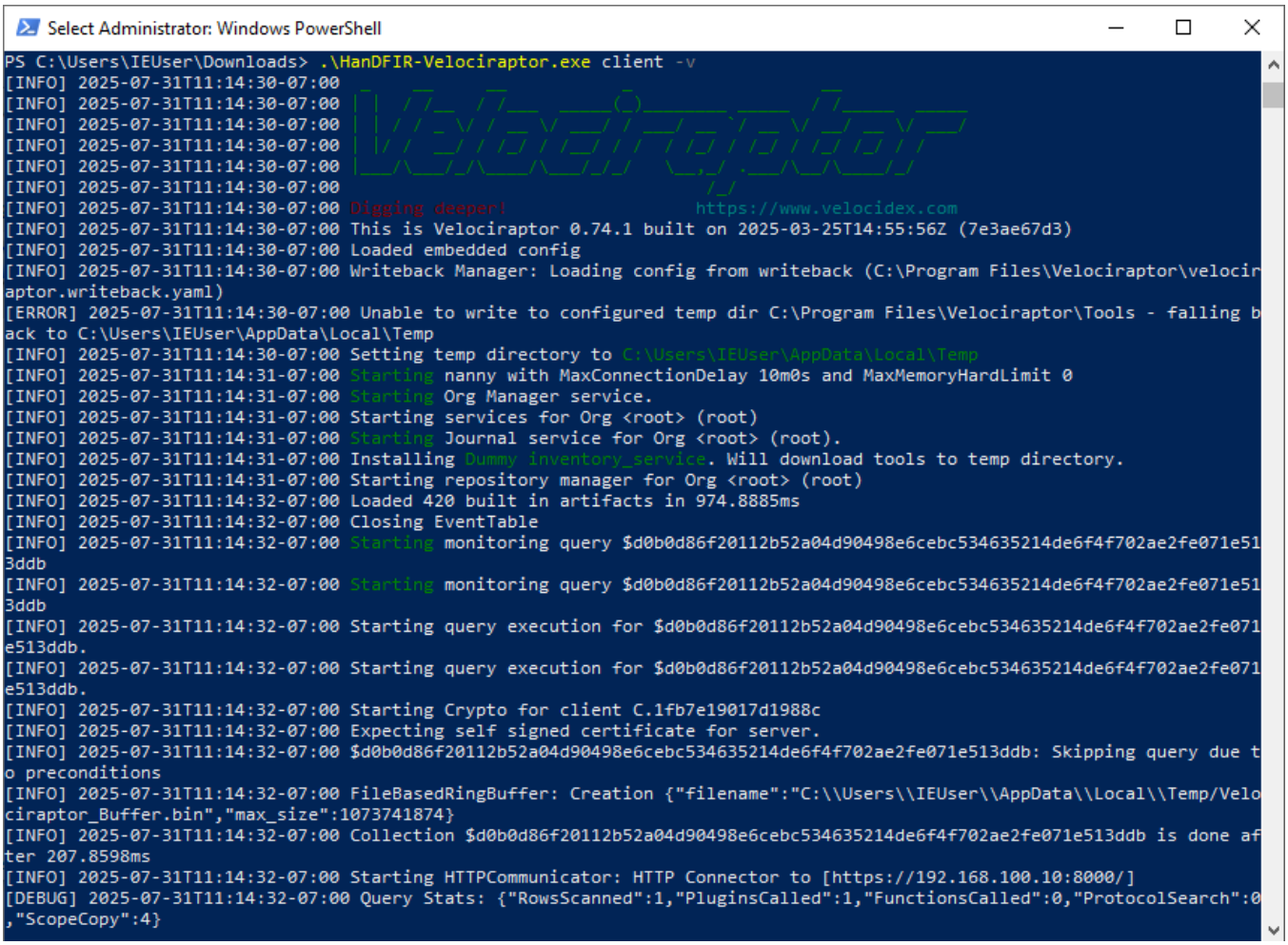


Figure 2.4: Alternative way to start client server - this is a log output detailing the client's startup process, including loading the configuration and starting the HTTP communicator (-v verbose mode).

Download QuasarRAT sample from **Malware Bazaar** and executed manually in a controlled, isolated virtual environment using the IEUser account to simulate user behavior in a Windows environment; therefore, if you are interested in dissecting the malware, please do so in a sandbox environment (isolated) with preventative measures in place.

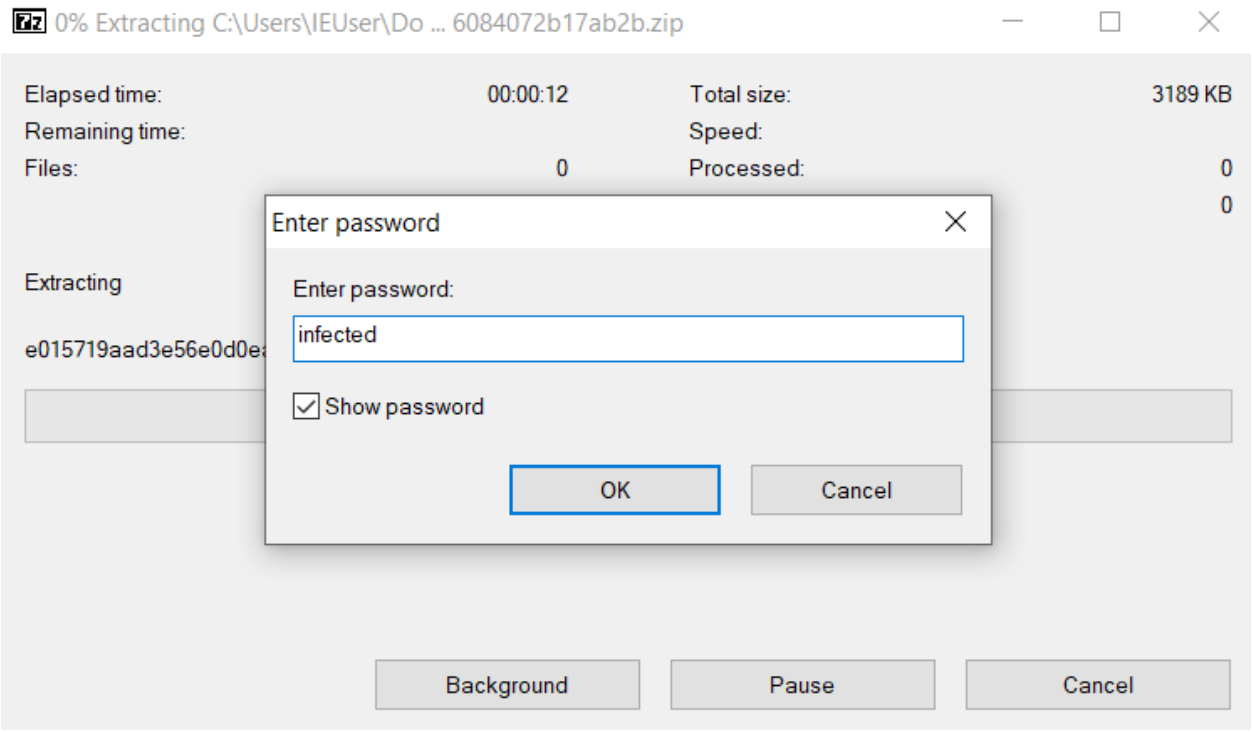


Figure 2.5: Enter "infected" as password provided by malwareBazaar to extract QuasarRAT sample

For visibility and controlled observation, the analyst deliberately named the malware sample quasarRAT-sample.exe prior to execution. This was done to: Simplify post-execution tracking across forensic artifacts; facilitate correlation between process execution, registry traces, and file system activity; validate Velociraptor's detection capabilities within the Amcache artifact collector.

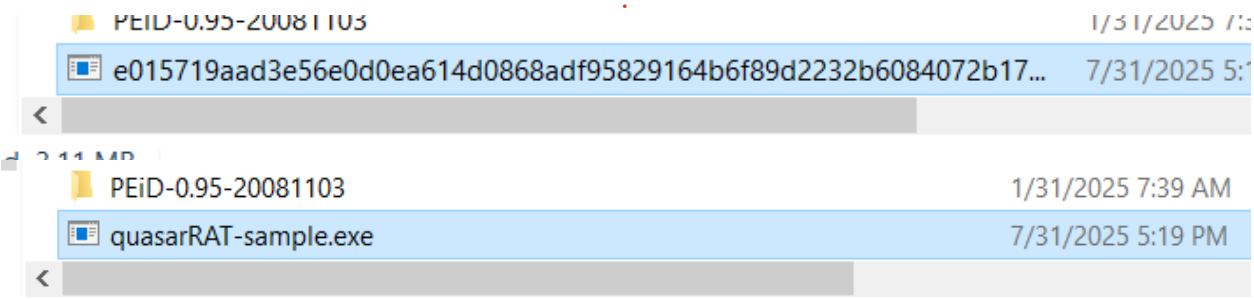


Figure 2.6: Change default filename (SHA256 hash) to quasarRAT-sample.exe for easier tracking

2.6.3. WHY USE VELOCIRAPTOR IN CONJUCTION WITH INETSIM IN MALWARE ANALYSIS

INetSim is a crucial tool for safely emulating internet services within an isolated lab. Without a response from DNS, HTTP, or other services, malware might fail to unpack, install, or run its payload; it might skip C2 beaconing; or it might stay dormant, detecting there is no internet connection.

INetSim tricks malware into "thinking" it's online, allowing for the capture of IOCs, the monitoring of C2 attempts and dropped files, and the study of full malware behavior without risking a real internet connection. **It complements Wireshark by providing actual responses to requests, which can trigger more complex malware behaviors that a simple packet capture might miss.** By running both tools in parallel across segmented networks, we ensure full visibility into malware behavior, both in terms of intent and actual data transmission.

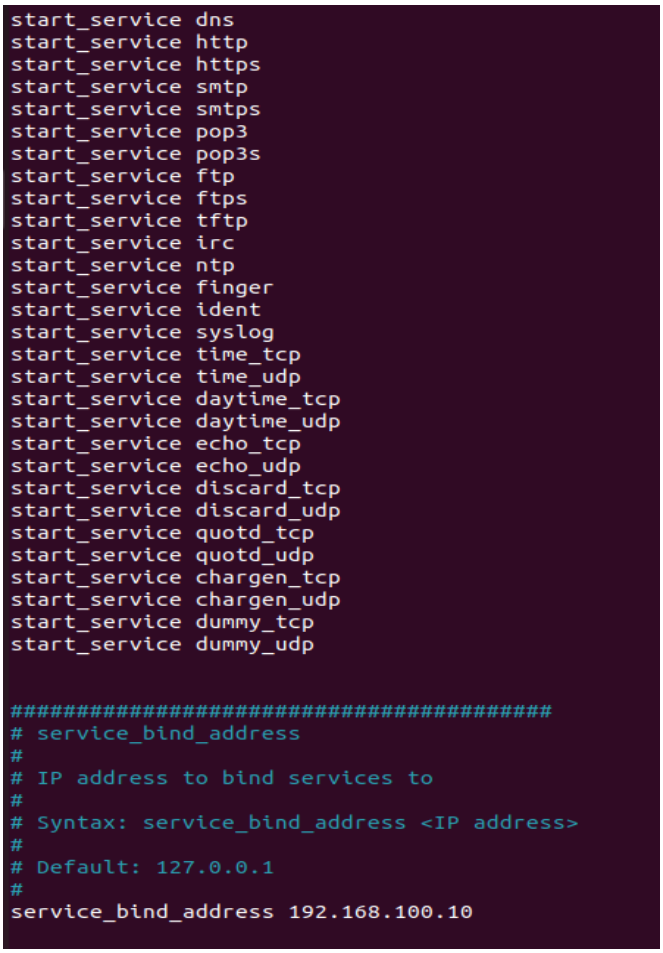


Figure 3.1: configuration file for INetSim, showing started services & service_bind_address set to the server's IP

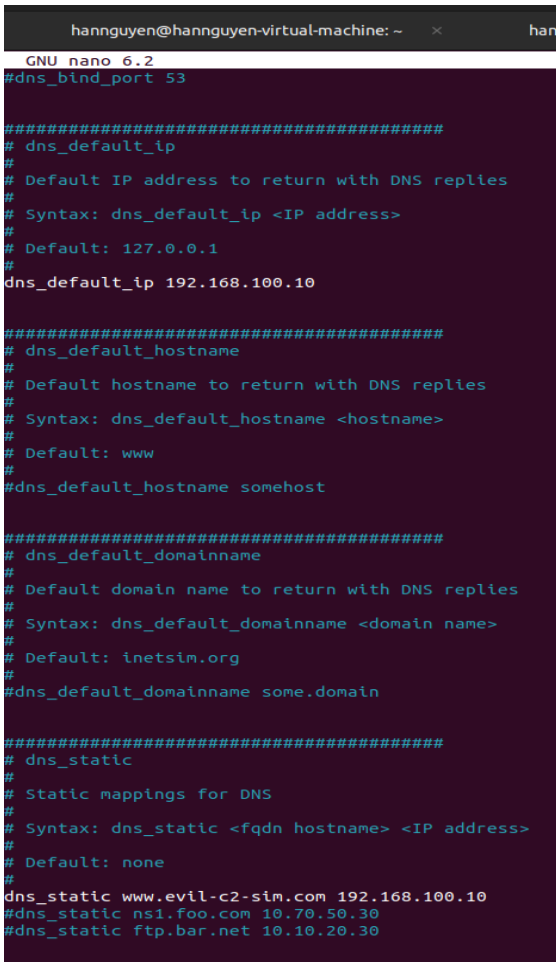


Figure 3.2. INetSim DNS Configuration

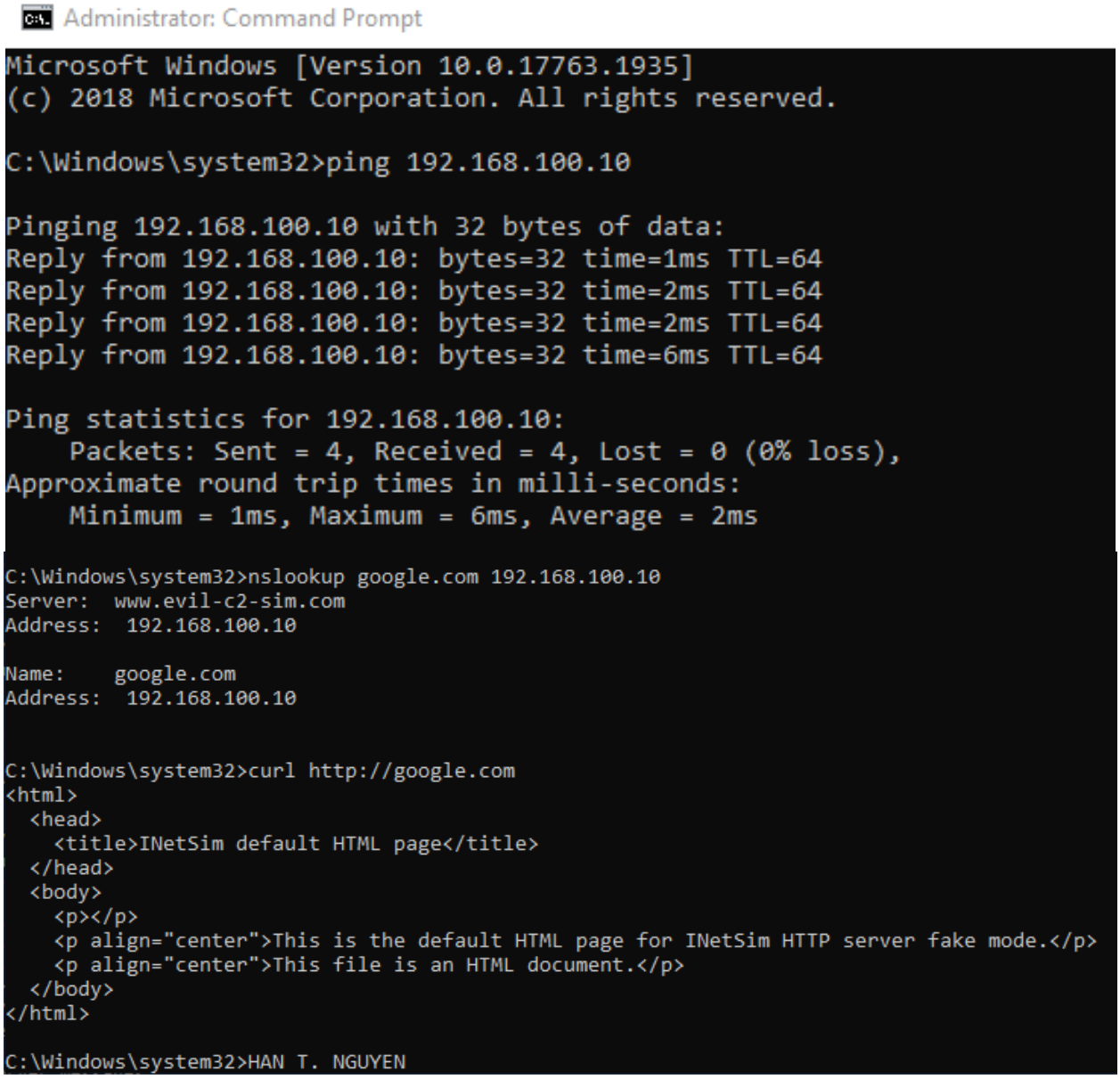


*Figure 3.3. Console output for ping and nslookup demonstrates a successful ping to the server IP and shows nslookup resolving google.com to the configured **INetSim server IP.***

2.6.4. NOW BACK TO VELOCIRAPTOR SERVER (UBUNTU)

We'll go over the steps to create and launch a hunt on Windows 10 client. To get started, open a web browser on the server and navigate to the Velociraptor GUI at `192.168.242.224:8889`. Log in with the credentials specified in the `velociraptor.config.yaml` server file. Once logged in, confirm that the client machine, labeled "**Windows 10**", is showing as connected and online. For future scalability and ease of querying, it is

good practice to use multiple descriptive labels (e.g., windows10, dev, sales, finance, IT) for different department machines.

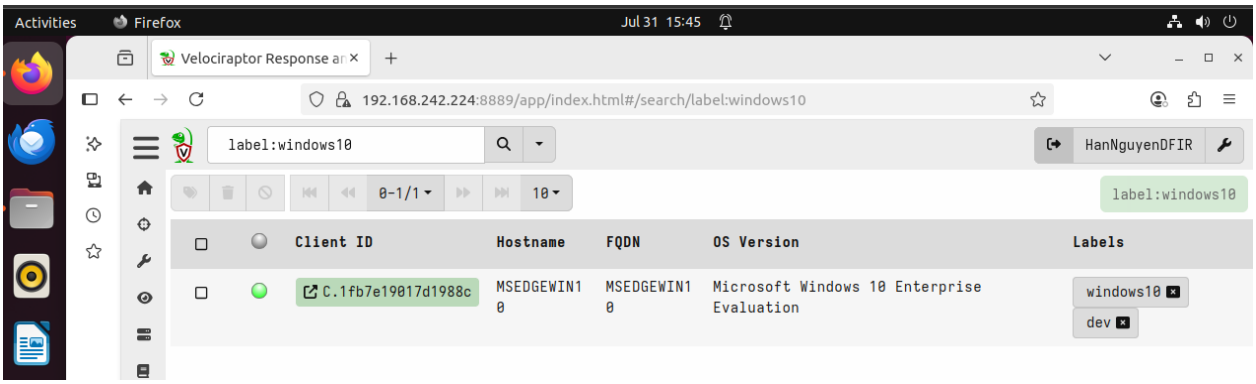


Figure 4.1: Velociraptor web interface shows a connected Windows 10 client with the label "windows10"

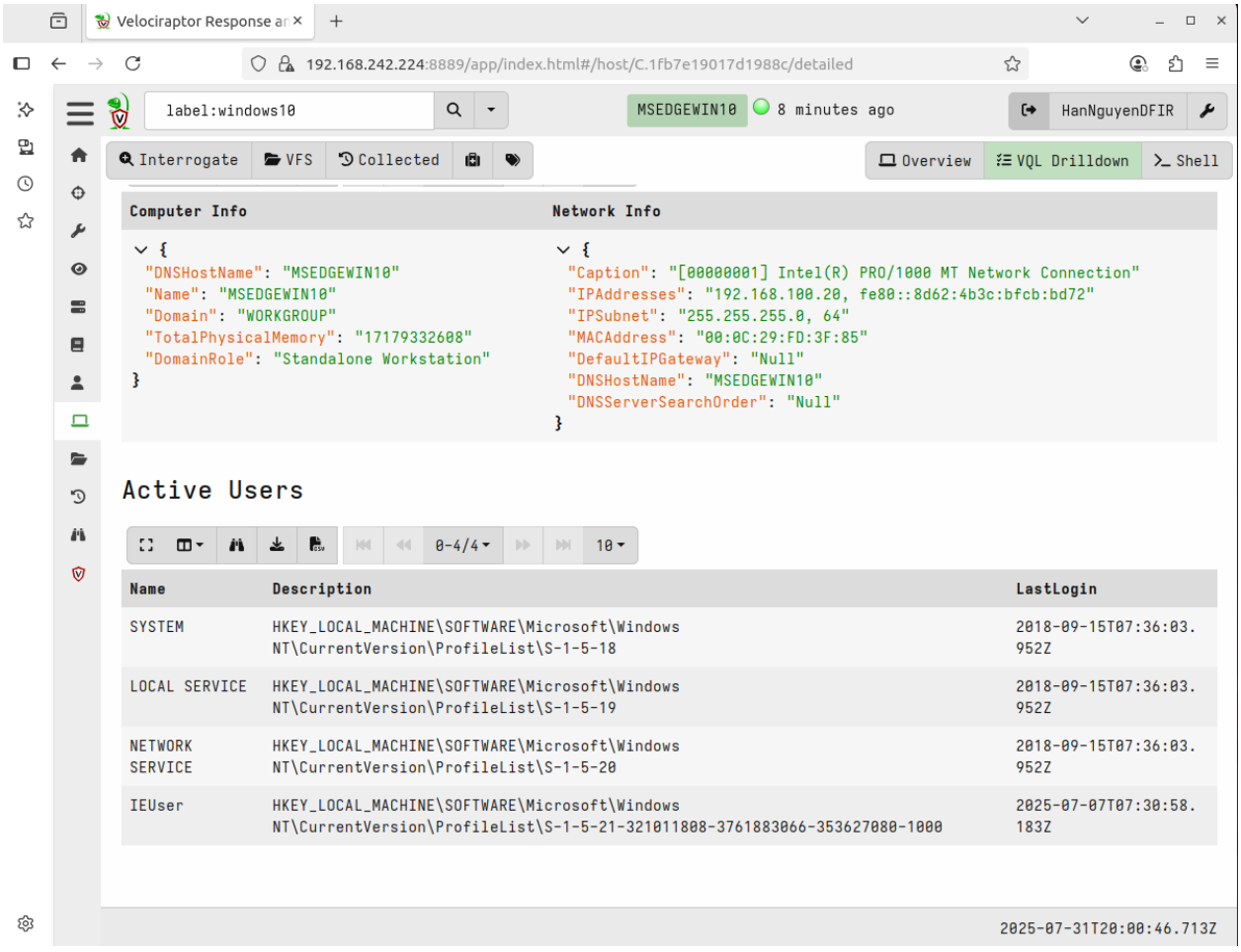


Figure 4.2: Velociraptor GUI Showing details about the connected client, including its Computer Info and Network Info

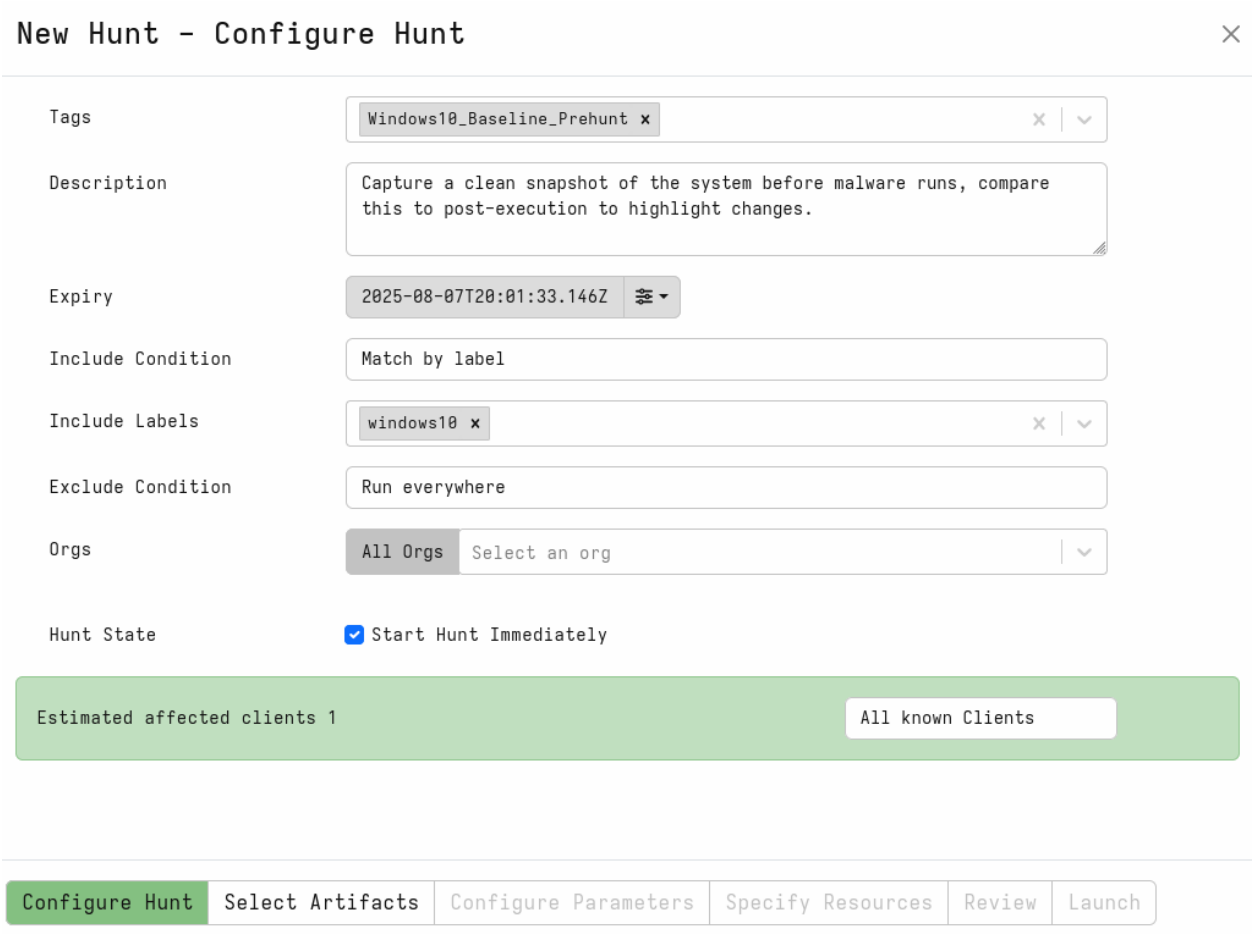


Figure 4.3: Click on the Windows 10 client and create new base-line hunt

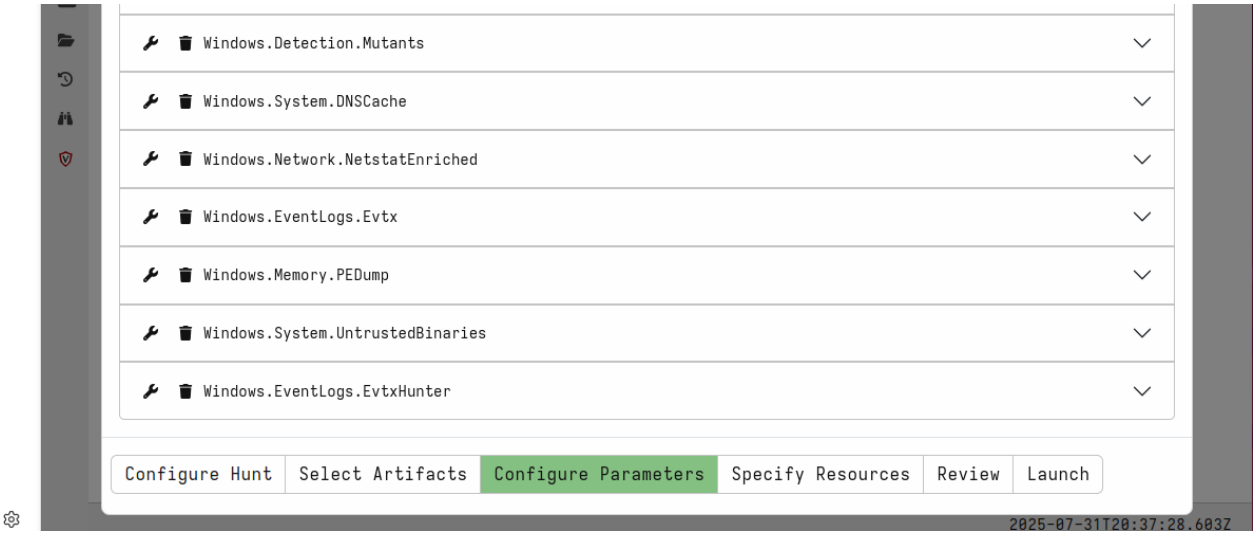


Figure 4.4: Select prioritized artifacts (detailed explanation for choices in Section 2.5)

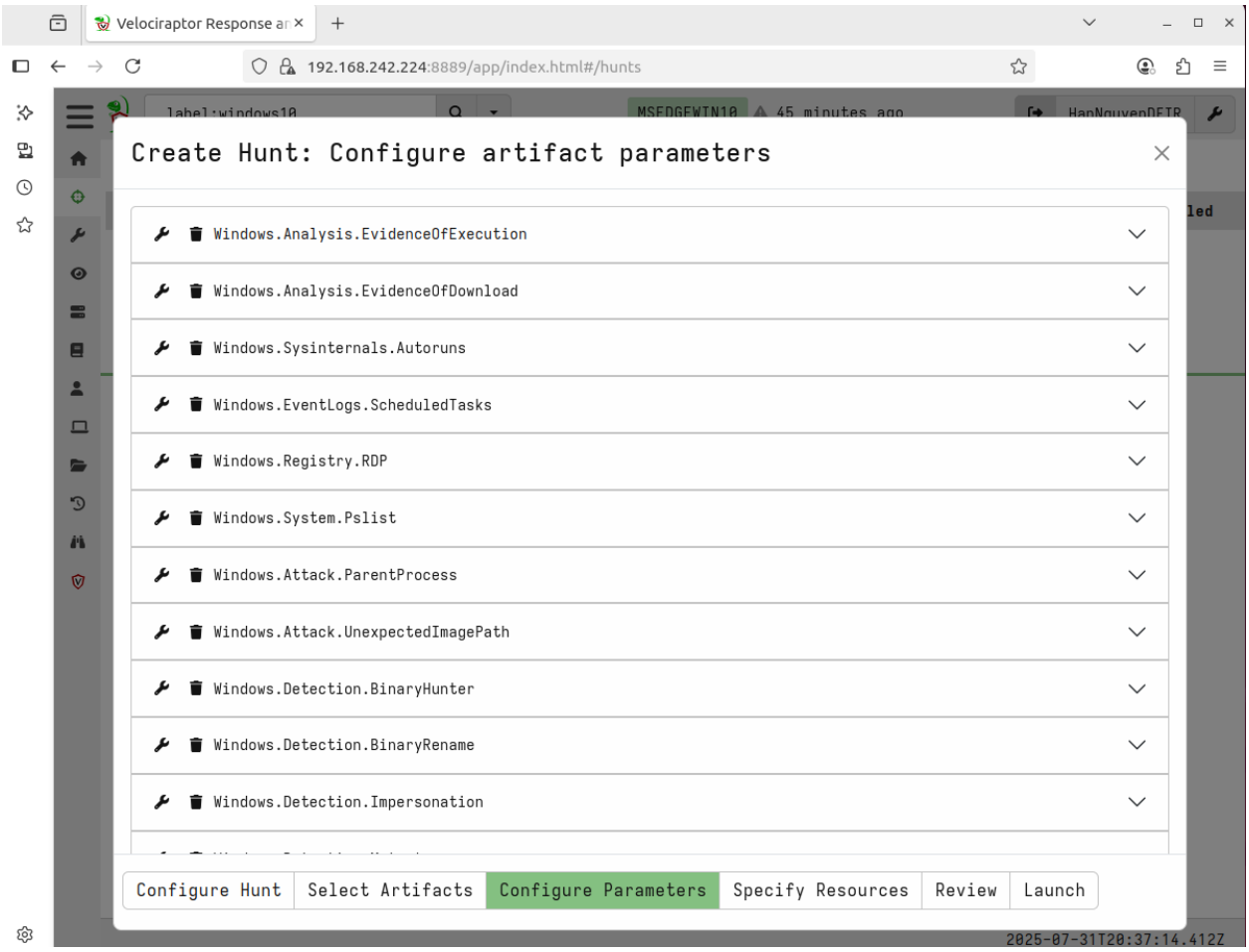


Figure 4.4: Select above artifacts (continued - explanation above, Section 2.5)

Leave parameters as default and specify resource limits - just update Max Execution Time to 1200 and Max MB Uploaded to 2048. Optionally check "Urgent" to make the hunt go **now** instead of waiting in the queue.

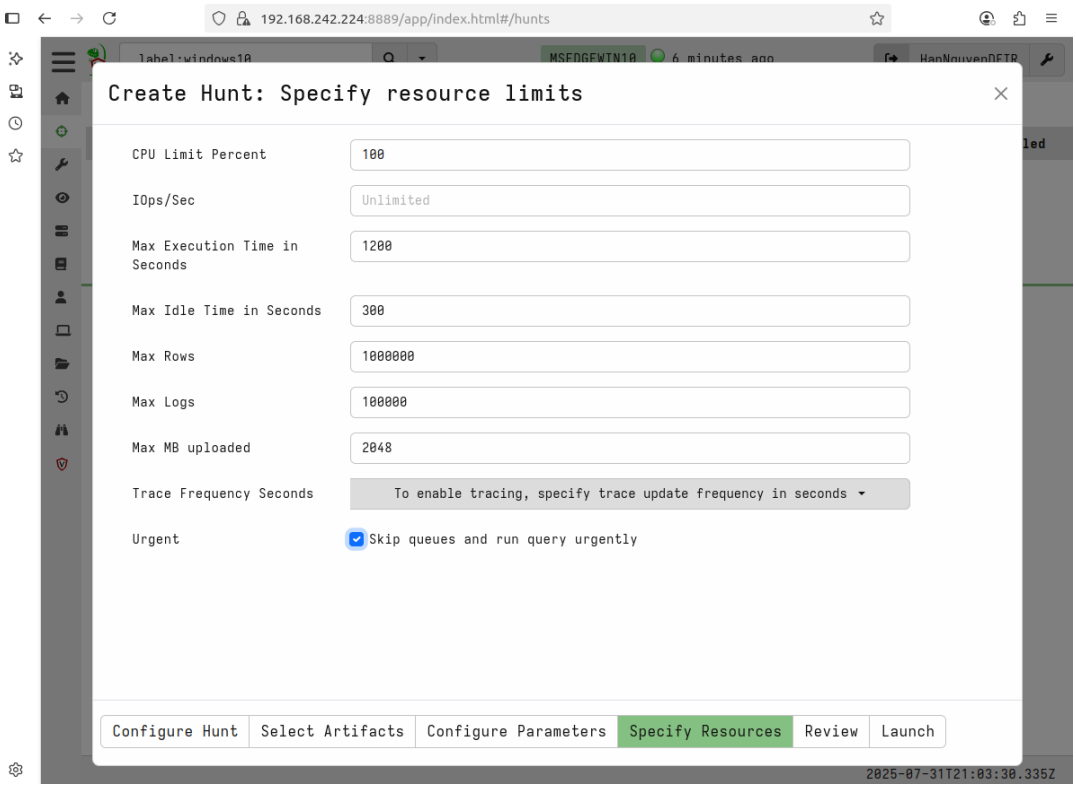


Figure 4.5: Specify Resource limits for Baseline Hunt

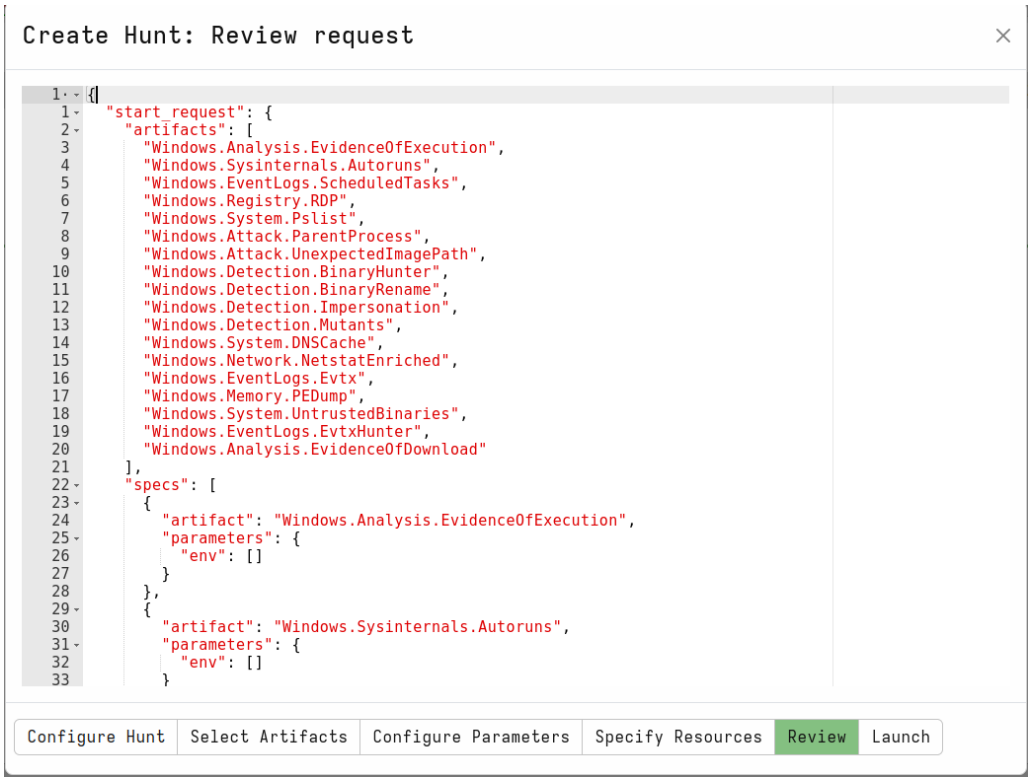


Figure 4.6: Review and make sure all desired/prioritized artifacts are there (baseline hunt)

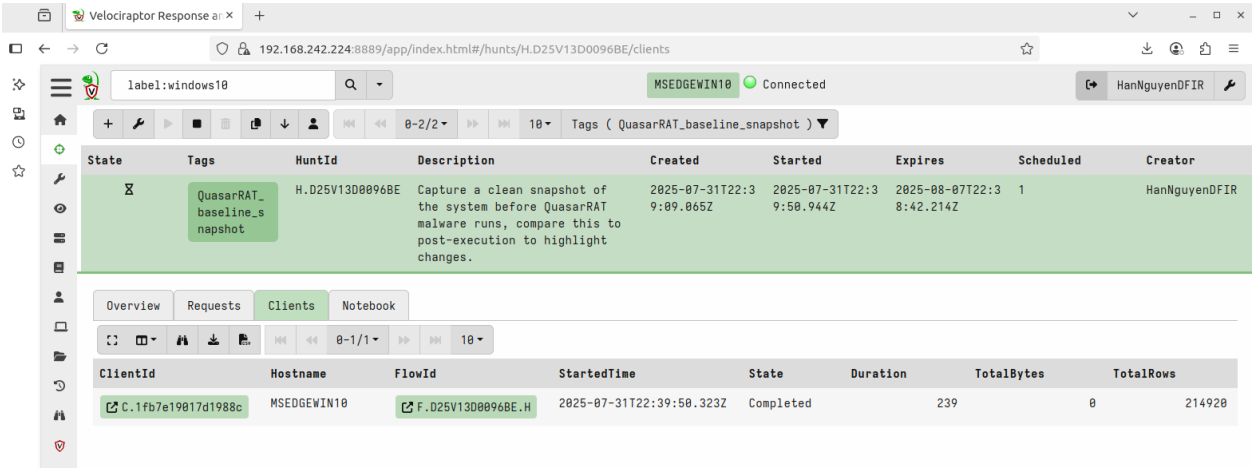


Figure 4.7: Launch the hunt on connected client (Windows 10 – Healthy baseline, pre-infection)

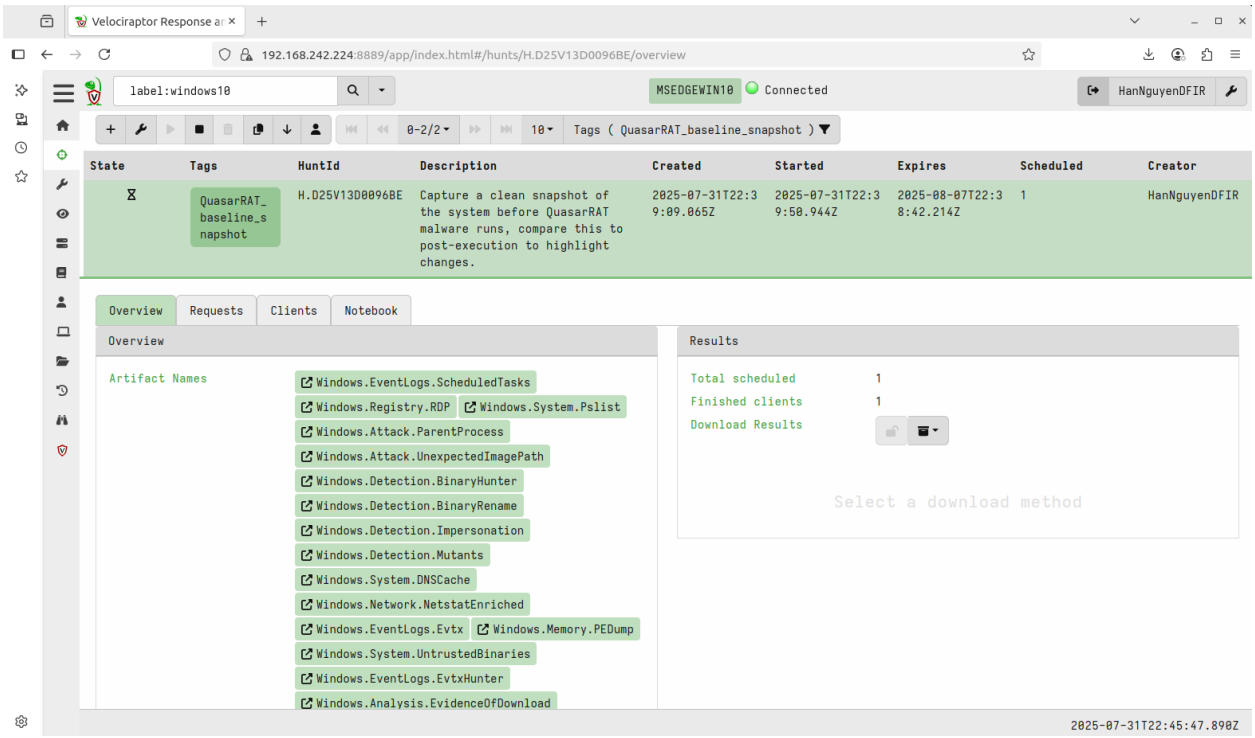


Figure 4.8: Baseline hunt complete, and we can download and view the report (either CSV or JSON format)

Repeat the same steps and same artifacts selection for the **post-infection threat hunt** for **differential analysis** (identifying anomalies such as new processes, program downloaded, executed, new traffic to unknown Ips, etc.). In other words, this hunt captures the system state after executing the QuasarRAT variant (SHA-256: e015719aad3e56e0d0ea614d0868adf95829164b6f89d2232b6084072b17ab2b). It is intended to collect artifacts that reflect execution footprints, persistence mechanisms, network activity, and

potential lateral movement. This snapshot will be compared with the pre-infection baseline to identify key IoCs and anomalous changes in the system.

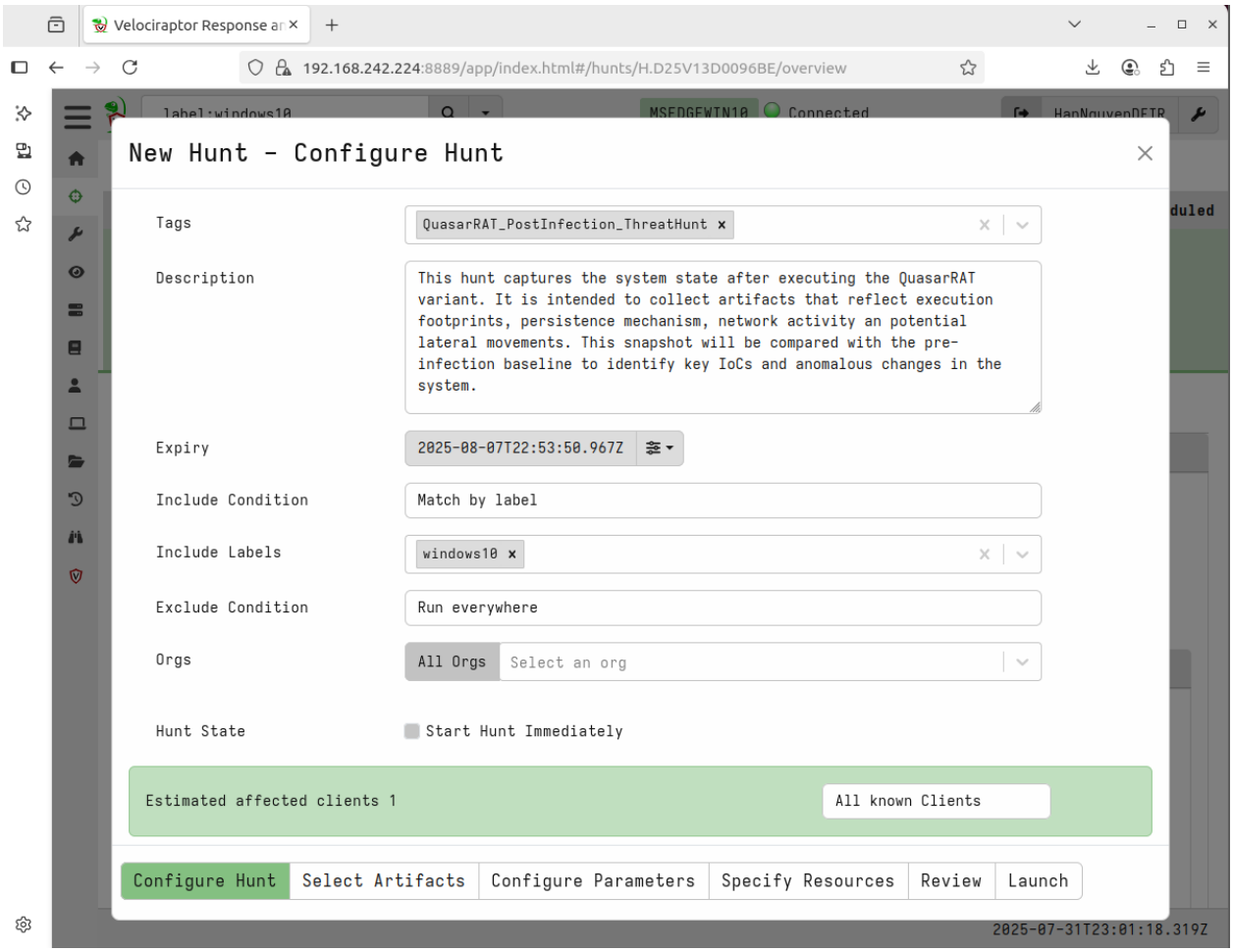


Figure 4.9: Create new hunt called QuasarRAT_PostInfection_ThreatHunt

Once again, keep All Artifacts the same between baseline and post-infection, that's the only way to do an apples-to-apples comparison (at least for Analysis.EvidenceOfExecution/Amcache).

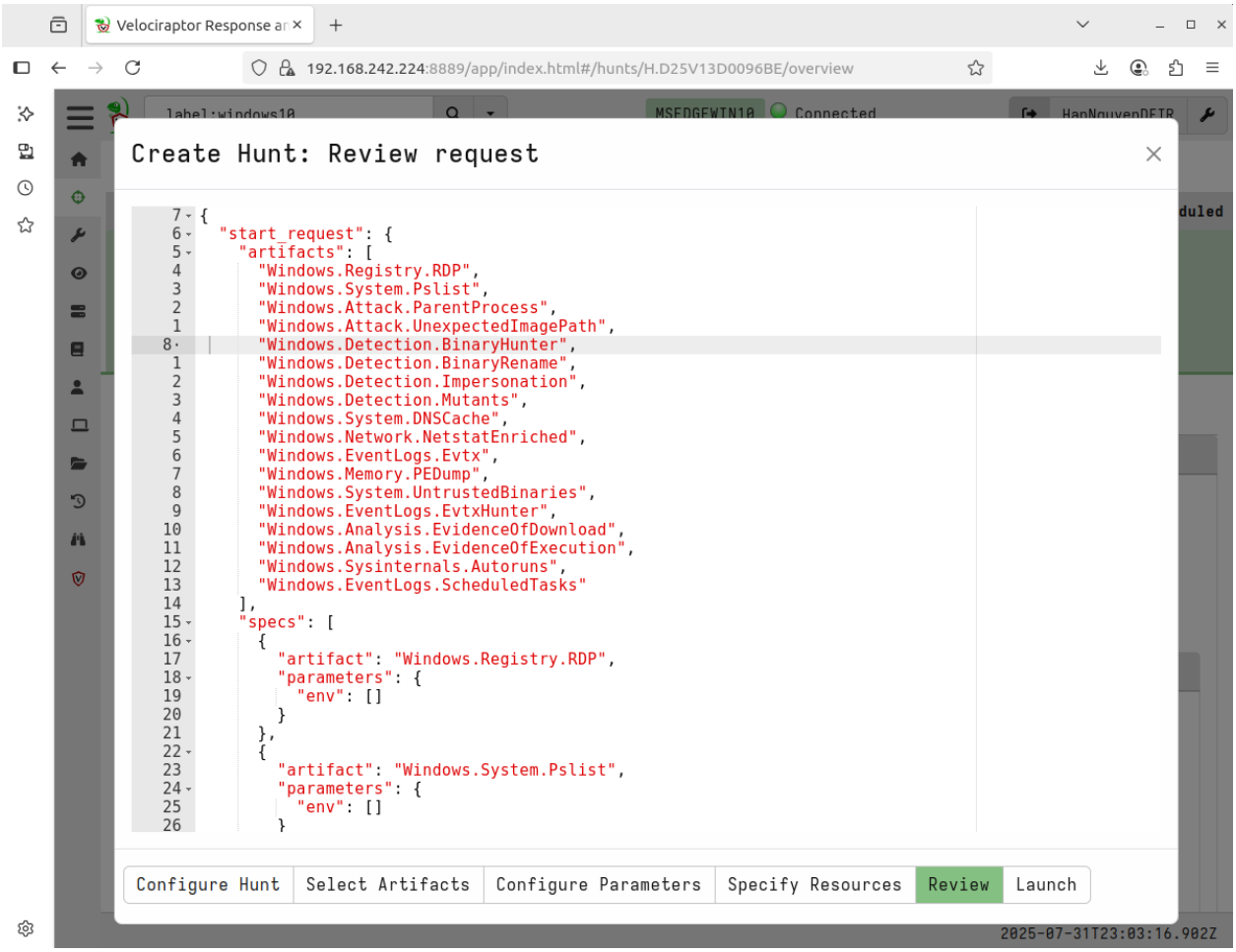


Figure 4.10: QuasarRAT_PostInfection_ThreatHunt should have the same artifacts as baseline hunt

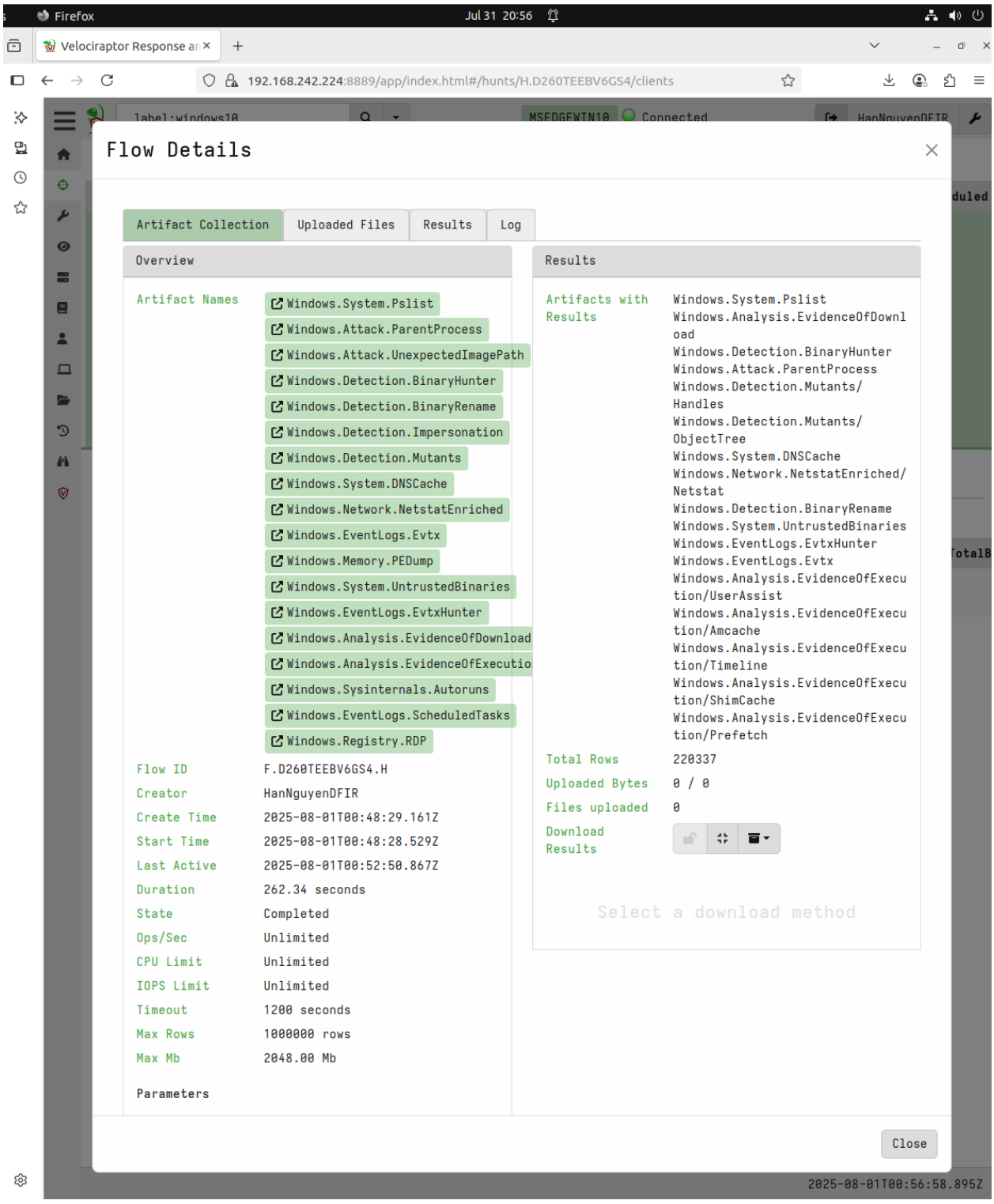


Figure 4.11: QuasarRAT_PostInfection_ThreatHunt example hunt showing the resulting artifacts collected

III. VELOCIRAPTOR HOST ANALYSIS

During the two hunt cycles (pre- and post-infection), I have run the full suite of artifacts, as detailed in **Table 1**, to establish a thorough baseline and identify malicious activity. For the purposes of this report, I will only be highlighting the artifacts that returned actionable results and clear IoCs. This approach ensures clarity and directs attention to the most critical findings.3.

3.1. EVIDENCE OF EXECUTION VIA AMCACHE.HVE

3.1.1. HOW IT WORKS

Windows.Analysis.EvidenceOfExecution

Type: client

In many investigations it is useful to find evidence of program execution.

This artifact combines the findings of several other collectors into an overview of all program execution artifacts. The associated report walks the user through the analysis of the findings.

Source UserAssist

```
1 SELECT * FROM Artifact.Windows.Registry.UserAssist()
2
```

Source Amcache

```
1 SELECT * FROM Artifact.Windows.Detection.Amcache()
2
```

Source Timeline

```
1 SELECT * FROM Artifact.Windows.Forensics.Timeline()
2
```

Source ShimCache

```
1 SELECT * FROM Artifact.Windows.Registry.AppCompatCache()
2
```

Source Prefetch

```
1 SELECT * FROM Artifact.Windows.Forensics.Prefetch()
2
```

Source Recent Apps

```
1 SELECT * FROM Artifact.Windows.Forensics.RecentApps()
2
```

Figure 5.1: Windows.Analysis.EvidenceOfExecution Queries

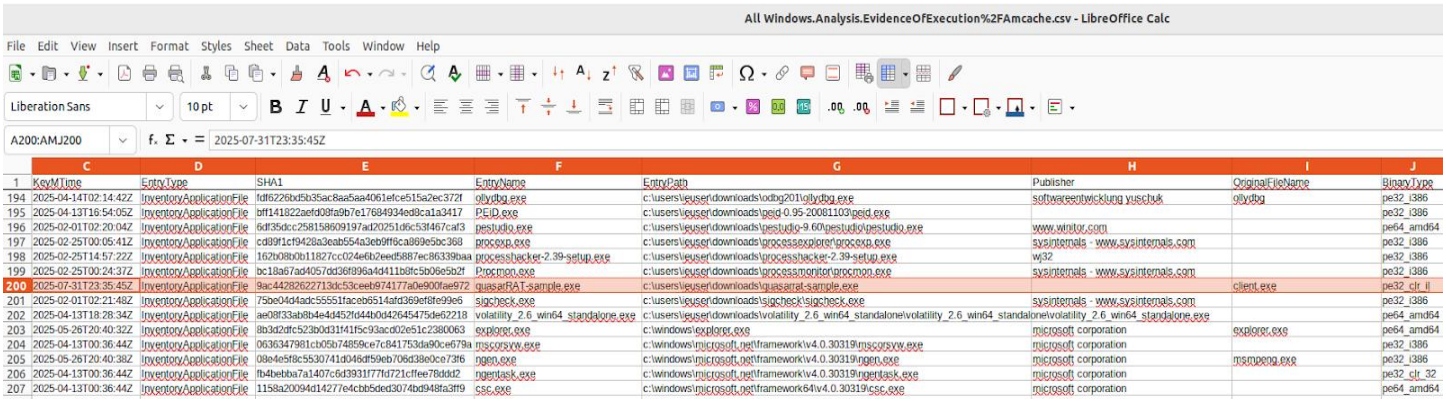
To confirm the execution of the malware, a targeted Velociraptor query was run to collect and parse the Amcache.hve artifact. This artifact, part of Windows' Application Compatibility framework, records metadata about executed programs, including their full path and SHA-1 hash

[2]. It facilitates the crucial task of identifying program execution by aggregating data from several key collectors. It integrates information from sources such as UserAssist, Amcache, Timeline, ShimCache, Prefetch, and Recent Apps into a single, comprehensive overview. The Amcache.csv diff shows modified registry entries related to Windows' Amcache.hve, which logs program execution. These entries are loaded from **C:\Windows\appcompat\Programs\Amcache.hve**

Result after each hunt will be saved automatically in a csv file (or json) called **All Windows.Analysis.EvidenceOfExecution%2FAmcache.csv**

3.1.2. WINDOWS. ANALYSIS. EVIDENCEOFEXECUTION (PHASE 1 – ISOLATED ENVIRONMENT)

The initial investigation was conducted in a securely isolated environment with INetSim. This phase focused on the initial infection and persistence discovery.



	C	D	E	F	G	H	I	J
1	KeyMTime	EntryType	SHA1	EntryName	EntryPath	Publisher	OriginalFileName	BinaryType
194	2025-04-14T02:14:42Z	InventoryApplicationFile	fdf6226bd5b35ac8aa5aa4061efce515a2ec372f	ollydbg.exe	c:\users\ieuser\downloads\odbg201\ollydbg.exe	softwareentwicklung yuschuk	ollydbg	pe32_i386
195	2025-04-13T16:54:05Z	InventoryApplicationFile	bff141822aefd08fa9b7e17684934ed9ca1a3417	PEID.exe	c:\users\ieuser\downloads\peid-0.95-20081103\peid.exe			pe32_i386
196	2025-02-01T02:20:04Z	InventoryApplicationFile	6df35dcc25815f609197ad20251d6c53f467caf3	pestudio.exe	c:\users\ieuser\downloads\pestudio-9.60\pestudio\pestudio.exe	www.winitor.com		pe64_amd64
197	2025-02-25T00:05:41Z	InventoryApplicationFile	cd89f1cf9428a3eab554a3eb9ff6ca869e5bc368	procexp.exe	c:\users\ieuser\downloads\processexplorer\procexp.exe	sysinternals - www.sysinternals.com		pe32_i386
198	2025-02-25T14:57:22Z	InventoryApplicationFile	162b08b0b11827cc024e6b2eed5887ec86339baa	processhacker-2.39-setup.exe	c:\users\ieuser\downloads\processhacker-2.39-setup.exe	wj32		pe32_i386
199	2025-02-25T00:24:37Z	InventoryApplicationFile	bc18a67ad4057dd36f896a4d411b8fc5b06e5b2f	Procmon.exe	c:\users\ieuser\downloads\procmon\procmon.exe	sysinternals - www.sysinternals.com		pe32_i386
200	2025-07-31T23:35:45Z	InventoryApplicationFile	9ac44282622713dc53ceeb974177a0e900fae972	quasarRAT-sample.exe	c:\users\ieuser\downloads\quasarrat-sample.exe		client.exe	pe32_clr_il
201	2025-02-01T02:21:48Z	InventoryApplicationFile	75be04d4adc55551faceb6514afd369ef8fe99e6	sigcheck.exe	c:\users\ieuser\downloads\sigcheck\sigcheck.exe	sysinternals - www.sysinternals.com		pe32_i386
202	2025-04-13T18:28:34Z	InventoryApplicationFile	ae08f33ab8b4e4d452fd44b0d42645475de62218	volatility_2.6_win64_standalone.exe	c:\users\ieuser\downloads\volatility_2.6_win64_standalone\volatility_2.6_win64_standalone.exe			pe64_amd64
203	2025-05-26T20:40:32Z	InventoryApplicationFile	8b3d2dfc523b0d31f41f5c93acd02e51c2380063	explorer.exe	c:\windows\explorer.exe	microsoft corporation	explorer.exe	pe64_amd64
204	2025-04-13T00:36:44Z	InventoryApplicationFile	0636347981cb05b74859ce7c841753da90ce679a	mscorsvw.exe	c:\windows\microsoft.net\framework\v4.0.30319\mscorsvw.exe	microsoft corporation		pe32_i386
205	2025-05-26T20:40:38Z	InventoryApplicationFile	08e4e5f8c5530741d046df59eb706d38e0ce73f6	ngen.exe	c:\windows\microsoft.net\framework\v4.0.30319\ngen.exe	microsoft corporation	msmpeng.exe	pe32_i386
206	2025-04-13T00:36:44Z	InventoryApplicationFile	fb4bebba7a1407c6d3931f77fd721cffee78ddd2	ngentask.exe	c:\windows\microsoft.net\framework\v4.0.30319\ngentask.exe	microsoft corporation		pe32_clr_32
207	2025-04-13T00:36:44Z	InventoryApplicationFile	1158a20094d14277e4cbb5ded3074bd948fa3ff9	csc.exe	c:\windows\microsoft.net\framework64\v4.0.30319\csc.exe	microsoft corporation		pe64_amd64

Figure 5.2. All Windows.Analysis.EvidenceOfExecution/Amcache.csv Findings

From the Amcache analysis, the following evidence of execution was found:

a. **Filename:** quasarRAT-sample.exe
b. **Full Path:** C:\Users\IEUser\Downloads\quasarRAT-sample.exe
c. **OriginalFileName Metadata:** client.exe
d. **Binary Type:** pe32_clr_il (.NET executable)
e. **SHA1:** 9ac44282622713dc53ceeb974177a0e900fae972
f. **Detection Source:** Amcache.hve (Windows Application Compatibility Cache)
g. **Detection Tool:** Velociraptor DFIR Agent
h. **System:** MSEDGEWIN10 (Quarantine VM)

The presence of the path **C:\USERS\IEUSER\APPDATA\ROAMING\WAVE GOOGLE\JACKES.EXE** in the **Amcache data** provides definitive evidence of the malware's execution on the system. Amcache is populated during the application launch process, making this a reliable indicator of execution rather than mere presence on disk. The associated SHA-1 hash is a critical IOC that can be used for future detection and threat intelligence lookups.

I have created a useful Python tool called **dff_util_execProof.py** to further identify suspicious file hash that interacted with the registry - taking the baseline snapshot and the post infection hunt CSVs as inputs. This Python script compares these two Velociraptor Amcache artifact CSV files - one from a clean (baseline) system and one from a potentially infected system - to detect new or suspiciously modified executables that may have been introduced during malware execution (e.g., QuasarRAT). It helps threat hunters and DFIR analysts quickly isolate potentially malicious binaries by spotlighting new file executions in shady locations after an infection. Please refer to the **Appendix – Section 1.1** for more information on source code

```
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo nano dff_util_execProof.py
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$ sudo -E python3 dff_util_execProof.py
Velociraptor Amcache Differential Scanner
========================================
Enter the full path to the baseline (before infection) Amcache CSV file: /home/hannguyen/Velociraptor_Artifacts/Win
dows10BaselineCSVs/All Windows.Analysis.EvidenceOfExecution%2FAmcache.csv
Enter the full path to the infected (after infection) Amcache CSV file: /home/hannguyen/Velociraptor_Artifacts/afte
r_infection/All Windows.Analysis.EvidenceOfExecution%2FAmcache.csv
[+] Successfully loaded /home/hannguyen/Velociraptor_Artifacts/Windows10BaselineCSVs/All Windows.Analysis.EvidenceO
fExecution%2FAmcache.csv (699 rows)
[+] Successfully loaded /home/hannguyen/Velociraptor_Artifacts/after_infection/All Windows.Analysis.EvidenceOfExecu
tion%2FAmcache.csv (700 rows)
[~] Found 13 suspicious entries in baseline..
[~] Found 14 suspicious entries in infectedsample...
[~] New suspicious hashes: 1hes...
[~] Suspicious but existing hashes: 13
[+] New suspicious entries saved to new_suspicious_hashes.csv
[+] Existing suspicious entries saved to existing_suspicious_hashes.csv
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$
```

*Figure 5.3. The GUI of **dff_util_execProof.py** console tool for easier amcache differentiation*

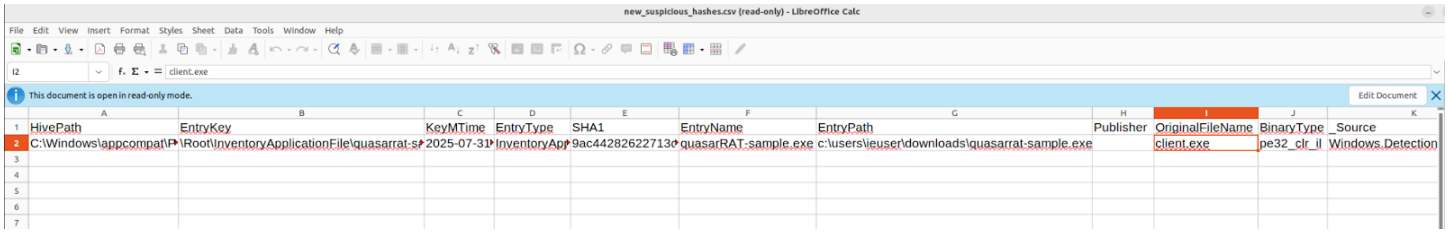
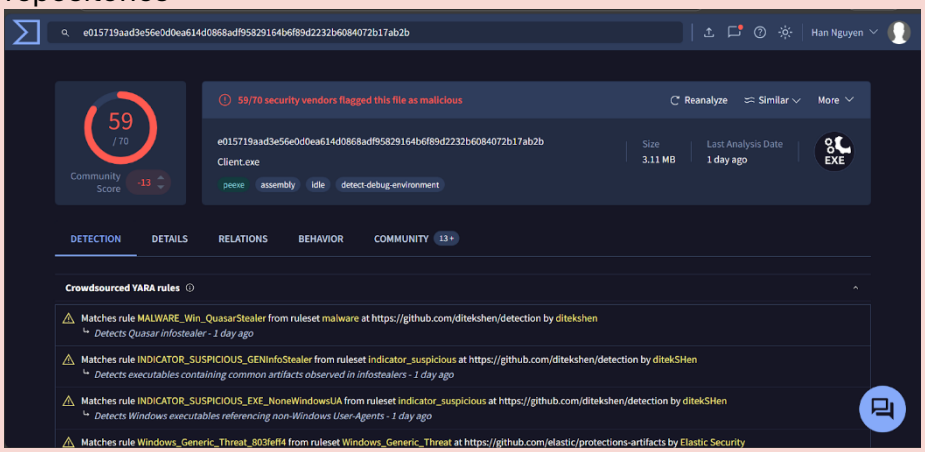


*Figure 5.4. **dff_util_execProof.py** returns new_suspicious_hashes.csv, saving time combing through all entries*

Table 2: Amcache Analysis Summary and Indicators

Attribute	Details
File Location	User download directory, common vector for malware delivery
SHA1 Hash	Matches known QuasarRAT payloads found in public malware repositories  => This event should be treated as an initial infection stage for the QuasarRAT implant.
Original Name	client.exe is consistent with default naming in QuasarRAT compiled stubs
Publisher Metadata	Missing which indicates unsigned and potentially malicious software
Binary Format	32-bit PE using .NET CLR, matches QuasarRAT's implementation language
Amcache Entry	Confirms local execution or installation on the system

3.1.3. WINDOWS. ANALYSIS. EVIDENCEOFEXECUTION (PHASE 2 – PRODUCTION ENVIRONMENT)

This phase involved a more advanced lab setup to test the full extent of the malware's capabilities (View **Section 2.6.1.b** for extra setup requirement), including its ability to download additional payloads or communicate with a C2 server over the live internet. This was a parallel setup to the first, designed for deeper visibility. Time to run the mal-sample.exe on Windows VM client and begin the real threat hunt:

Restart new capture for both Wireshark, Inetsim and Velociraptor:

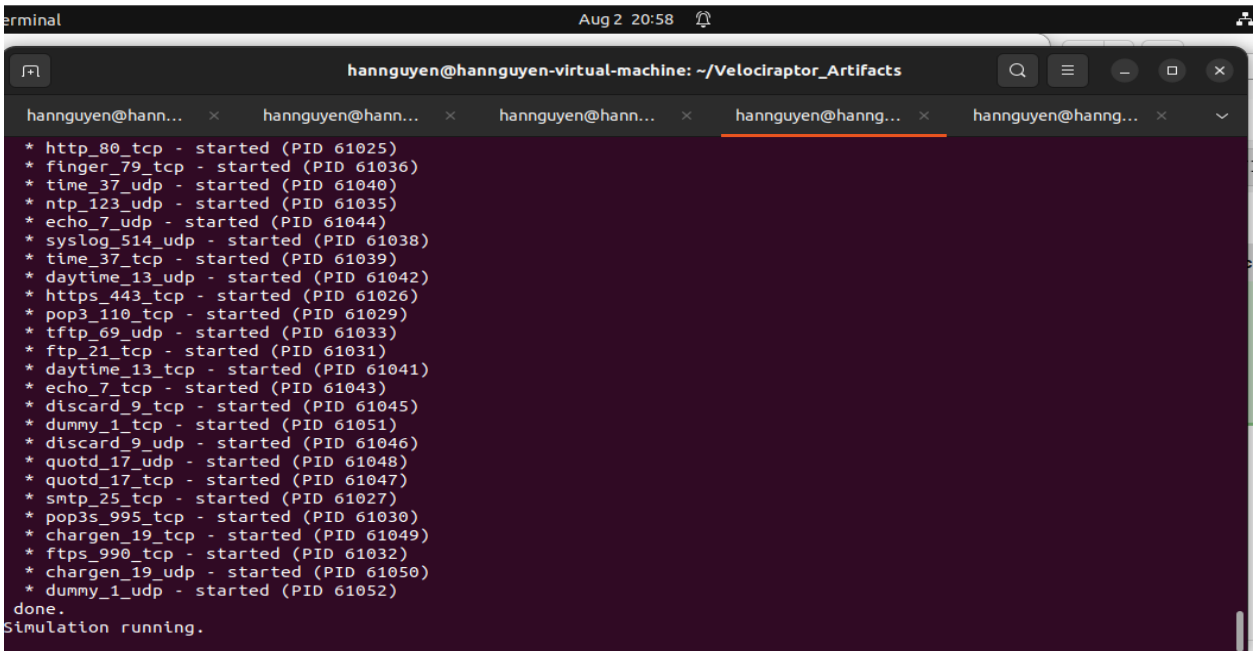


Figure 7.1. Restart InetSim before running the malware and start the hunt

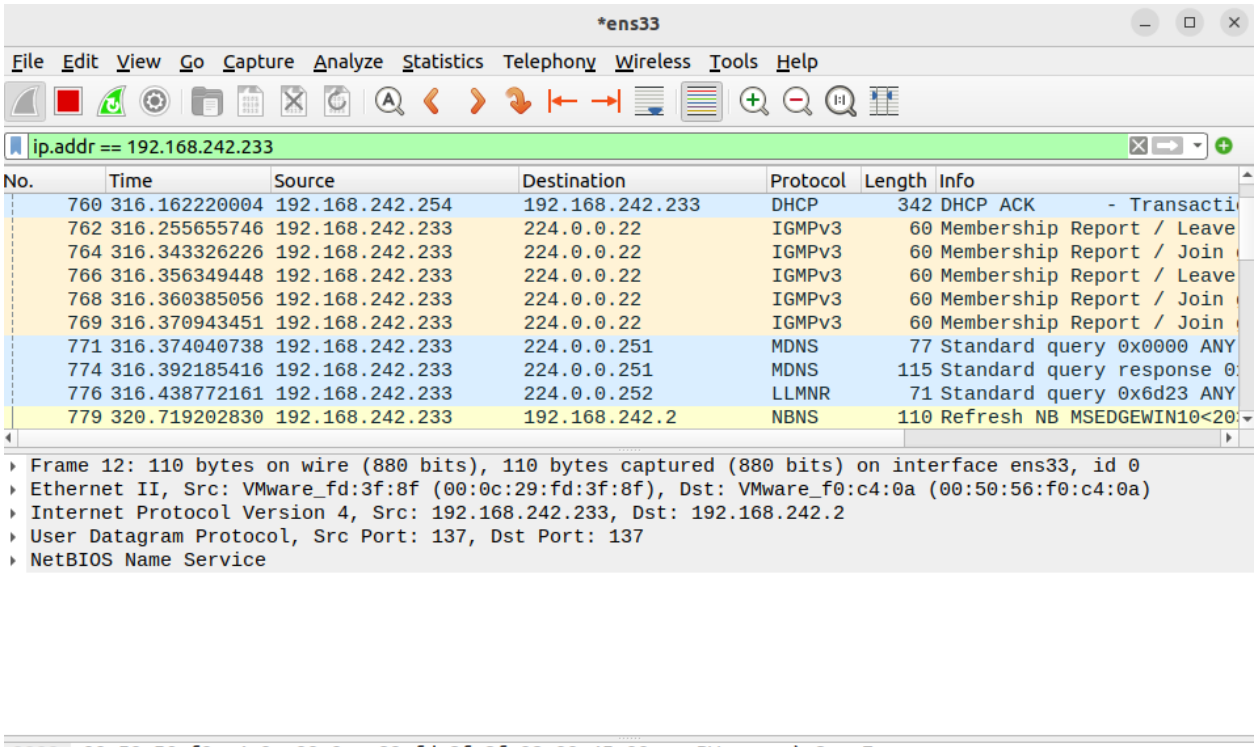


Figure 7.2. Restart Wireshark to capture Wins 10 VM and the internet traffic (ens 33, ip 192.168.242.233)

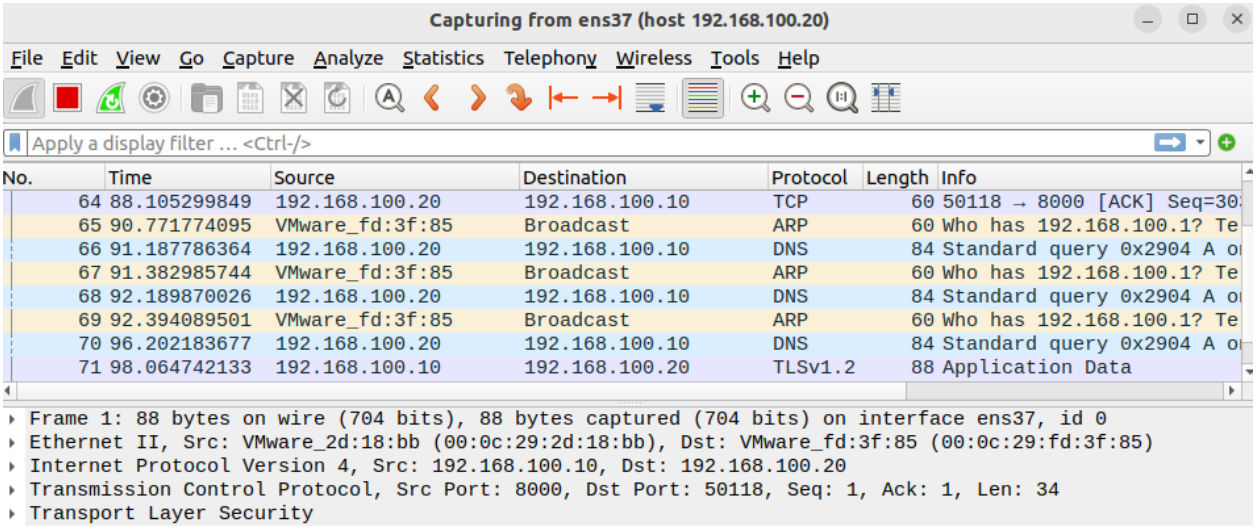


Figure 7.3. Restart & Capture Wins 10 VM and the Ubuntu Inetsim traffic (ens 37, ip 192.168.100.20)

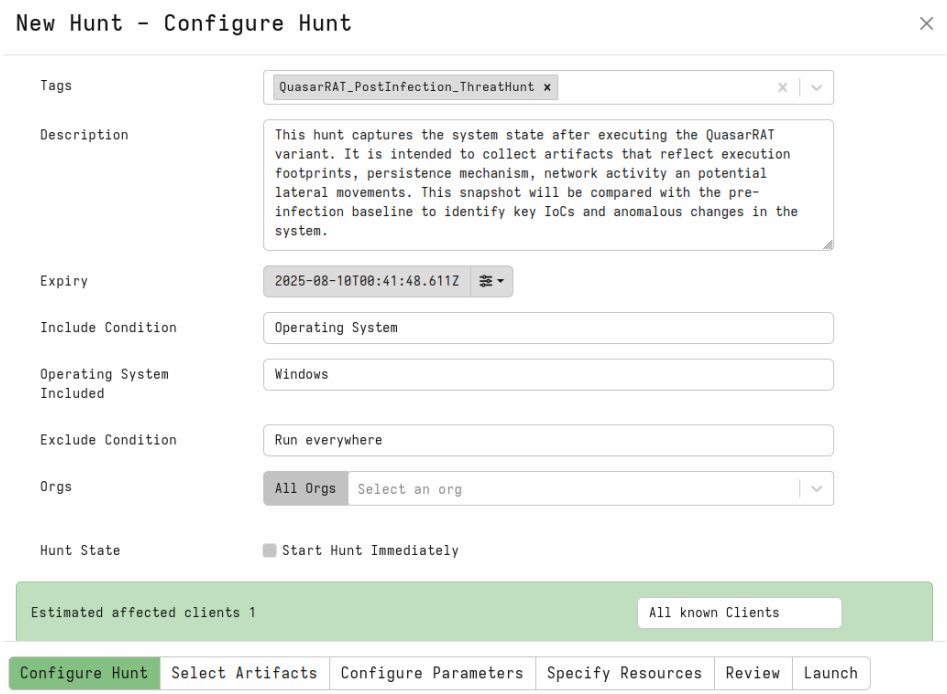


Figure 7.4. Create and launch a new threat hunt after malware infection, same artifacts as Table 1

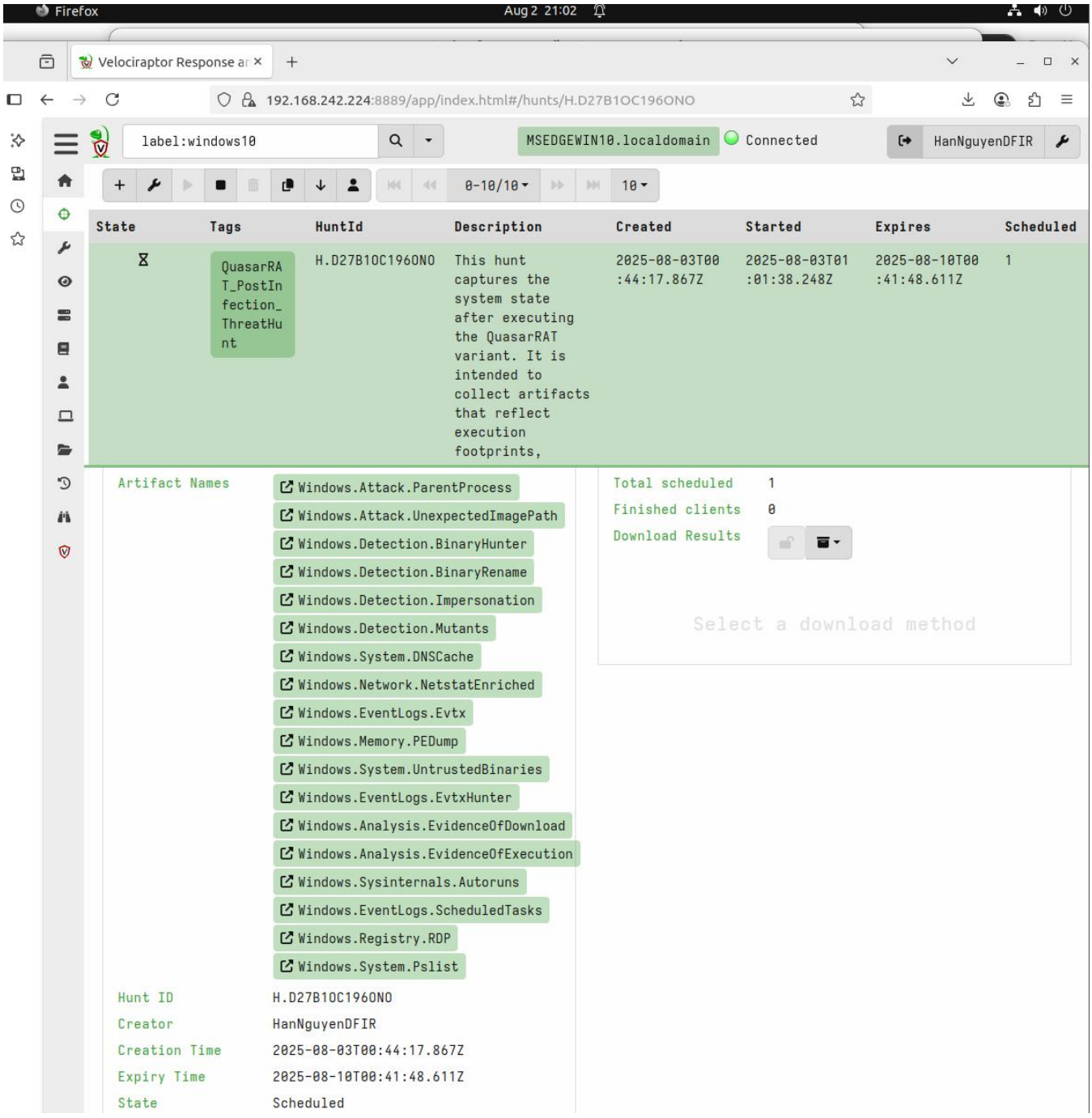


Figure 7.4 (Continue). Create and launch a new threat hunt after malware infection, same artifacts as Table 1

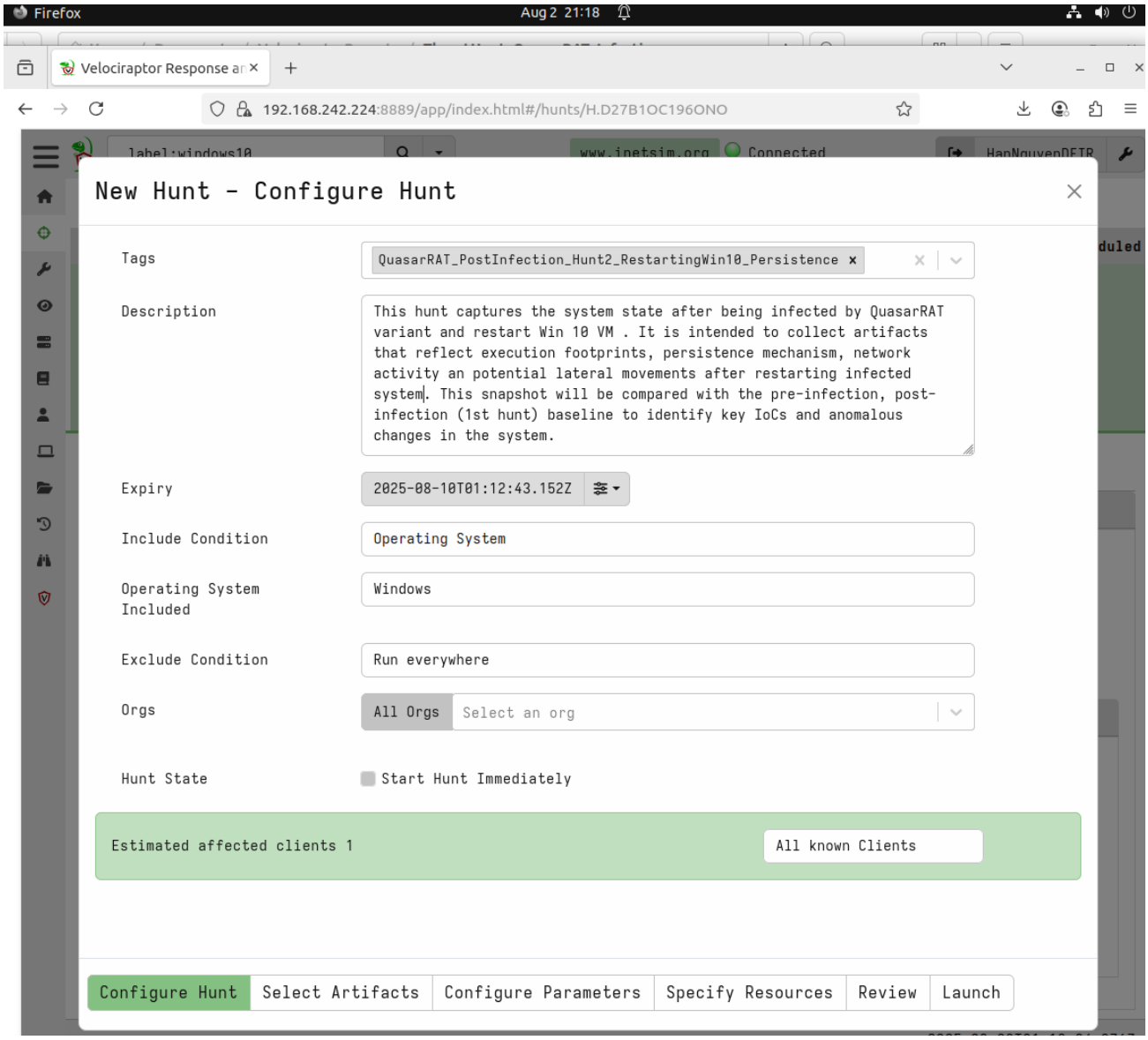


*Figure 7.5. Create and launch a post-infection but after **RESTARTING the still infected client machine** to check for **new behaviors/development/persistence**, same artifacts as Table 1*

The following reports were generated from phase 2 – this will be reused in the other post-infection analysis concerning other artifacts as well:

- **QuasarRATInfected_WireShark_Win10NInternet_Aug2_21h35.pcap**
- **QuasarRATInfected_WireShark_Win10NInetsim_Aug2_21h35.pcap**

```
Simulation stopped.
 Report written to '/var/log/inetsim/report/report.61022.txt' (330 lines)
=== INetSim main process stopped (PID 61022) ===
.
hannguyen@hannguyen-virtual-machine:~/Velociraptor_Artifacts$
```

Figure 7.6. INetSim output renamed for easier tracking (default name was report.61022.txt)

- **QuasarRATInfected_Inetsim_Aug2_2025_21h35.csv**
- **QuasarRAT_Post_Infection_Restart_Win10_H.D27B1OC196ONO_Aug2_21h30.zip**

Collect from **QuasarRAT_PostInfection_Hunt2_RestartingWin10_Persistence**, Hunt ID **H.D27BKJETCMN8K**, the evidence of execution / Amcache report showed 2 suspicious entries:

All Windows.Analysis.EvidenceOfExecution%2FAmcache.csv - LibreOffice Calc

File Edit View Insert Format Styles Sheet Data Tools Window Help

Liberation Sans | 10 pt | A158 | C:\Windows\appcompat\Programs\Amcache.hve

	E	F	G	H	I	J	
1	SHA1	EntryName	EntryPath	Publisher	OriginalFileName	BinaryType	_Sour
155	7c4e15be70d2137e8f48d	iexplore.exe	c:\program files (x86)\internet explorer\iexplore.exe	microsoft corpo	bintext.exe	pe32_i386	Windo
156	308114143e274f5c95985	iexplore.exe	c:\program files\internet explorer\iexplore.exe	microsoft corpo	iexplore.exe	pe64_amd64	Windo
157	0c9982e173a709a477327	ig.exe	c:\program files\malwarebytes\anti-malware\ig.exe	malwarebytes	ig.exe	pe32_i386	Windo
158	9ac44282622713dc53cee	Jackes.exe	c:\users\ieuser\appdata\roaming\wave google\jackes.exe		client.exe	pe32_clr_il	Windo
159	8318f1495cbdae1523f4bc	LicensingUI.exe	c:\windows\system32\licensingui.exe	microsoft corpo	bintext.exe	pe64_amd64	Windo
160	e1ab36e5c3c1453c592e2	LogonUI.exe	c:\windows\system32\logonui.exe	microsoft corporation		pe64_amd64	Windo
161	bae500ee5604634f6ce51	maintenanceservice.exe	c:\program files\mozilla firefox\maintenanceservice.exe	mozilla foundati	maintenanceservice.exe	pe64_amd64	Windo
162	e469490bc953c7905b34e	maintenanceservice_installer.exe	c:\program files\mozilla firefox\maintenanceservice_installer.exe	mozilla corporat	maintenanceservice_install	pe32_i386	Windo
163	bae500ee5604634f6ce51	maintenanceservice.exe	c:\program files (x86)\mozilla maintenance service\maintenanceservice	mozilla foundati	maintenanceservice.exe	pe64_amd64	Windo
164	9ac44282622713dc53cee	mal-sample.exe	c:\users\ieuser\downloads\mal-sample.exe		client.exe	pe32_clr_il	Windo
165	3dfd12bb7cafacfd5385c3	Malwarebytes.exe	c:\program files\malwarebytes\anti-malware\malwarebytes.exe	malwarebytes	malwarebytes.dll	pe64_amd64	Windo
166	a70fec4b3c8870fcca765c	malwarebytes_assistant.exe	c:\program files\malwarebytes\anti-malware\malwarebytes_assistant.ex	malwarebytes	malwarebytes_assistant.dll	pe64_amd64	Windo
167	d3a136aab0d951f65b579	mb5uns.exe	c:\program files\malwarebytes\anti-malware\mb5uns.exe	malwarebytes	bintext.exe	pe32_i386	Windo

Figure 7.7. All Windows.Analysis.EvidenceOfExecution%2FAmcache.csv

I have improved on Evidence of Execution / Amcache differentiation tool with Virus Total API checks, for more information on the source code – please refer to **Appendix – Section 1.2**. The resulting report includes 2 files - **suspicious_amcache_diff.csv** and **suspicious_amcache_virustotal_report.csv**

suspicious_amcache_diff.csv (read-only) - LibreOffice Calc

File Edit View Insert Format Styles Sheet Data Tools Window Help

B16

This document is open in read-only mode. Edit Document

	E	F	G	H	I	J	K
1	SHA1	EntryName	EntryPath	Publisher	OriginalFileName	BinaryType	_Source
2	9ac44282622713dc53ceeb974177a0e900fae972	Jackes.exe	c:\users\ieuser\appdata\roaming\wave google\jackes.exe		client.exe	pe32_clr_il	Windows.Detection.Amcache
3	9ac44282622713dc53ceeb974177a0e900fae972	mal-sample.exe	c:\users\ieuser\downloads\mal-sample.exe		client.exe	pe32_clr_il	Windows.Detection.Amcache
4							
5							

Figure 7.8. suspicious_amcache_diff.csv

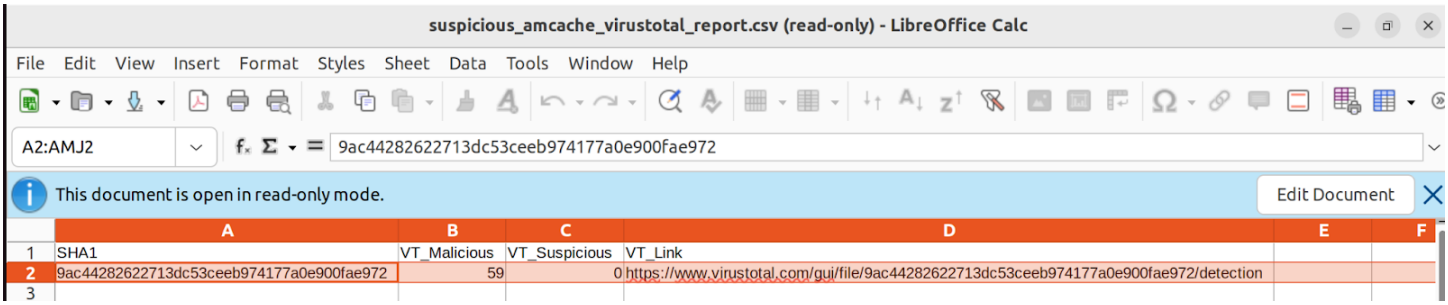


Figure 7.9. suspicious_amcache_virustotal_report.csv

These two entries from the Amcache artifact are suspicious for the following reasons:

a. Identical Hash for Different Files

SHA1 Hash: **9ac44282622713dc53ceeb974177a0e900fae972** appears in both entries.

However, the filenames and paths differ:

Jackes.exe located in:

c:\users\ieuser\appdata\roaming\wave google\jackes.exe

mal-sample.exe located in:

c:\users\ieuser\downloads\mal-sample.exe

A hash collision for different filenames and locations typically indicates a deliberately distributed or renamed binary - a common trait in malware propagation and evasion tactics.

b. Suspicious File Paths

Both files reside in user-accessible and commonly abused locations:

AppData\Roaming and **Downloads** folders are often used by malware to evade detection and persist across reboots.

These directories are writable by standard users and not protected by default Windows security settings.

c. Generic or Misleading Metadata

OriginalFileName: client.exe is overly generic (Figure 7.8 – Column I) and does not reflect the actual filename (common in malware to mislead investigators).

Publisher: Blank (Figure 7.8 – Column H). Legitimate applications usually have valid digital signatures and publisher metadata.

BinaryType: pe32_clr_il (Figure 7.8 – Column J) suggests it's a .NET executable, commonly used in malware due to ease of obfuscation and reflection-based loading.

d. Timing and Duplication

Both entries were recorded within one second of each other:

2025-08-03T01:09:05Z and **2025-08-03T01:09:06Z**, suggesting they may have been dropped simultaneously during infection.

e. Virus Total hash check / Cross-verification:

Use SHA-1 value in the suspicious_amcache_virustotal_report.csv (Figure 7.9 – Column A) to cross check with Virus Total:

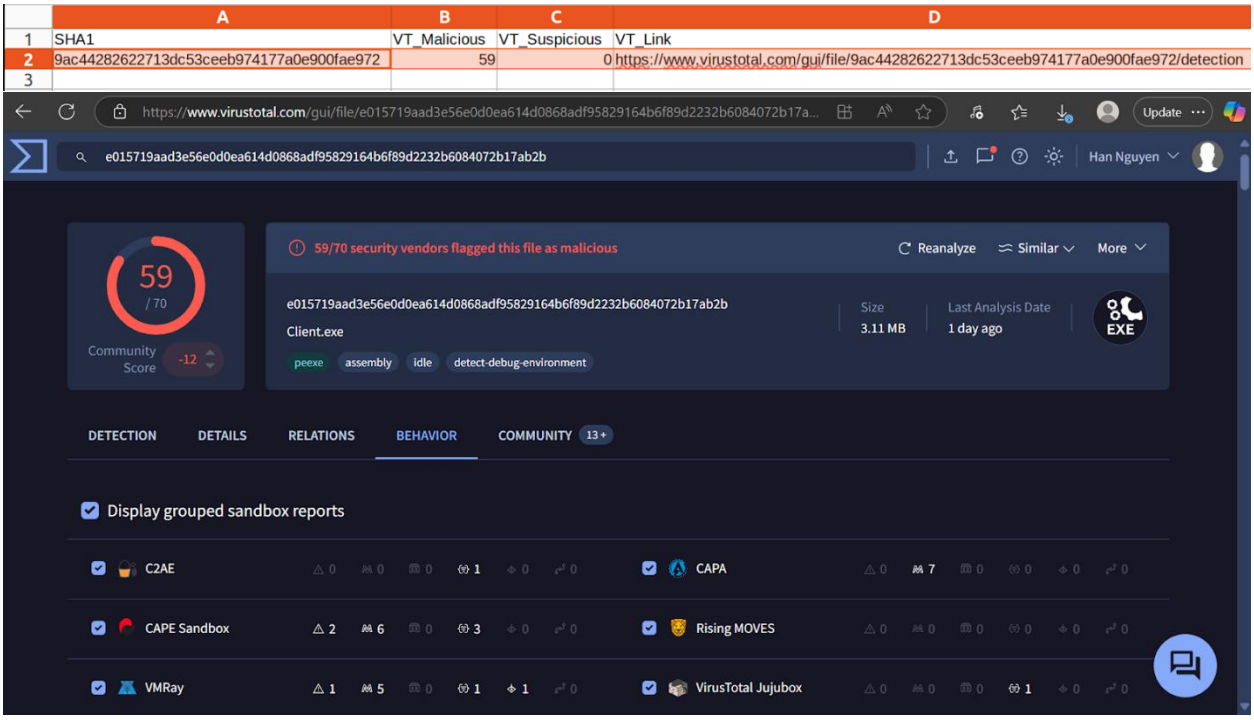


Figure 7.10. Cross-Check hash with Virus Total

3.2. WINDOWS.DETECTION.BINARYHUNTER

Source

```
-- setup hash lists if needed
LET MD5Array ≤ split(sep='\\s+',string=MD5List)
LET SHA1Array ≤  split(sep='\\s+',string=SHA1List)
LET SHA256Array ≤ split(sep='\\s+',string=SHA256List)

-- firstly find files in scope with performance
LET find_files = SELECT *,
      read_file(filename=OSPath,accessor=Accessor,offset=0,length=2) as _Header
  FROM if(condition=DateBefore AND DateAfter,
      then={
          SELECT OSPath, Name, Size,Mtime,Atime,Ctime,Btime
          FROM glob(globs=TargetGlob,accessor=Accessor)
          WHERE NOT IsDir AND NOT IsLink
              AND Size > SizeMin AND Size < SizeMax
              AND ( Mtime < DateBefore OR Ctime < DateBefore OR Btime < DateBefore )
              AND ( Mtime > DateAfter OR Ctime > DateAfter OR Btime > DateAfter )
      },
      else={ SELECT * FROM  if(condition=DateBefore,
          then={
              SELECT OSPath, Name, Size,Mtime,Atime,Ctime,Btime
              FROM glob(globs=OSPath,accessor=Accessor)
              WHERE NOT IsDir AND NOT IsLink
                  AND Size > SizeMin AND Size < SizeMax
                  AND ( Mtime < DateBefore OR Ctime < DateBefore OR Btime < DateBefore )
          },
          else={ SELECT * FROM  if(condition=DateAfter,
          then={
              SELECT OSPath, Name, Size,Mtime,Atime,Ctime,Btime
              FROM glob(globs=TargetGlob,accessor=Accessor)
              WHERE NOT IsDir AND NOT IsLink
                  AND Size > SizeMin AND Size < SizeMax
                  AND ( Mtime > DateAfter OR Ctime > DateAfter OR Btime > DateAfter )
          },
          else={
              SELECT OSPath, Name, Size,Mtime,Atime,Ctime,Btime
              FROM glob(globs=TargetGlob,accessor=Accessor)
              WHERE NOT IsDir AND NOT IsLink
                  AND Size > SizeMin AND Size < SizeMax
          })})})
  WHERE _Header = 'MZ'
```

Figure 8.1. Windows.Detection.BinaryHunter Velociraptor Query Language (VQL)

As part of our proactive threat hunting with Velociraptor, we identified two suspicious executables flagged under the ***Windows.Detection.BinaryHunter*** artifact. Both samples appear identical in hash and structure, indicating a reused binary present in two different user directories. These binaries raise multiple red flags and require further containment and review, they are:

(1) JACKES.EXE:

```
A
File
{"OSPath":"C:\\Users\\IEUser\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\9nl4xu0d.default-release\\gmp-gmpopenh264\\2.6.0\\gmpopenh264.dll","Name":"gmpopenh264.dll","Size":1168960,"Timest
{"OSPath":"C:\\Users\\IEUser\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\9nl4xu0d.default-release\\gmp-widevinecdm\\4.10.2891.0\\widevinecdm.dll","Name":"widevinecdm.dll","Size":19439200,"Ti
{"OSPath":"C:\\Users\\IEUser\\AppData\\Roaming\\Wave Google\\Jackes.exe","Name":"Jackes.exe","Size":3266048,"Timestamps":{"Mtime":"2025-08-01T00:19:12Z","Atime":"2025-08-
03T00:57:56.86844Z","Ctime":"2025-08-01T00:19:12Z","Btime":"2025-08-03T00:57:52.3389944Z"}}
{"OSPath":"C:\\Users\\IEUser\\AppData\\Local\\Microsoft\\OneDrive\\OneDrive.exe","Name":"OneDrive.exe","Size":4969848,"Timestamps":{"Mtime":"2025-08-02T22:16:25.3066017Z","Atime":"2025-08

B
Authenticode
{"Filename":"C:\\Users\\IEUser\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\9nl4xu0d.default-release\\gmp-gmpopenh264\\2.6.0\\gmpopenh264.dll","ProgramName":"","PublisherLink":null,"MoreInfoLi
{"Filename":"C:\\Users\\IEUser\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\9nl4xu0d.default-release\\gmp-widevinecdm\\4.10.2891.0\\widevinecdm.dll","ProgramName":"","PublisherLink":null,"MoreI
{"Filename":"C:\\Users\\IEUser\\AppData\\Roaming\\Wave Google\\
Jackes.exe","ProgramName":null,"PublisherLink":null,"MoreInfoLink":null,"SerialNumber":null,"IssuerName":null,"SubjectName":null,"Timestamp":null,"Trusted":"untrusted","_ExtraInfo":{"Catalog":""}}
{"Filename":"C:\\Users\\IEUser\\AppData\\Local\\Microsoft\\OneDrive\\OneDrive.exe","ProgramName":"Microsoft","PublisherLink":null,"MoreInfoLink":"http://www.microsoft.com","SerialNumber":"3300
{"Filename":"C:\\Users\\IEUser\\AppData\\Local\\Microsoft\\OneDrive\\OneDriveStandaloneUpdater.exe","ProgramName":"Microsoft","PublisherLink":null,"MoreInfoLink":"http://www.microsoft.com","Se
{"Filename":"C:\\Users\\IEUser\\AppData\\Local\\Microsoft\\OneDrive\\25.130.0706.0004\\CheckboxWindows.dll","ProgramName":"Microsoft","PublisherLink":null,"MoreInfoLink":"http://www.microsoft.c

C
PE
thm":"SHA256-RSA","Subject":"C=US, O=DigiCert, Inc., CN=DigiCert Trusted G4 RSA4096 SHA256 TimeStamping CA","Issuer":"C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert Trusted
cd:b1\n Exponent: 65537 (0x10001)\n","Extensions":{"BasicConstraints":{"Critical":true,"IsCA":true,"MaxPathLen":0},"SubjectKeyId":{"Critical":false,"Value":"BA:16:D9:6D:4D:8
{"FileHeader":{"Machine":"IMAGE_FILE_MACHINE_I386","TimeDateStamp":"2023-03-
12T16:16:39Z","TimeDateStampRaw":1678637799,"Characteristics":258,"ImageBase":4194304},"GUIDAge":"","PDB":"","Directories":{"Base_Relocation_Directory":{"Timestamp":"1970-01-
01T00:00:00Z","TimestampRaw":0,"Size":12,"FileAddress":3284992,"SectionName":""},"DotNet_Directory":{"Timestamp":"1979-01-
16T14:30:43Z","TimestampRaw":285345043,"Size":72,"FileAddress":8200,"SectionName":".text"},"IAT_Directory":{"Timestamp":"1991-06-
26T21:16:01Z","TimestampRaw":677970961,"Size":8,"FileAddress":8192,"SectionName":".text"},"Import_Directory":{"Timestamp":"1970-01-
01T00:00:00Z","TimestampRaw":0,"Size":83,"FileAddress":3269544,"SectionName":""},"Resource_Directory":{"Timestamp":"1970-01-
01T00:00:00Z","TimestampRaw":0,"Size":2707,"FileAddress":3276800,"SectionName":""}},"Sections":[{"Perm":"xr-","Name":".text","FileOffset":512,"VMA":4202496,"RVA":8192,"Size":3261952},
{"Perm":"-r-","Name":".rsrc","FileOffset":3262464,"VMA":7471104,"RVA":3276800,"Size":3072},{"Perm":"-
r-","Name":".reloc","FileOffset":3265536,"VMA":7479296,"RVA":3284992,"Size":512}],"Resources":[{"Type":"RT_VERSION","TypeId":16,"FileOffset":3262624,"DataSize":796,"CodePage":0},
{"Type":"RT_MANIFEST","TypeId":24,"FileOffset":3263420,"DataSize":1751,"CodePage":0}],"VersionInformation":{"Comments":"","CompanyName":"","FileDescription":"Quasar
Client","FileVersion":"1.4.1","InternalName":"Client.exe","LegalCopyright":"Copyright © MaxXor
2023","LegalTrademarks":"","OriginalFilename":"Client.exe","ProductName":"Quasar","ProductVersion":"1.4.1","Assembly Version":"1.4.1.0"},"Imports":["mscoree.dll!_CorExeMain"],"Exports":
[],"Forwards":[],"ImpHash":"f34d5f2d4577ed6d9ceec516c1f5a744","Authenticode":null,"AuthenticodeHash":
{"MD5":"14f2da46d07a62ea9bdb3a87ba1c33f5","SHA1":"b142c01b7c6f17de30804493d085420a38c7b918","SHA256":"d5357bfb61fd4154dc47b75dff41b854cc0a67355bab40b0f99fc22d54b4436a","Has
hMatches":false}}
```

*Figure 8.2. Suspicious Jackes.exe file entry in Windows.Detection.BinaryHunter (stemming from mal-sample.exe – and Jackes.exe was **not known** to us)*

(2) MAL-SAMPLE.EXE:

```
B
Authenticode
{"Filename":"C:\\Users\\IEUser\\Downloads\\7z2409-x64.exe","ProgramName":null,"PublisherLink":null,"MoreInfoLink":null,"SerialNumber":null,"IssuerName":null,"SubjectName":null,"Timestamp":null,"
{"Filename":"C:\\Users\\IEUser\\Downloads\\HanDFIR-Velociraptor.exe","ProgramName":"","PublisherLink":null,"MoreInfoLink":"","SerialNumber":"bab856909c15ab30f2285aed94c8c37","IssuerName":"
{"Filename":"C:\\Users\\IEUser\\Downloads\\MyDFIR-Velociraptor(1).exe","ProgramName":"","PublisherLink":null,"MoreInfoLink":"","SerialNumber":"bab856909c15ab30f2285aed94c8c37","IssuerName"
{"Filename":"C:\\Users\\IEUser\\Downloads\\MyDFIR-Velociraptor.exe","ProgramName":"","PublisherLink":null,"MoreInfoLink":"","SerialNumber":"bab856909c15ab30f2285aed94c8c37","IssuerName":"C
{"Filename":"C:\\Users\\IEUser\\Downloads\\Sample-10-4.exe","ProgramName":null,"PublisherLink":null,"MoreInfoLink":null,"SerialNumber":null,"IssuerName":null,"SubjectName":null,"Timestamp":null,
{"Filename":"C:\\Users\\IEUser\\Downloads\\autoruns.exe","ProgramName":"Autoruns","PublisherLink":null,"MoreInfoLink":"https://www.sysinternals.com","SerialNumber":"33000003af30400e4ca34d05
{"Filename":"C:\\Users\\IEUser\\Downloads\\mal-
sample.exe","ProgramName":null,"PublisherLink":null,"MoreInfoLink":null,"SerialNumber":null,"IssuerName":null,"SubjectName":null,"Timestamp":null,"Trusted":"untrusted","_ExtraInfo":{"Catalog":""}}
{"Filename":"C:\\Users\\IEUser\\Downloads\\ProcessExplorer\\procexp64.exe","ProgramName":"ProcExp","PublisherLink":null,"MoreInfoLink":"https://www.sysinternals.com","SerialNumber":"3300000

{"FileHeader":{"Machine":"IMAGE_FILE_MACHINE_I386","TimeDateStamp":"2023-03-
12T16:16:39Z","TimeDateStampRaw":1678637799,"Characteristics":258,"ImageBase":4194304},"GUIDAge":"","PDB":"","Directories":{"Base_Relocation_Directory":{"Timestamp":"1970-01-
01T00:00:00Z","TimestampRaw":0,"Size":12,"FileAddress":3284992,"SectionName":""},"DotNet_Directory":{"Timestamp":"1979-01-
16T14:30:43Z","TimestampRaw":285345043,"Size":72,"FileAddress":8200,"SectionName":".text"},"IAT_Directory":{"Timestamp":"1991-06-
26T21:16:01Z","TimestampRaw":677970961,"Size":8,"FileAddress":8192,"SectionName":".text"},"Import_Directory":{"Timestamp":"1970-01-
01T00:00:00Z","TimestampRaw":0,"Size":83,"FileAddress":3269544,"SectionName":""},"Resource_Directory":{"Timestamp":"1970-01-
01T00:00:00Z","TimestampRaw":0,"Size":2707,"FileAddress":3276800,"SectionName":""}},"Sections":[{"Perm":"xr-","Name":".text","FileOffset":512,"VMA":4202496,"RVA":8192,"Size":3261952},
{"Perm":"-r-","Name":".rsrc","FileOffset":3262464,"VMA":7471104,"RVA":3276800,"Size":3072},{"Perm":"-
r-","Name":".reloc","FileOffset":3265536,"VMA":7479296,"RVA":3284992,"Size":512}],"Resources":[{"Type":"RT_VERSION","TypeId":16,"FileOffset":3262624,"DataSize":796,"CodePage":0},
{"Type":"RT_MANIFEST","TypeId":24,"FileOffset":3263420,"DataSize":1751,"CodePage":0}],"VersionInformation":{"Comments":"","CompanyName":"","FileDescription":"Quasar
Client","FileVersion":"1.4.1","InternalName":"Client.exe","LegalCopyright":"Copyright © MaxXor
2023","LegalTrademarks":"","OriginalFilename":"Client.exe","ProductName":"Quasar","ProductVersion":"1.4.1","Assembly Version":"1.4.1.0"},"Imports":["mscoree.dll!_CorExeMain"],"Exports":
[],"Forwards":[],"ImpHash":"f34d5f2d4577ed6d9ceec516c1f5a744","Authenticode":null,"AuthenticodeHash":
{"MD5":"14f2da46d07a62ea9bdb3a87ba1c33f5","SHA1":"b142c01b7c6f17de30804493d085420a38c7b918","SHA256":"d5357bfb61fd4154dc47b75dff41b854cc0a67355bab40b0f99fc22d54b4436a","Has
hMatches":false}}
```

Figure 8.3. Suspicious Mal-sample.exe (our MalwareBazaar sample) file tracked in Windows.Detection.BinaryHunter

Table 3: Suspicious Binary Reuse of these 2 files

Filename	Path	File Size	File Version	Product Name
mal-sample.exe	C:\Users\IEUser\Downloads\mal-sample.exe	3,266,048 B	1.4.1	Quasar
Jackes.exe	C:\Users\IEUser\AppData\Roaming\Wave Google\Jackes.exe	3,266,048 B	1.4.1	Quasar

Both files share the same hash (SHA-256: e015719aad3e56...) and are internally identified as QuasarRAT, a well-known remote access trojan often used for unauthorized surveillance, keylogging, and command execution.

(3) WHY THESE FILES ARE SUSPICIOUS

i. **Unusual Location:** The binaries are located in user-accessible folders commonly abused by malware for persistence (**Downloads and AppData\Roaming**).

ii. **Lack of Digital Signature:** Neither binary is signed or trusted. Windows marks them as **"untrusted."**

Table 4: Authenticode Summary

Field	Value
Trusted	Untrusted
Publisher	Not available
Timestamp	Not available
Certificate Serial Number	Not available

iii. **Suspicious Version Info:** The embedded metadata claims legitimate product names (Quasar, Client.exe), but these are easily spoofed.

iv. **Duplicate Hash:** Both files have identical hashes and sizes but different filenames and paths, indicating possible lateral movement or repeated execution attempts.

v. **Known RAT Variant:** The embedded strings and structure match QuasarRAT clients, including .NET executable format and import of mscoree.dll!_CorExeMain.

Table 5: VirusTotal Results (Hash: e015719aad3e56...)

Score	Malicious Vendors	Community Score	Verdict
59/71	TrendMicro, Kaspersky, BitDefender, Microsoft, etc.	87/100	Likely Malicious

The high detection rate and elevated community score (87) confirm this is an actively flagged malicious sample across multiple vendors. These executables are most likely unauthorized RAT clients, possibly installed via phishing or user error. While no active C2 connections were observed during this review, the presence of known malware is enough to consider the system compromised.

At this point, recommended actions would be to isolate the affected machine from the network. Perform a full memory and disk forensic acquisition. Search for persistence mechanisms (scheduled tasks, registry entries, startup folders). Investigate the initial access vector (email logs, user behavior, external drive usage). Check other endpoints for matching hashes using EDR/SIEM.

3.3. WINDOWS.EVENTLOGS.EVTX

The **Windows.EventLogs.Evtx** artifact was critical for establishing the exact time of execution. By filtering for the **Microsoft-Windows-Shell-Core/Operational** log channel, we located events associated with the malicious file.

(1) SOURCE ARTIFACT

a. Velociraptor Artifact: **Windows.EventLogs.Evtx**
b. Log Channel: **Microsoft-Windows-Shell-Core/Operational**
c. Correlated Hunt: **Windows.Network.NetstatEnriched**
d. Here's how each VQL query in the source code works:

`SELECT * FROM Artifact.Windows.Registry.UserAssist()` collects data from the UserAssist registry key, which logs information about programs a user has recently executed. It provides details like the program's path, its execution count, and the last time it was run.

`SELECT * FROM Artifact.Windows.Detection.Amcache()` retrieves data from the Amcache registry hive, which is a key component for application compatibility and tracks a history of executed programs. This is a very valuable source for forensic analysis as it contains file paths and hashes.

`SELECT * FROM Artifact.Windows.Forensics.Timeline()`: This is a powerful query that generates a timeline of file system events from various sources (like MFT, USN Journal, and Event Logs). It helps investigators see the sequence of events on a system.

`SELECT * FROM Artifact.Windows.Registry.AppCompatCache()`: This query, also known as ShimCache, retrieves data from the Application Compatibility Cache. It records metadata about executed programs, including file paths and sizes, making it another crucial source for detecting evidence of execution.

`SELECT * FROM Artifact.Windows.Forensics.Prefetch()`: This query collects Prefetch files, which are created by Windows to speed up application startup. These files contain information about which files and folders an application accessed, providing a clear record of its execution.

`SELECT * FROM Artifact.Windows.Forensics.RecentApps()`: This query is used to find and collect information about recently used applications, helping to uncover evidence of user activity.

Source

```
LET VSS_MAX_AGE_DAYS ≤ VSSAnalysisAge
LET Accessor = if(condition=VSSAnalysisAge > 0, then="ntfs_vss", else="auto")

// expand provided glob into a list of paths on the file system (fs)
LET fspaths =
    SELECT OSPath FROM glob(globs=expand(path=EvtxGlob), accessor=Accessor)
    WHERE OSPath =~ PathRegex

// function returning parsed evtx from list of paths
LET evtxsearch(pathList) = SELECT * FROM foreach(
      row=pathList,
      query={
        SELECT *,
          timestamp(epoch=int(int=System.TimeCreated.SystemTime)) AS TimeCreated,
          System.Channel as Channel,
          System.EventRecordID as EventRecordID,
          System.EventID.Value as EventID,
          OSPath
        FROM parse_evtx(filename=OSPath, accessor=Accessor)
        WHERE
          if(condition=StartDate,
             then=TimeCreated ≥ timestamp(string=StartDate),
             else=true)
          AND if(condition=EndDate,
                 then=TimeCreated ≤ timestamp(string=EndDate),
                 else=true)
          AND Channel =~ ChannelRegex
          AND str(str=EventID) =~ IDRegex
      }
    )

SELECT * FROM evtxsearch(pathList=fspaths)

```

Figure 9.1. Windows.EventLogs.Evtx

(2) FINDINGS

On **2025-08-01 at 00:13:58 UTC**, the Microsoft-Windows-Shell-Core Operational log recorded the execution of a binary named **Jackes.exe** from the following location:

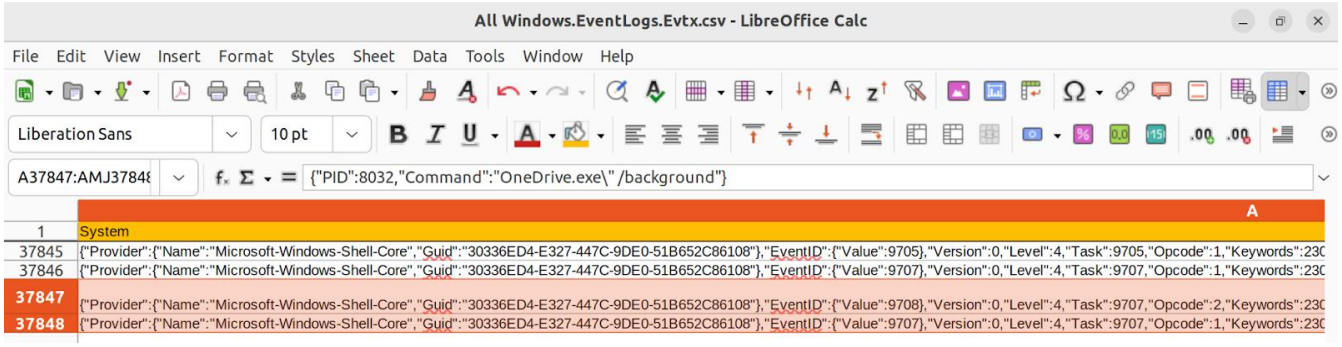


Figure 9.2. Windows.EventLogs.Evtx output, showing the two Event IDs 9707 and 9708 for Jackes.exe.

This data provides a definitive timestamp for the start of the malicious process, which is essential for correlating with network activity.

Table 4: Event Log Analysis

Timestamp (UTC)	Event ID	Description	Process ID	Command
2025-08-01 00:13:58	9707	Command execution initiated	5928	Jackes.exe
2025-08-01 00:13:59	9708	Command execution completed	8096	Jackes.exe

(3) CORRELATED INDICATORS

i. File Hashes:

- MD5: cb2e51e6572d2b9e45fc39d4ab138a11
- SHA1: 9ac44282622713dc53ceeb974177a0e900fae972
- SHA256: e015719aad3e56e0d0ea614d0868adf95829164b6f89d2232b6084072b17ab2b

ii. Signature Status: Unsigned / Untrusted

iii. Network Activity (from NetstatEnriched):

- Outbound TCP connection to **66.63.187.164:8596** immediately after execution
- Remote Host: Resolved to suspected C2 infrastructure

iv. Analysis

The binary Jackes.exe was executed from an **unusual, non-standard directory** (AppData\Roaming\Wave Google) not associated with legitimate Google products. The filename and storage path are consistent with **malware hiding within user profile folders** to evade privilege requirements and security controls.

The **Shell-Core Operational logs** confirm the process was launched interactively by the Windows shell in the context of **MSEDGEWIN10\IEUser**. This aligns with known behavior of remote access trojans (RATs) such as QuasarRAT, which execute in user space and establish outbound C2 connections shortly after launch.

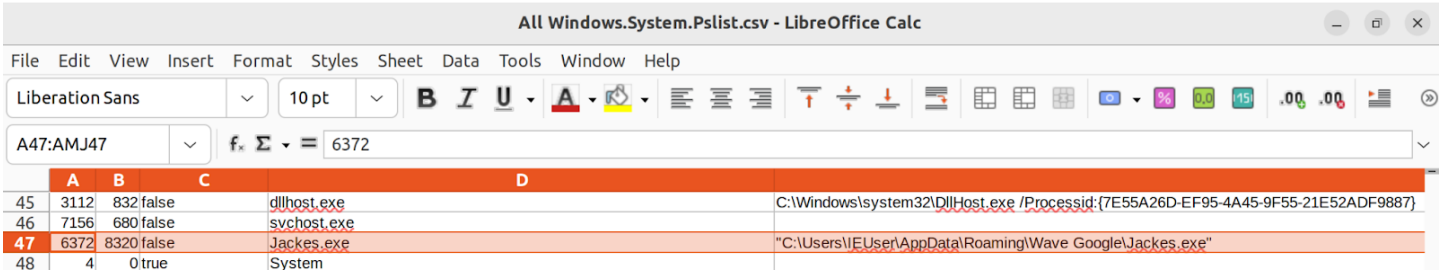
Temporal correlation between the process start time and outbound network activity strongly indicates that this execution triggered the malicious network session. The lack of a valid digital signature, coupled with confirmed C2 communication, categorizes this binary as **malicious**.

v. Conclusion

The collected evidence demonstrates that Jackes.exe was **actively executed** on the system and engaged in network communication with an external host known to be linked with malicious infrastructure. This satisfies two key elements of the intrusion kill chain: **Execution** and **Command & Control**.

3.4. WINDOWS.SYSTEM.PSLIST

This artifact confirmed that the **Jackes.exe** process was running on the system during the period of observation, correlating with the execution timestamp from the Event Logs.



	A	B	C	D	
45	3112	832	false	dllhost.exe	C:\Windows\system32\DllHost.exe /Processid:{7E55A26D-EF95-4A45-9F55-21E52ADF9887}
46	7156	680	false	svchost.exe	
47	6372	8320	false	Jackes.exe	"C:\Users\IEUser\AppData\Roaming\Wave Google\Jackes.exe"
48	4	0	true	System	

*Figure 10: Confirm unknown program **Jackes.exe** running during infection period*

3.5. WINDOWS.NETWORK.NETSTATENRICHED

(1) SOURCE ARTIFACT

The NetstatEnriched VQL queries provides a comprehensive table that shows:

a. The **Protocol** being used (e.g., TCP, UDP).
b. The **Local Address** and **Port**.
c. The **Remote Address** and **Port** the connection is going to.
d. The connection **State** (e.g., ESTABLISHED, LISTEN).
e. The **Process ID (PID)** and the **Process Name** (exe name) associated with that connection.
f. The **Command Line** used to start the process.
g. The **Time** the connection was established.

Hence, this is a critical tool for a threat hunt because it helps an investigator quickly see which programs are communicating with what remote servers, allowing them to spot suspicious C2 (command and control) traffic or other unauthorized network activity.

Source Netstat

```
LET VerifiedRegex <= SELECT Regex
      FROM parse_csv(filename=AuthenticodeVerifiedMap, accessor="data")
      WHERE Choice=AuthenticodeVerified LIMIT 1
LET StatusRegex <= SELECT Regex
      FROM parse_csv(filename=StatusMap, accessor="data")
      WHERE Choice=Status LIMIT 1
LET FamilyRegex <= SELECT Regex
      FROM parse_csv(filename=FamilyMap, accessor="data")
      WHERE Choice=Family LIMIT 1
LET TypeRegex <= SELECT Regex
      FROM parse_csv(filename=TypeMap, accessor="data")
      WHERE Choice=Type LIMIT 1

LET process <= SELECT Pid as PsId,
      Ppid,
      Name,
      CommandLine,
      Exe,
      Hash,
      Authenticode,
      Username
  FROM Artifact.Windows.System.Pslist(
      DISABLE_DANGEROUS_API_CALLS=DISABLE_DANGEROUS_API_CALLS)
  WHERE Name =~ ProcessNameRegex
      AND Exe =~ ProcessPathRegex
      AND CommandLine =~ CommandLineRegex

LET results = SELECT Pid,
          { SELECT Ppid FROM process WHERE PsId = Pid } as Ppid,
          { SELECT Name FROM process WHERE PsId = Pid } as Name,
          { SELECT Exe FROM process WHERE PsId = Pid } as Path,
          { SELECT CommandLine FROM process WHERE PsId = Pid } as CommandLine,
          { SELECT Hash FROM process WHERE PsId = Pid } as Hash,
          { SELECT Username FROM process WHERE PsId = Pid } as Username,
          { SELECT Authenticode FROM process WHERE PsId = Pid } as Authenticode,
          FamilyString as Family,
          TypeString as Type,
          Status,
          Laddr.IP as Laddr,
          Laddr.Port as Lport,
          Raddr.IP as Raddr,
          Raddr.Port as Rport,
          Timestamp
      FROM netstat()
      WHERE
          Name =~ ProcessNameRegex
          AND Path =~ ProcessPathRegex
          and CommandLine =~ CommandLineRegex
          and Username =~ UsernameRegex
          and ( Hash.MD5 =~ HashRegex
            or Hash.SHA1 =~ HashRegex
            or Hash.SHA256 =~ HashRegex
            or not Hash )
          and ( Authenticode.IssuerName =~ AuthenticodeIssuerRegex or not Authenticode )
          and ( Authenticode.SubjectName =~ AuthenticodeSubjectRegex or not Authenticode )
          and ( Authenticode.Trusted =~ VerifiedRegex.Regex[0] or not Authenticode )
          and Status =~ StatusRegex.Regex[0]
          and Family =~ FamilyRegex.Regex[0]
          and Type =~ TypeRegex.Regex[0]
          and ( format(format="%v", args=Laddr) =~ IPRegex
              or format(format="%v", args=Raddr) =~ IPRegex )
          and ( format(format="%v", args=Lport) =~ PortRegex
              or format(format="%v", args=Rport) =~ PortRegex )

LET Regions(Pid) = SELECT dict(Offset=Address, Length=Size) AS Sparse
  FROM vad(pid=Pid)
  WHERE Protection =~ "r"
LET dump = SELECT *,
      upload(accessor="sparse",
             file=pathspec(
             Path=serialize(item=Regions(Pid=Pid).Sparse),
             DelegateAccessor="process",
             DelegatePath=format(format="/%d", args=Pid)),
              name=pathspec(Path=format(format="%d.dd", args=Pid))) AS ProcessMemory
  FROM results
LET kill = SELECT *, pskill(pid=Pid) AS KillProcess
  FROM results
LET dumpandkill = SELECT *, pskill(pid=Pid) AS KillProcess
  FROM dump

SELECT * FROM switch(
      a = {
          SELECT *, if(condition= KillProcess=Null,then='Success',else=KillProcess) AS KillProcess
          FROM if(condition= DumpProcess AND KillProcess, then= dumpandkill )},
      b = { SELECT * FROM if(condition= DumpProcess, then= dump )},
      c = {
          SELECT *, if(condition= KillProcess=Null,then='Success',else=KillProcess) AS KillProcess
          FROM if(condition= KillProcess, then= kill)
      },
      catch = results
  )
```

Figure 11.1: Windows.Network.NetstatEnriched give a detailed view of a computer's network connections, beyond the basic netstat command by combining network connection data with info about the processes that are making those connections.

(2) FINDINGS

a. Suspicious file location and name (Known - Confirmation)

Path: C:\Users\IEUser\AppData\Roaming\Wave Google\Jackes.exe. Legitimate Google software does not install under AppData\Roaming\Wave Google.

The folder name “Wave Google” is likely a typo-squatted name mimicking a real vendor to look harmless. Malware authors love blending into AppData\Roaming because it avoids UAC prompts and survives reboots.

b. File hash matches known malware (Known - Confirmation)

SHA256: e015719aad3e56e0d0ea614d0868adf95829164b6f89d2232b6084072b17ab2b

This is the exact QuasarRAT variant we chose from MalwareBazaar earlier in our test. That alone confirms it's malicious.

c. Persistence potential

The AppData\Roaming location suggests the executable could register in Run keys, Scheduled Tasks, or Startup folders, which is common QuasarRAT behaviour.

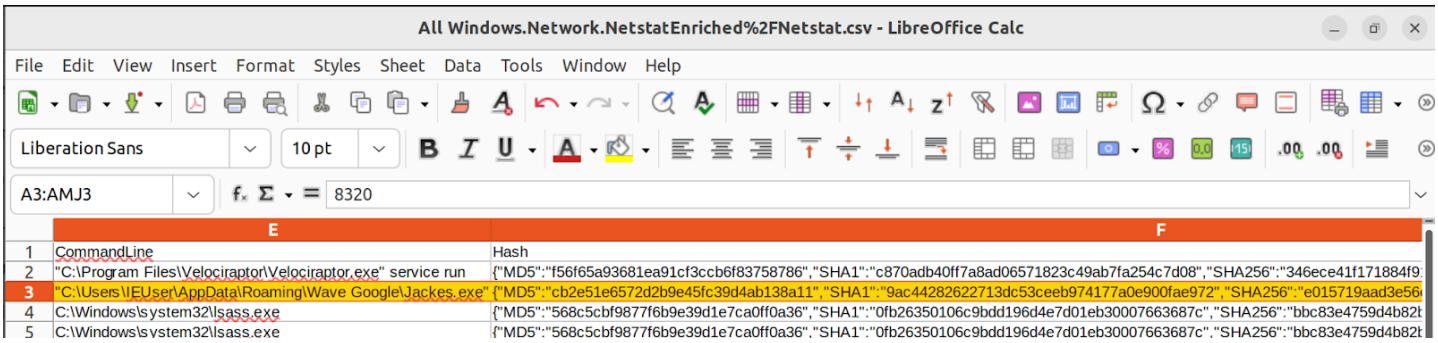


Figure 11.2.1: Windows.Network.NetstatEnriched Output – Command Line path executed + File Hash

d. Certificate trust status (Known - Confirmation)

Velociraptor's enrichment shows **"Trusted":"untrusted".** Legitimate software is usually digitally signed by a vendor certificate that passes Windows trust validation. Malware is often unsigned or uses self-signed certs.

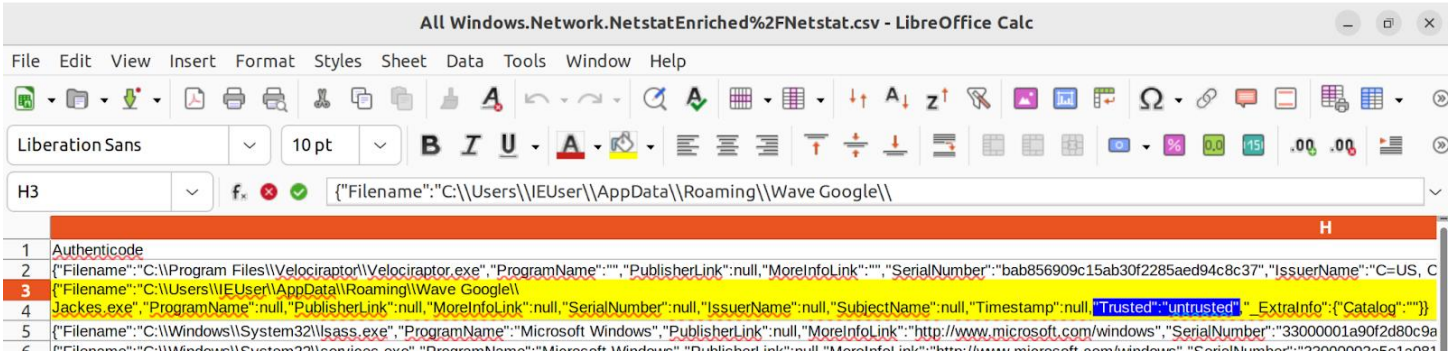


Figure 11.2.2: Windows.Network.NetstatEnriched Output (Continued) – Untrusted Vendor/Cert

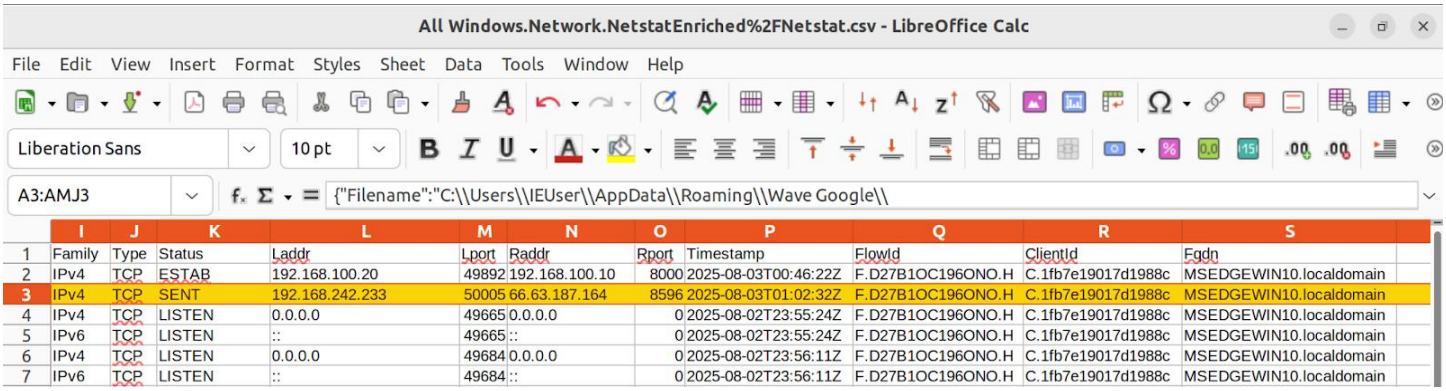
e. Network activity to suspicious remote IP

Local endpoint: **192.168.242.233:50005**

Remote endpoint: **66.63.187.164:8596**

⇨ **That external IP is not part of normal OS update or Microsoft CDN ranges.**

Port 8596 is non-standard and often used by malware for Command and Control (C2) communications.



	I	J	K	L	M	N	O	P	Q	R	S
1	Family	Type	Status	Laddr	Lport	Raddr	Rport	Timestamp	FlowId	ClientId	Fqdn
2	IPv4	TCP	ESTAB	192.168.100.20	49892	192.168.100.10	8000	2025-08-03T00:46:22Z	F.D27B1OC196ONO.H	C.1fb7e19017d1988c	MSEDGEWIN10.localdomain
3	IPv4	TCP	SENT	192.168.242.233	50005	66.63.187.164	8596	2025-08-03T01:02:32Z	F.D27B1OC196ONO.H	C.1fb7e19017d1988c	MSEDGEWIN10.localdomain
4	IPv4	TCP	LISTEN	0.0.0.0	49665	0.0.0.0	0	2025-08-02T23:55:24Z	F.D27B1OC196ONO.H	C.1fb7e19017d1988c	MSEDGEWIN10.localdomain
5	IPv6	TCP	LISTEN	::	49665	::	0	2025-08-02T23:55:24Z	F.D27B1OC196ONO.H	C.1fb7e19017d1988c	MSEDGEWIN10.localdomain
6	IPv4	TCP	LISTEN	0.0.0.0	49684	0.0.0.0	0	2025-08-02T23:56:11Z	F.D27B1OC196ONO.H	C.1fb7e19017d1988c	MSEDGEWIN10.localdomain
7	IPv6	TCP	LISTEN	::	49684	::	0	2025-08-02T23:56:11Z	F.D27B1OC196ONO.H	C.1fb7e19017d1988c	MSEDGEWIN10.localdomain

Figure 11.2.3: Windows.Network.NetstatEnriched Output (Continued) – Suspicious remote IP communication

Verify remote ip 66.63.187.164:8596 via Virus total + MalwareBazaar and found malicious indicator of it being a C2 domain

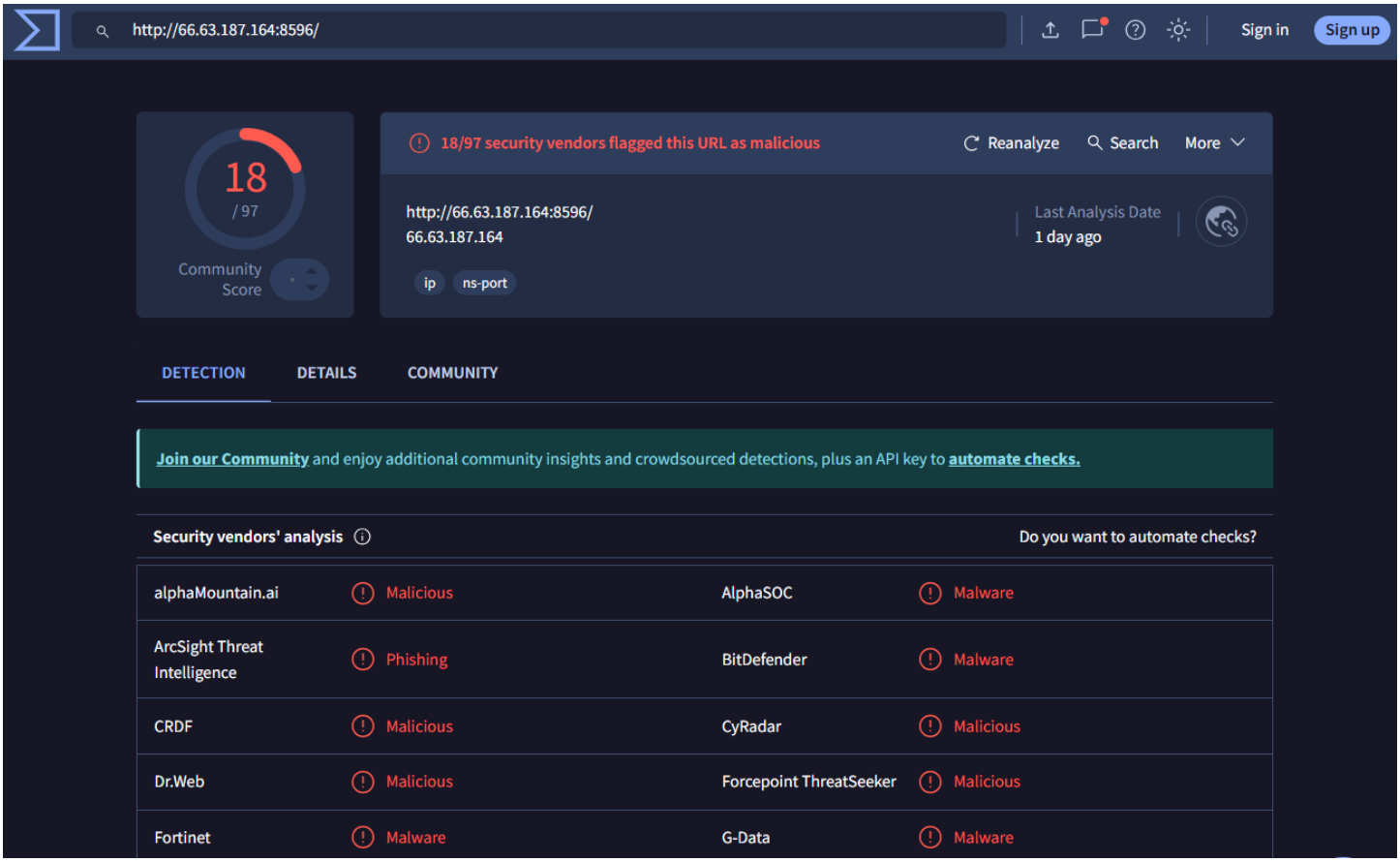


*Figure 11.3.1: Discovered IP domain **66.63.187.164:8596** flagged as Malicious in Virus Total*

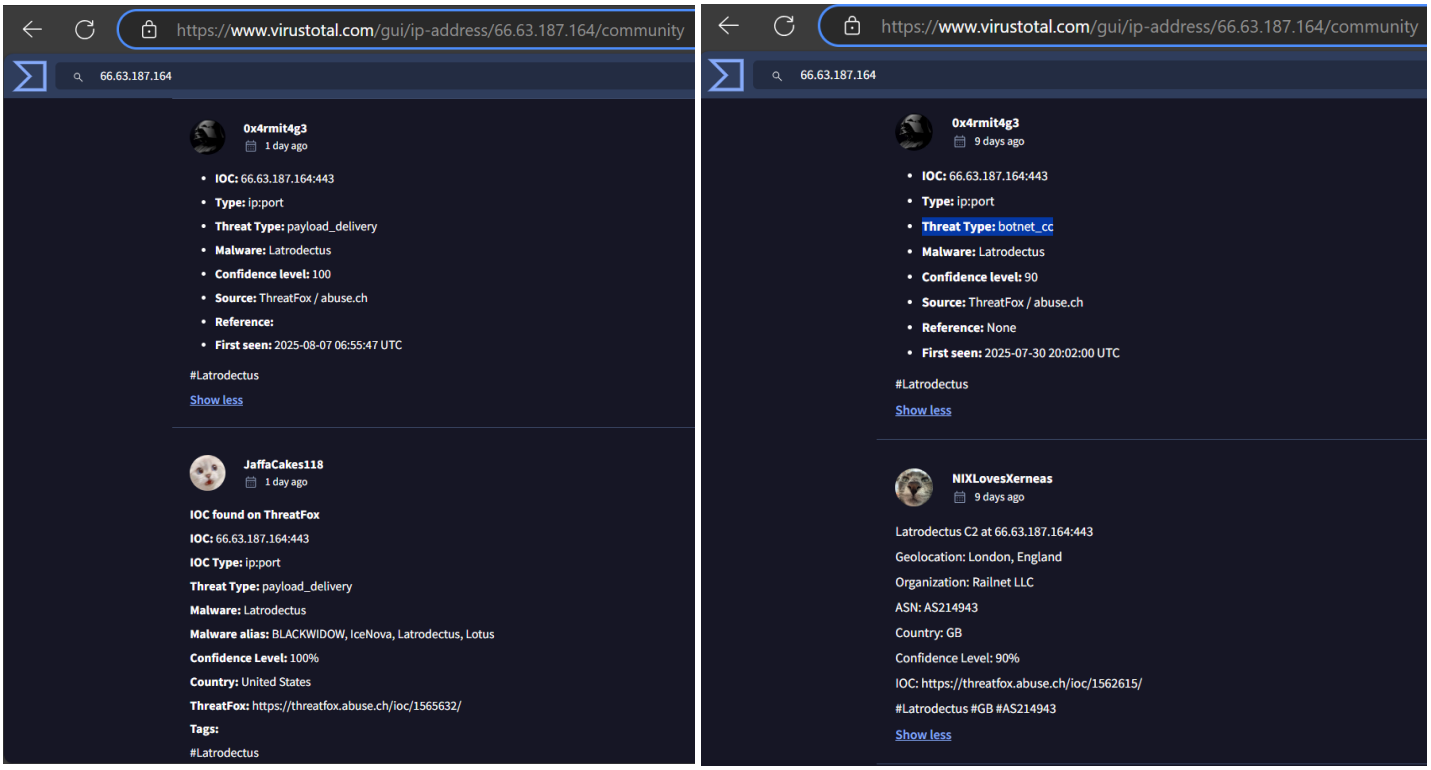


Figure 11.3.2: IOC identified by Virus Total Community

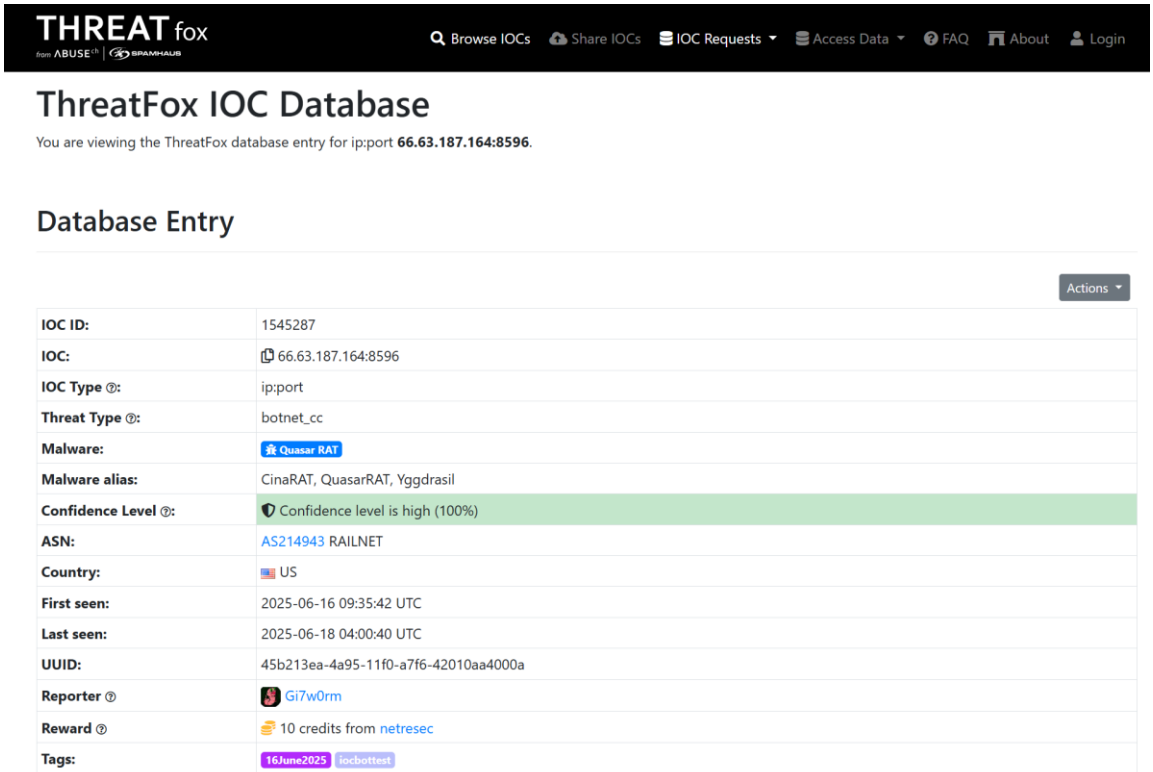


*Figure 11.3.2: MalwareBazaar IoC Database shows IP domain **66.63.187.164** being a C2 center – type: QuasarRAT*

f. Timing correlation

The connection happened right after malware execution time in our test scenario (post-infection hunt), strengthening the case that this process initiated outbound C2.

g. Why this is clearly malicious

In a real SOC report, we'd flag this because:

i. It's a known malicious hash (direct IoC).
ii. It lives in a non-standard install path with vendor name spoofing.
iii. It's unsigned and running from a user space location that allows easy persistence.
iv. It contacts an external IP over a high, non-standard TCP port, likely C2.
v. Behaviour matches known QuasarRAT TTPs: initial beacon, persistence, and potential lateral

3.6. HONOURABLE MENTIONS (VELOCIRAPTOR ARTIFACTS)

3.6.1. WINDOWS.ANALYSIS.EVIDENCEOFEXECUTION/PREFETCH

Prefetch files, which are Windows artifacts used to speed up application startup, would be expected to contain records of `Jackes.exe` being executed. This artifact provides another layer of evidence to confirm the execution history. While no specific data was shown/provided, it would be a standard part of a thorough host analysis.

3.6.2. WINDOWS.ANALYSIS.EVIDENCEOFDOWNLOAD

The EvidenceOfDownload artifact returned no significant results - only the initial download of the malware sample for testing/observation from Malware Bazaar showed up, meaning there was **no Zone.Identifier ADS metadata** present on the files. This is not uncommon and does not indicate the files are benign. Potential reasons for this include:

a. The payload was downloaded via a non-standard method (e.g., Direct transfer through PowerShell/SMB/bitsadmin, or via a loader, another malware stage or a script) – not standard browsers or HTTP Clients
b. The malware or a dropper manually removed the Zone.Identifier ADS.

c. The INetSim environment did not trigger Windows' security tagging for external downloads.

d. The payload was saved in a hidden location different from the typical Downloads folder

Parameters

Name	Type	Default	Description
DirectoryPathGlob	csv	Path C:/Users/*/Downloads//* C:/$Recycle.Bin/*//$R*	
ZoneIdRegex		ZoneId=[34]	A Regular expression to match the required zone (default Internet and Restricted Zones).

Source

```
LET glob_patterns = SELECT Path + ':Zone.Identifier' AS Glob FROM DirectoryPathGlob
LET X = SELECT
   split(string=OSPath, sep=":Zone.Identifier")[0] AS DownloadedFilePath,
   Mtime,
   read_file(filename=OSPath, accessor="ntfs") AS _ZoneIdentifierContent
FROM glob(globs=glob_patterns.Glob, accessor="ntfs")
WHERE NOT IsDir

SELECT *,
  if(condition=DownloadedFilePath, then=hash(path=DownloadedFilePath)) AS FileHash,
  parse_string_with_regex(regex="ZoneId=([^\\r\\n]+)", string=_ZoneIdentifierContent).g1 AS ZoneId,
  parse_string_with_regex(regex="HostUrl=([^\\r\\n]+)", string=_ZoneIdentifierContent).g1 AS HostUrl,
  parse_string_with_regex(regex="ReferrerUrl=([^\\r\\n]+)", string=_ZoneIdentifierContent).g1 AS ReferrerUrl
FROM X

```

Figure 12: Windows.Analysis.EvidenceOfDownload Source Artifact

The artifact VQL identifies files downloaded by a user by searching for the Zone.Identifier alternate data stream. This stream is automatically added to files saved from the internet or an intranet, with a zoneId of 3 or 4 respectively. The artifact returns the file's hash and path, along with the contents of the stream, which can contain valuable information about the download.

3.6.3. WINDOWS.SYSINTERNALS.AUTORUNS & WINDOWS.EVENTLOGS.SCHEDULEDTASKS

A search for persistence mechanisms using these artifacts yielded no results. This is a notable observation and suggests that in this specific simulation, the malware did not establish a standard persistence method such as a `Run` key or a scheduled task. This could be a limitation of the specific malware sample, the simulation setup, or a deliberate design choice by the malware author.

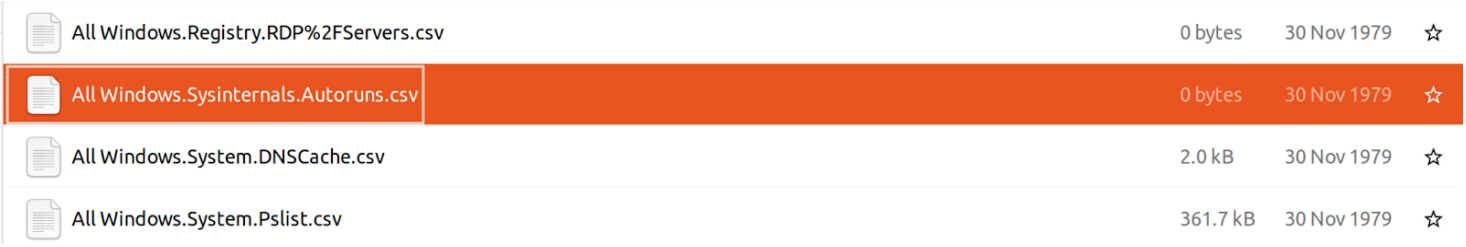


Figure 14.1: Velociraptor Post-Infection Hunt yield no Autoruns findings

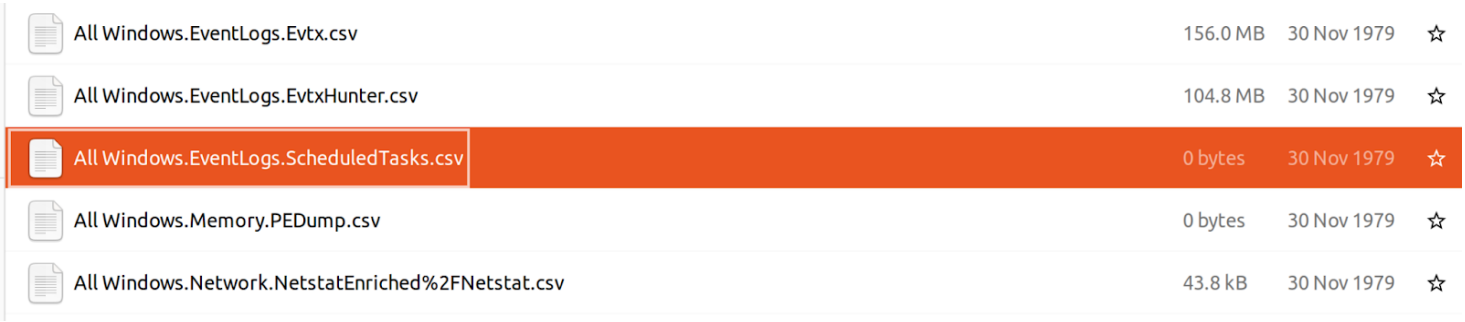


Figure 14.1: Velociraptor Post-Infection Hunt yield no Scheduled Tasks findings

3.7. FILE ANALYSIS SUMMARY

The analysis conclusively demonstrates that the Windows endpoint was compromised by a known QuasarRAT variant. The evidence from multiple sources: **Windows.Detection.BinaryHunter** confirming the file's presence and hash, **Windows.EventLogs.Evtx** pinpointing the exact time of execution, and **Windows.Network.NetstatEnriched** with Wireshark corroborating the C2 communication, forms a strong and undeniable case. **The absence of EvidenceOfDownload** and persistence artifacts serves as a valuable insight into the malware's evasion techniques and the limitations of a standard hunt. The file `Jackes.exe` and its copy mal-sample.exe are confirmed to be the QuasarRAT variant. The file analysis provides the following key points:

- **Hash:** e015719aad3e56e0d0ea614d0868adf95829164b6f89d2232b6084072b17ab2b (SHA-256).
- **Size:** 3,266,048 B.
- **Location:** C:\Users\IEUser\AppData\Roaming\Wave Google\Jackes.exe.
- **Trust:** Untrusted, lacking a digital signature.
- **VirusTotal:** High detection rate, confirming its malicious nature.

IV. NETWORK ANALYSIS

1. SUMMARY

This analysis of network traffic from the infected endpoint `192.168.242.233` following the execution of `Jackes.exe` identified periodic TCP SYN packets sent to a remote server at `66.63.187.164:8596`. These consistent intervals are a strong indicator of Command-and-Control (C2) beaconing behavior associated with the QuasarRAT remote access trojan. The findings are separated into two sections: one for traffic analysis using Wireshark and one for logs from the InetSim server.

2. KEY FINDINGS

A. WIRESHARK REPORTS - SUSPECTED QUASARRAT BEACONING

The investigation focused on the relationship between the infected host and the suspected C2 server. Based on Figure **15.2**, periodic TCP SYN packets were identified from the infected host to a remote server at 66.63.187.164 on port 8596, strongly indicate beaconing behavior.

Table 6: Key Host and Network Identifiers

Attribute	Value
Infected Host IP	192.168.242.233
Suspect Remote IP	66.63.187.164
Suspect Remote Port	8596 (TCP)
Executable Observed	Jackes.exe
Protocol	IPv4 / TCP
Notable Pattern	Regular TCP SYNs without ACKs (*)

(*) Periodic Beaconing Observed: `tshark` command was used to isolate the SYN packets from the infected host. The time deltas between these packets revealed a consistent, alternating pattern of short (approximately **3.3 seconds**) and long (approximately **8.3 seconds**) intervals. This

suggests a programmed, scheduled activity rather than normal user-driven traffic. The infected host sends repeated SYNs to the same remote endpoint without corresponding application data exchange. Such patterns align with **keep-alive** or **polling** methods used by RATs to maintain persistence. The lack of varied ports, protocols, or destinations further suggests a single-purpose C2 connection.

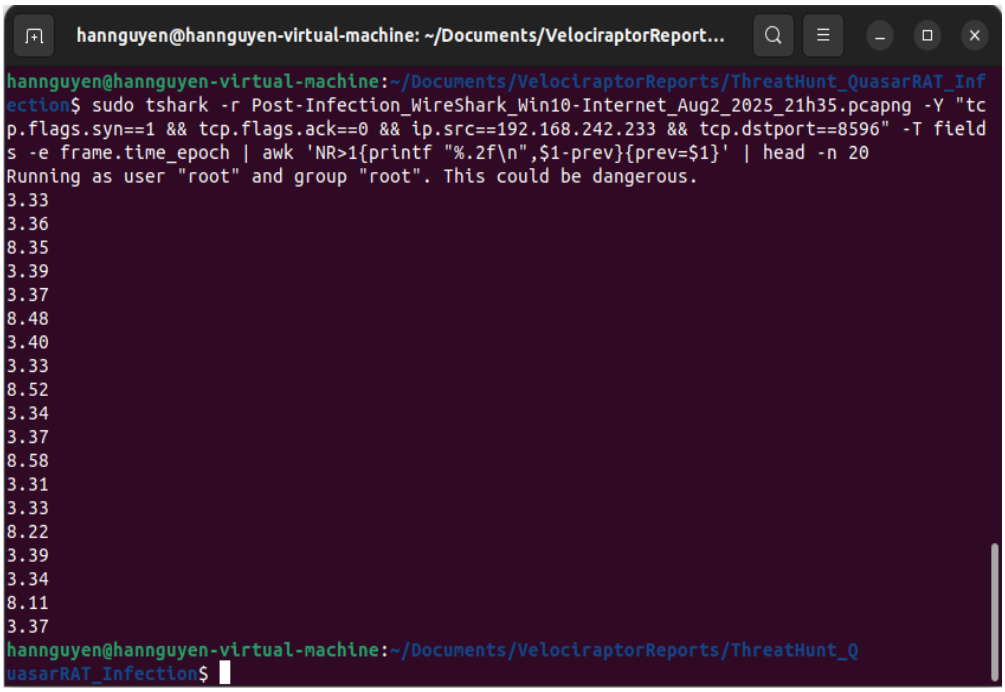
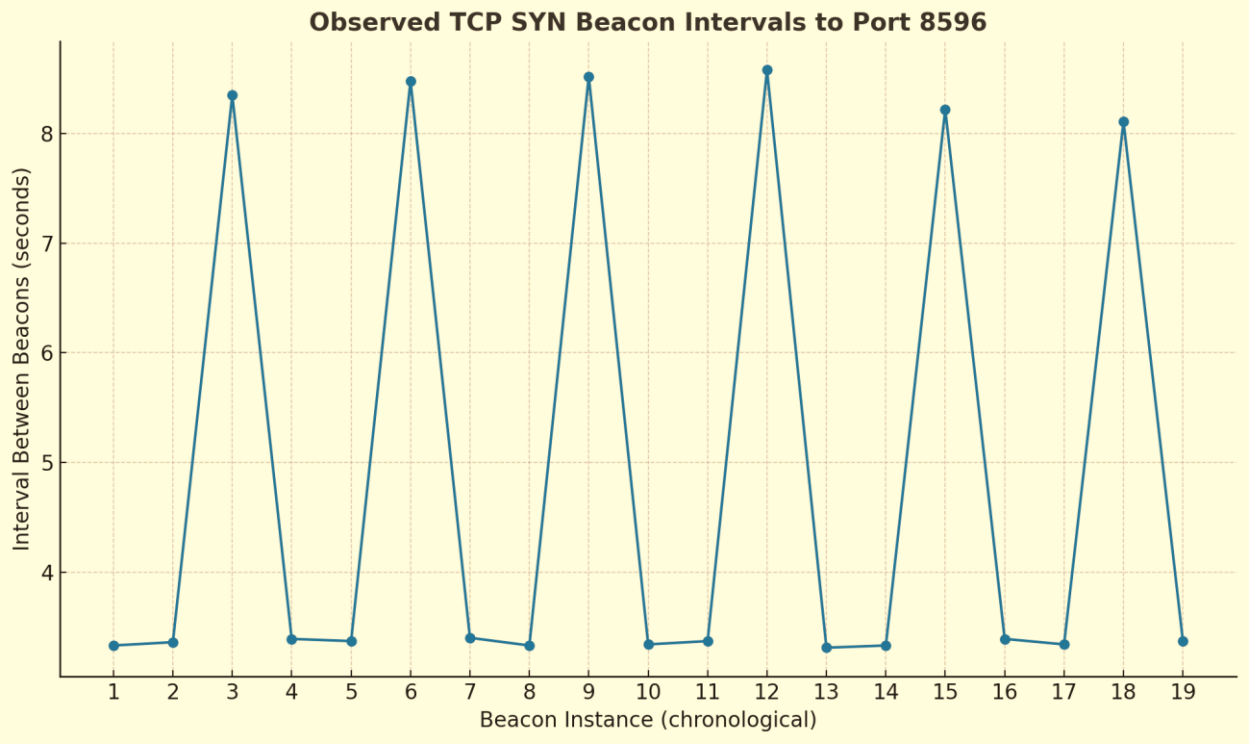


Figure 15.1. Output of tshark command showing beacon intervals.



The network activity correlated with the execution of the unsigned/untrusted binary: `Jackes.exe`, which was found in a suspicious user profile path. The hash of this file confirms it is the QuasarRAT sample. This is shown in **Figure 15.2.**, demonstrating a clean chain of events: process execution followed by immediate outbound C2 attempts.

Not to mention, Repeated **`Conversation completeness: Incomplete`**, **SYN retransmissions**, and **RST/ACK** responses (**Figure 15.2, Figure 15.3**) are typical when malware can't reach or is denied by C2 (server down, sinkholed, blocked, or geo-filtered). The intent to maintain a session is still clear.

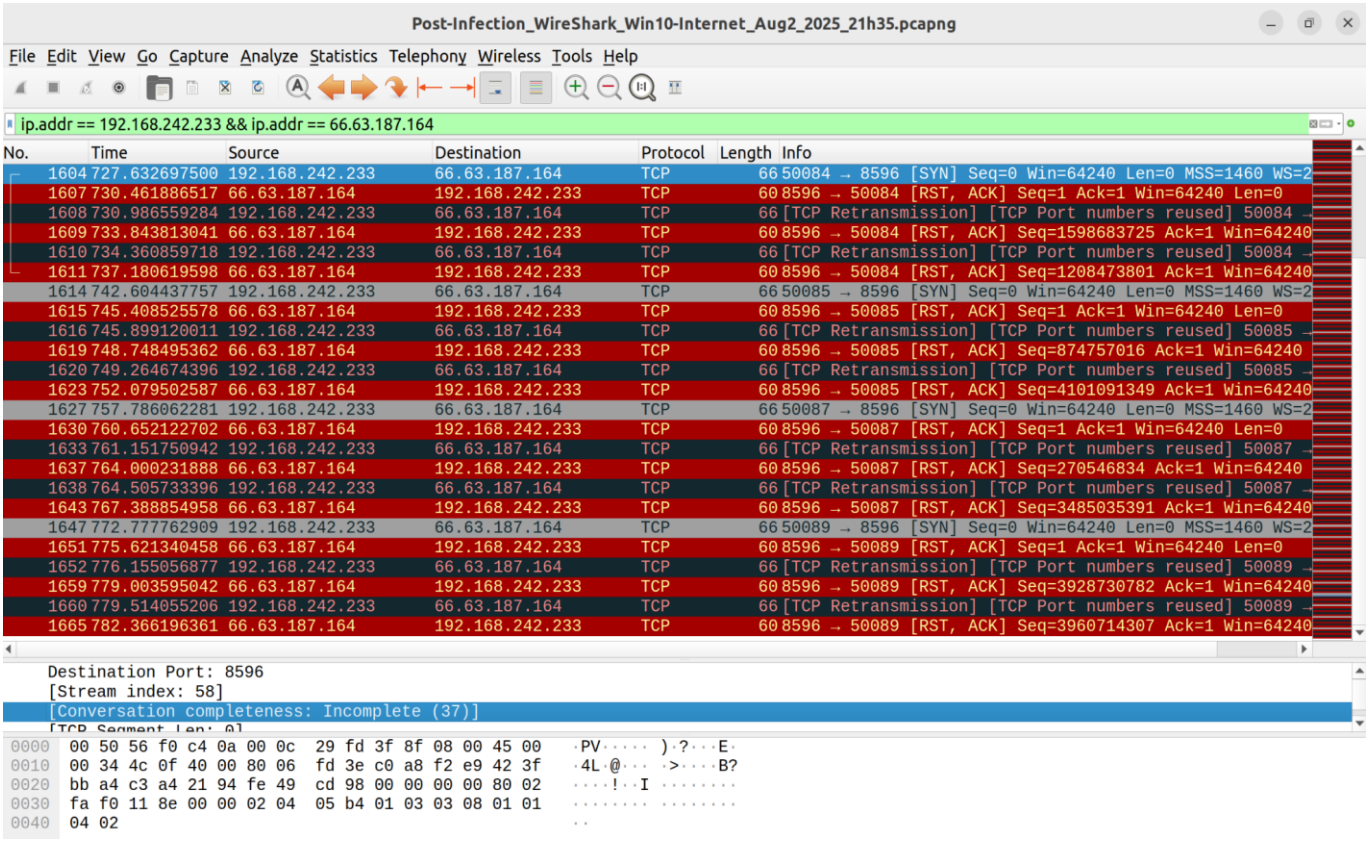


Figure 15.2. Post-infection Wireshark traffic with Beaconing Behavior, C2 filter applied

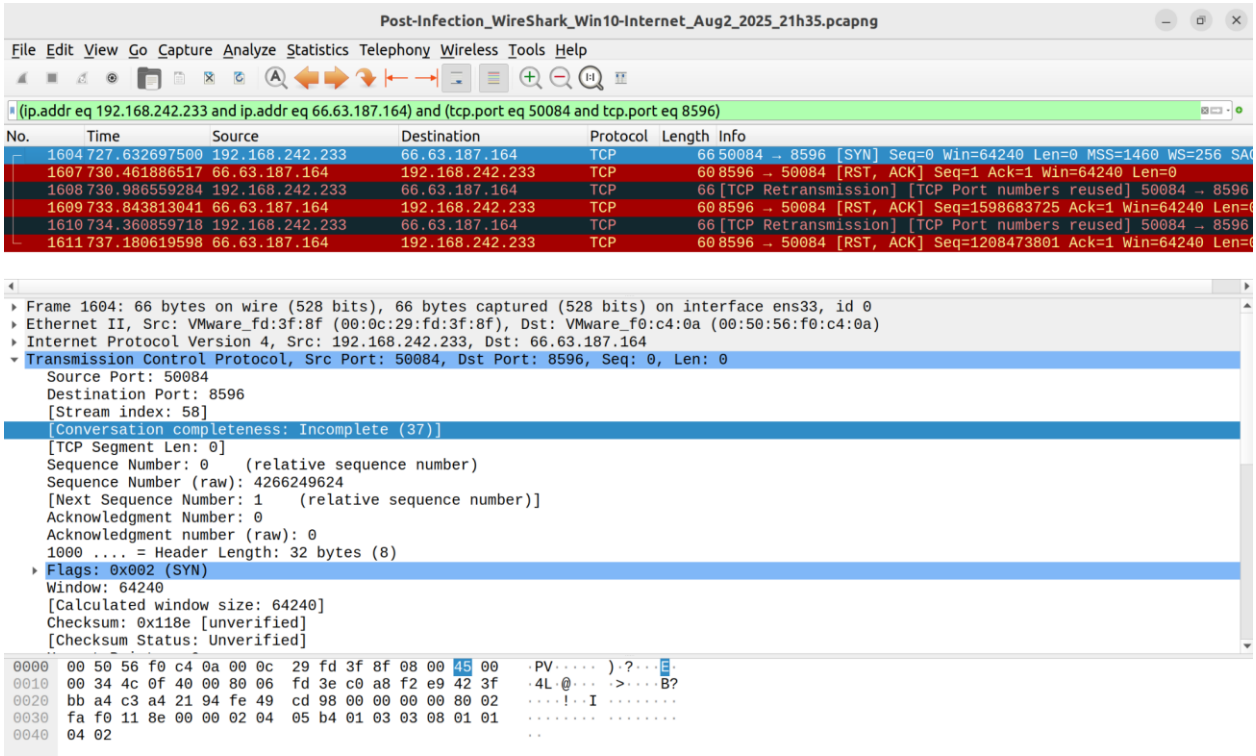


Figure 15.3. Check Statistics → Conversations → TCP and look at the pair 66.63.187.164:8596 for many short/incomplete flows

Overal Assessment and Beacon Characteristics:

The network activity captured in this analysis matches typical RAT beaconing patterns. Given the corroborating host evidence of a malicious executable, it is highly likely that **66.63.187.164:8596** is acting as a QuasarRAT C2 server.

MITRE ATT&CK:

⇨ T1071.001 – Application Layer Protocol (web-like C2 over TCP)

⇨ T1095 – Non-application layer protocols (raw TCP to non-standard port)

Risk: C2 beaconing allows tasking, data theft, and persistence management once the server answers.

Confidence: **High**, due to hash-based attribution to QuasarRAT, untrusted binary in suspicious path, fixed-interval SYNs to a high port, and tight timing correlation with execution logs.

Table 7: Summary of C2 Beacon Characteristics

Attribute	Value
Beacon Destination	66.63.187.164:8596
Interval Pattern	~3.3s and ~8.3s alternation
Protocol	TCP
SYN without ACK	Yes
Associated Process	Jackes.exe
Host OS	Windows 10 (MSEdge Win10 VM)
Likely Malware Family	QuasarRAT

B. INETSIM CLEAN TRAFFIC BASELINE

An analysis of InetSim logs was performed to identify any indicators of compromise or other anomalous traffic. The investigation revealed that the traffic in this dataset was not necessarily malicious, since the log primarily contained expected background network noise for a host with an active browser.

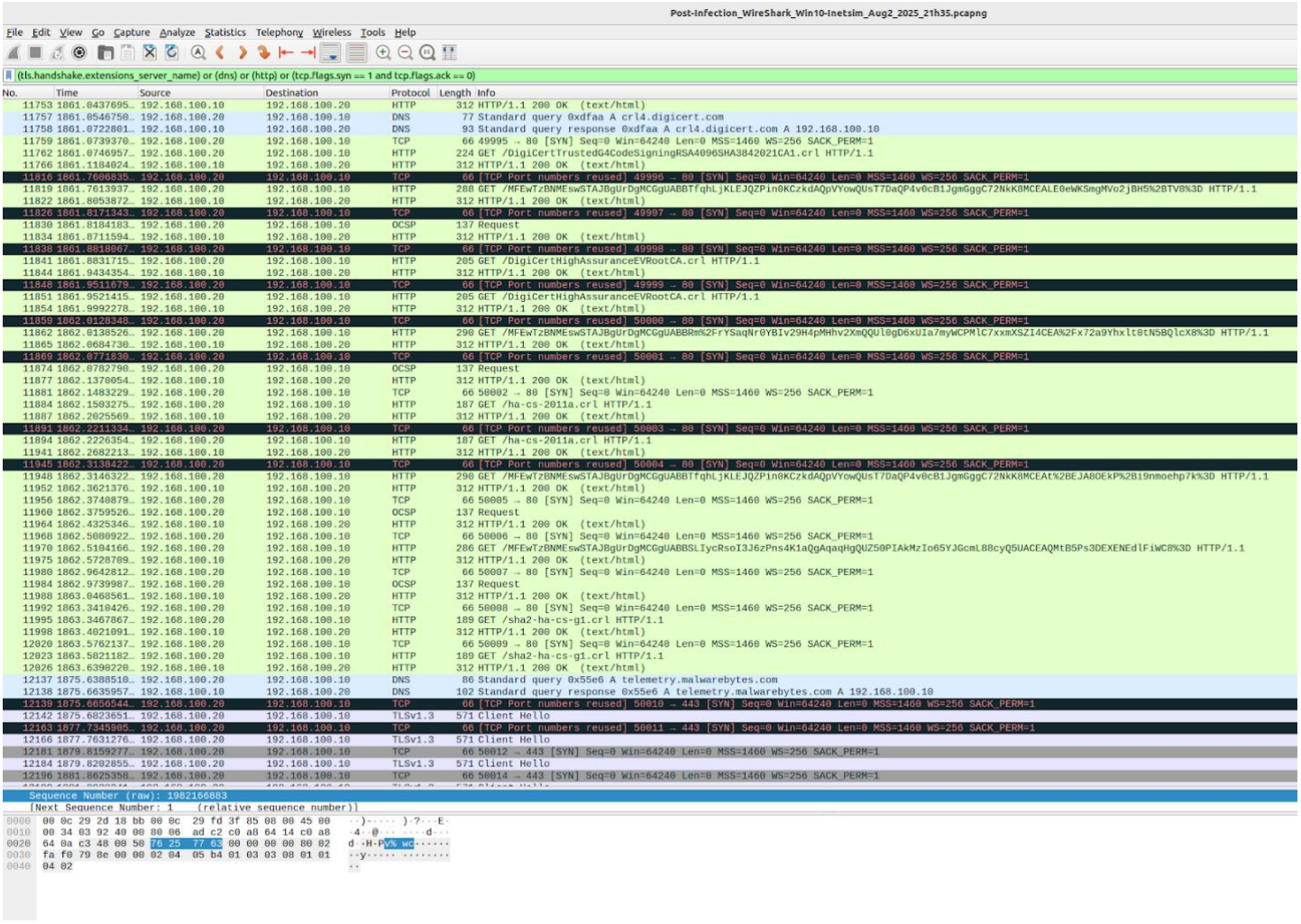


Figure 16. Wireshark Traffic with InetSim logs

This traffic mostly includes DNS lookups for Mozilla services; certificate validation traffic to Digicert and Microsoft OCSP/CRL endpoints; HTTP GET/POST activity related to certificate revocation checks (GET/200 OK), SYN packets to various ports that are then reused, which can be normal during repeated HTTP/TLS connections in a simulated environment.

No evidence of known QuasarRAT C2 domains or IPs, unusual frequencies, or data exfiltration was found within this dataset. The reviewed session aligns with expected background

network noise for a host with an active browser and standard operating system certificate verification processes.

Significance

While no malicious activity was identified in this log, documenting this outcome is critical for establishing a clean baseline for this timeframe. It also narrows the investigative focus to other data sources, such as live PCAP captures and Velociraptor endpoint artifacts. It is important to note that while InetSim successfully simulated normal services, certain patterns like multiple quick certificate requests or port reuse might be worth mentioning in a report as observed but non-suspicious behaviors within this controlled setup.

V. POST-ERADICATION ANALYSIS

After the initial analysis, a second phase of the exercise was conducted to simulate the eradication of the malware and verify its success. This is a critical step in a DFIR process to ensure the threat has been completely neutralized and to prevent re-infection.The eradication process involved:

1. Deleting the malicious executable Jackes.exe from its discovered location.
2. Removing the malicious registry entry under the Run key.

Following the eradication, another Velociraptor hunt was performed. The results of this post-eradication hunt were compared against the post-infection state to confirm that the malicious file and its persistence mechanism were no longer present on the system. This verification step is crucial for an analyst to confidently close an incident, ensuring no residual IOCs remain. This final step confirms the effectiveness of the remediation actions and provides a clear audit trail of the entire DFIR process.

Even after manual eradication and a Malwarebytes scan, some artifacts of the malware persisted, as detected by Windows Security. A final re-analysis was conducted to ensure all traces were removed. The differential analysis using the Python scripts helped to confirm the absence of the key malicious hashes, providing strong confidence in the eradication process.

VI. COMPARISON, CONTRAST, AND DEEP ANALYSIS OF VARIOUS ARTIFACTS

Table 8: Summarizing the role of each artifact in building a comprehensive picture of the attack

Artifact	Type	Purpose	Key Data Points	Integration and Use Cases
Windows.Detection.BinaryHunter	Host-based	Hunting for files with specific attributes and hashes.	File paths, hashes (MD5, SHA1, SHA256), Authenticode status, PE information.	Confirmed the presence of the malicious file and its properties. Provided the hash for cross-referencing with VirusTotal.
Windows.EventLogs.Evtx	Host-based	Capturing execution events and system activity.	Timestamp, Event ID, Process ID, Command Line.	Provided the exact time of execution, which was crucial for correlating with network activity.
Windows.System.Pslist	Host-based	Listing running processes.	Process name, PID, parent PID.	Confirmed the malicious process was active on the system during the investigation period.
Windows.Network.NetstatEnriched	Host-based & Network	Capturing and enriching network connections.	Local/Remote IP and port, process name, timestamp, and hostname.	Provided the critical link between the Jackes.exe process and the C2 server, confirming network communication.
Windows.Analysis.EvidenceOfDownload	Host-based	Identifying how a file arrived on the system.	Zone.Identifier ADS metadata.	Its absence was a key finding, suggesting the malware used a non-standard infection vector or removed its metadata.
Wireshark Capture	Network-based	Deep packet inspection and traffic analysis.	Source/Destination IP, ports, packet payloads, and timing.	Independently validated the network connection identified by NetstatEnriched and provided context on the nature of the C2 traffic.
VirusTotal	Open-Source Intelligence	Threat reputation and hash analysis.	Detection scores, known malicious vendors, community scores.	Confirmed the Jackes.exe hash was a known malicious sample, validating the initial hypothesis.

VII. CONCLUSION AND RECOMMENDATIONS

This report provides a comprehensive account of a threat hunting exercise that successfully identified and analyzed a **QuasarRAT** malware sample. The use of a structured methodology and advanced tools like **Velociraptor, INetSim,** and **custom Python scripts** enabled the discovery of key IOCs and TTPs, including file execution artifacts, a covert persistence mechanism, and a behavioral profile of its network communication attempts.

Based on the findings from this analysis, the following recommendations are provided for a real-world incident response scenario:

1. Isolate and Eradicate: Immediately isolate any host found with these IOCs. Eradicate the malicious file and its persistence mechanism as demonstrated in the post-eradication phase.
2. IOC Blocking: Add the identified file hash and other IOCs to all perimeter and endpoint security solutions to prevent future infections.
3. Behavioral Monitoring: Implement behavioral monitoring rules that can detect anomalous registry key modifications, especially in Run keys, and unexpected network connections.
4. Endpoint Visibility: Advocate for the continued use of endpoint visibility tools like Velociraptor to enable proactive threat hunting and rapid incident response.

REFERENCE LIST

[1] CrowdStrike. "What is Cyber Threat Hunting? [Proactive Guide]." *CrowdStrike*, 11 Aug. 2025, https://www.crowdstrike.com/en-us/cybersecurity-101/threat-intelligence/threat-hunting/.

[2] Psmths. "Amcache.hve." *GitHub*, 2025, https://github.com/Psmths/windows-forensic-artifacts/blob/main/execution/amcache.md

[3] "QuasarRAT (e015719aad3e56...)." *VirusTotal*, 12 Aug. 2025, https://www.virustotal.com/gui/file/e015719aad3e56e0d0ea614d0868adf95829164b6f89d2232b6084072b17ab2b/details.

[4] Velociraptor, version 0.7.1, 2025, https://docs.velociraptor.app/.

[5] Wireshark, version 4.2.5, 2025, https://www.wireshark.org/.

[6] MalwareBazaar, 2025, https://bazaar.abuse.ch/.

[7] MyDFIR. "FREE Cybersecurity Tool: Velociraptor (Step-By-Step Guide)." YouTube, 23 December 2023, http://www.youtube.com/watch?v=p9pQ2g-18o4.

APPENDIX

1. DIFFERENTIAL ANALYSIS WITH PYTHON SCRIPTS

1.1. DFF_UTIL_EXECPROOF.PY

i. **Velociraptor Amcache Differential Scanner**

This Python script compares two Velociraptor Amcache artifact CSV files - one from a clean (baseline) system and one from a potentially infected system - to detect new or suspiciously modified executables that may have been introduced during malware execution (e.g., QuasarRAT).

j. **Key Features:**

- Suspicious Path Filtering: Narrows down analysis to entries with paths commonly used by malware (e.g., %AppData%, Temp, Public, etc.).
- SHA Hash Comparison: Finds new or altered executable hashes in the infected system that didn't exist in the baseline.
- Interactive CLI: Asks the user to input file paths and provides color-coded progress messages with animated spinners.
- Readable Report: Outputs a CSV listing suspicious entries unique to the infected state.

k. **Purpose:**

This tool helps threat hunters and DFIR analysts quickly isolate potentially malicious binaries by spotlighting new file executions in shady locations after an infection.

l. **Source code for dff_util_execProof.py for easier differentiation:**

```
import pandas as pd
import os
import re
from colorama import Fore, Style, init
import time
from itertools import cycle

# Initialize colorama
init(autoreset=True)

# Suspicious keywords for filtering
```

```
SUSPICIOUS_KEYWORDS = ['AppData', 'Temp', 'Roaming', 'Local',
'Startup', 'Tasks', 'Quasar', 'qRAT', 'payload', 'client.exe',
'stub.exe']

def load_csv(path):
    try:
        df = pd.read_csv(path, encoding='utf-8', engine='python')
        print(f"{Fore.GREEN}[+] Successfully loaded {path} ({len(df)}
rows)")
        return df
    except Exception as e:
        print(f"{Fore.RED}[!] Error loading {path}: {e}")
        return pd.DataFrame()

def filter_suspicious(df):
    suspicious = pd.DataFrame()
    for col in df.columns:
        if df[col].dtype == object:
            mask =
df[col].astype(str).str.contains('|'.join(SUSPICIOUS_KEYWORDS),
case=False, na=False)
            suspicious = pd.concat([suspicious, df[mask]], axis=0)
    return suspicious.drop_duplicates()

def compare_hashes(before_df, after_df):
    sha_cols = ['SHA1', 'SHA256', 'SHA-1', 'SHA-256']
    found_sha_col = None
    for col in sha_cols:
        if col in before_df.columns and col in after_df.columns:
            found_sha_col = col
            break
    if not found_sha_col:
        print(f"{Fore.RED}[!] SHA column not found in either CSV.")
        return pd.DataFrame()

    before_hashes = set(before_df[found_sha_col].dropna().unique())
    after_hashes = set(after_df[found_sha_col].dropna().unique())

    new_hashes = after_hashes - before_hashes
    old_but_suspicious = before_hashes.intersection(after_hashes)

    new_entries = after_df[after_df[found_sha_col].isin(new_hashes)]
    existing_suspicious =
after_df[after_df[found_sha_col].isin(old_but_suspicious)]

    return new_entries, existing_suspicious

def spinner_animation(task_name, duration=2):
    spinner = cycle(['|', '/', '-', '\\'])
    end_time = time.time() + duration
    while time.time() < end_time:
```

```
        print(f"\r{Fore.CYAN}[{next(spinner)}] {task_name}...",
end='', flush=True)
        time.sleep(0.1)
    print("\r", end='')

def main():
    print(f"{Fore.CYAN}Velociraptor Amcache Differential Scanner")
    print("="*40)

    before_path = input("Enter the full path to the baseline (before
infection) Amcache CSV file: ").strip()
    after_path = input("Enter the full path to the infected (after
infection) Amcache CSV file: ").strip()

    if not os.path.isfile(before_path):
        print(f"{Fore.RED}[!] File not found: {before_path}")
        return
    if not os.path.isfile(after_path):
        print(f"{Fore.RED}[!] File not found: {after_path}")
        return

    before_df = load_csv(before_path)
    after_df = load_csv(after_path)

    spinner_animation("Filtering suspicious paths in baseline")
    before_suspicious = filter_suspicious(before_df)
    print(f"{Fore.YELLOW}[~] Found {len(before_suspicious)}
suspicious entries in baseline")

    spinner_animation("Filtering suspicious paths in infected
sample")
    after_suspicious = filter_suspicious(after_df)
    print(f"{Fore.YELLOW}[~] Found {len(after_suspicious)} suspicious
entries in infected")

    spinner_animation("Detecting suspicious hashes")
    new_hash_df, reused_suspicious_df =
compare_hashes(before_suspicious, after_suspicious)

    print(f"{Fore.YELLOW}[~] New suspicious hashes:
{len(new_hash_df)}")
    print(f"{Fore.YELLOW}[~] Suspicious but existing hashes:
{len(reused_suspicious_df)}")

    if not new_hash_df.empty:
        new_hash_df.to_csv("new_suspicious_hashes.csv", index=False)
        print(f"{Fore.GREEN}[+] New suspicious entries saved to
new_suspicious_hashes.csv")
    if not reused_suspicious_df.empty:
        reused_suspicious_df.to_csv("existing_suspicious_hashes.csv",
index=False)
```

```
        print(f"{Fore.GREEN}[+] Existing suspicious entries saved to
existing_suspicious_hashes.csv")

main()
```

1.2. AMCACHE_DIFF_WITH_VT.PY

Improved on the tool **dff_util_execProof.py** to diff baseline and post-infection amcache entries by feeding the suspicious new hashes identify into a function which calls/requests Virus Total API to check and report status on these hashes along with the link to virus total web GUI vendor assessment for easier navigation and timesaving. The resulting report includes 2 files - **suspicious_amcache_diff.csv and suspicious_amcache_virustotal_report.csv**

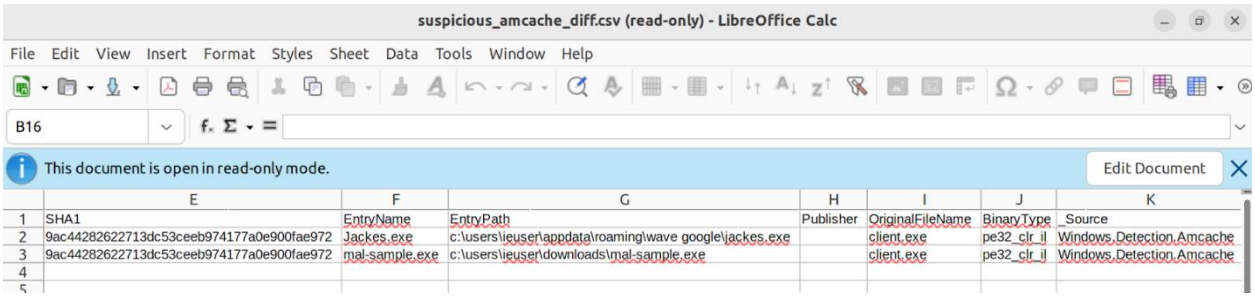


Figure A.1. suspicious_amcache_diff.csv

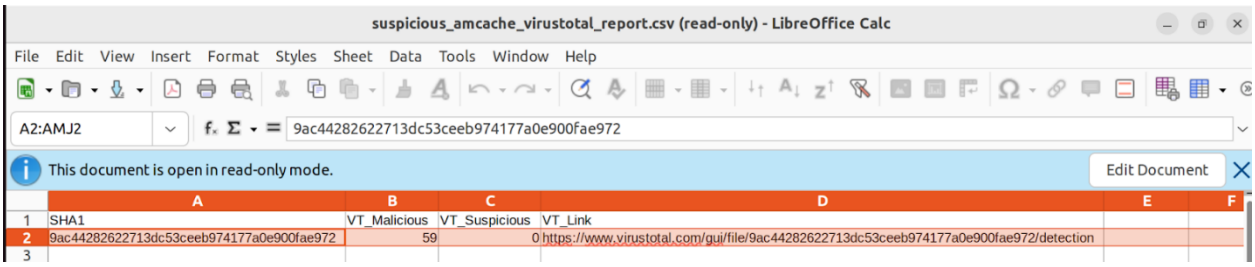


Figure A.2.suspicious_amcache_virustotal_report.csv

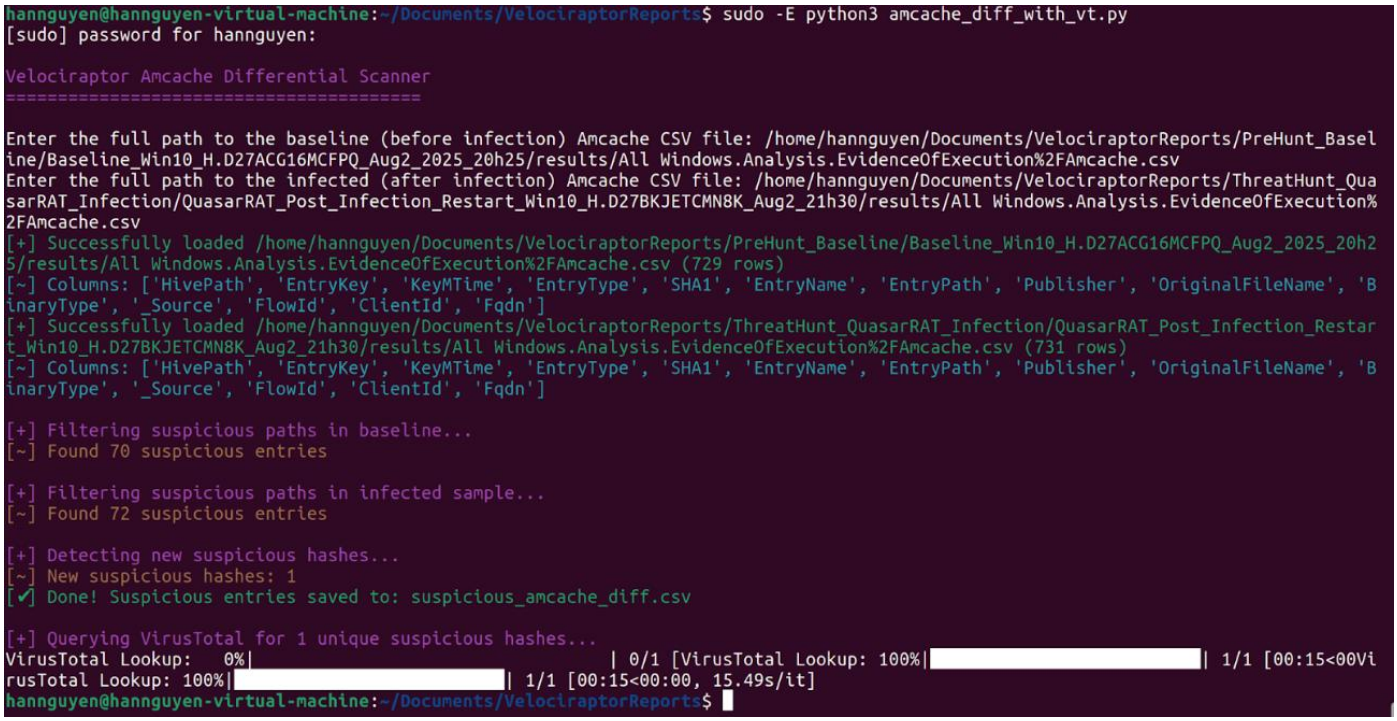


Figure A.3. The GUI of the tools will take 2 inputs in CSVs format, as long as full path provided

Source code of **amcache_diff_with_vt.py tool,** with **virus total check of suspicious hash**

```
import pandas as pd
import os
import requests
from colorama import Fore, Style, init
from tqdm import tqdm
from time import sleep

init(autoreset=True)

VT_API_KEY = [ENTER YOUR OWN VIRUS TOTAL API KEY HERE]
VT_URL = "https://www.virustotal.com/api/v3/files/"

def print_banner():
    print(Fore.MAGENTA + "\nVelociraptor Amcache Differential Scanner")
    print(Fore.MAGENTA + "========================================\n")

def load_csv(file_path):
    try:
        df = pd.read_csv(file_path, encoding='utf-8', low_memory=False)
        print(Fore.GREEN + f"[+] Successfully loaded {file_path} ({len(df)} rows)")
        print(Fore.CYAN + f"[~] Columns: {list(df.columns)}")
        return df
    except FileNotFoundError:
        print(Fore.RED + f"[!] File not found: {file_path}. Try again.")
        return None
    except Exception as e:
        print(Fore.RED + f"[!] Error loading {file_path}: {e}")
```

```
        return None

def filter_suspicious(df):
    if 'EntryPath' not in df.columns:
        print(Fore.RED + "[!] 'EntryPath' column not found in CSV.")
        return pd.DataFrame()

    keywords = ['AppData', 'Temp', 'Roaming', 'Local', 'System32', 'Users\\',
'Public','Downloads']
    filtered_df = df[df['EntryPath'].astype(str).str.contains('|'.join(keywords),
case=False, na=False)]
    print(Fore.YELLOW + f"[~] Found {len(filtered_df)} suspicious entries")
    return filtered_df

def compare_hashes(before_df, after_df):
    hash_column = None
    for candidate in ['SHA1', 'SHA256', 'Hash']:
        if candidate in after_df.columns and candidate in before_df.columns:
            hash_column = candidate
            break

    if not hash_column:
        print(Fore.RED + "[!] No hash column (SHA1, SHA256) found in both files.")
        return pd.DataFrame()

    before_hashes = set(before_df[hash_column].dropna().unique())
    after_hashes = set(after_df[hash_column].dropna().unique())

    new_hashes = after_hashes - before_hashes
    print(Fore.YELLOW + f"[~] New suspicious hashes: {len(new_hashes)}")

    return after_df[after_df[hash_column].isin(new_hashes)]

def query_virustotal(hash_value):
    headers = {
        "x-apikey": VT_API_KEY
    }
    response = requests.get(VT_URL + hash_value, headers=headers)

    if response.status_code == 200:
        data = response.json()
        stats = data.get("data", {}).get("attributes",
{}).get("last_analysis_stats", {})
        malicious = stats.get("malicious", 0)
        suspicious = stats.get("suspicious", 0)
        link = f"https://www.virustotal.com/gui/file/{hash_value}/detection"
        return malicious, suspicious, link
    elif response.status_code == 404:
        return 0, 0, "Not Found"
    else:
        return None, None, "Error"

def enrich_with_virustotal(diff_df, hash_column):
    unique_hashes = diff_df[hash_column].dropna().unique()
```

```
    print(Fore.MAGENTA + f"\n[+] Querying VirusTotal for {len(unique_hashes)}
unique suspicious hashes...")

    vt_results = []

    for h in tqdm(unique_hashes, desc="VirusTotal Lookup", ncols=80):
        malicious, suspicious, link = query_virustotal(h)
        if malicious is None:
            print(Fore.RED + f"[!] Error querying hash: {h}")
            continue
        vt_results.append({
            hash_column: h,
            "VT_Malicious": malicious,
            "VT_Suspicious": suspicious,
            "VT_Link": link
        })
        sleep(15)  # Respect public API rate limit

    vt_df = pd.DataFrame(vt_results)
    report_file = "suspicious_amcache_virustotal_report.csv"
    vt_df.to_csv(report_file, index=False)
    print(Fore.GREEN + f"[✓] VirusTotal report saved as: {report_file}")

def main():
    print_banner()

    before_file = input("Enter the full path to the baseline (before infection)
Amcache CSV file: ").strip().strip('"')
    while not os.path.isfile(before_file):
        print(Fore.RED + f"[!] File not found: {before_file}. Try again.")
        before_file = input("Enter the full path to the baseline (before infection)
Amcache CSV file: ").strip().strip('"')

    after_file = input("Enter the full path to the infected (after infection)
Amcache CSV file: ").strip().strip('"')
    while not os.path.isfile(after_file):
        print(Fore.RED + f"[!] File not found: {after_file}. Try again.")
        after_file = input("Enter the full path to the infected (after infection)
Amcache CSV file: ").strip().strip('"')

    before_df = load_csv(before_file)
    after_df = load_csv(after_file)
    if before_df is None or after_df is None:
        print(Fore.RED + "[!] Failed to load both files. Exiting.")
        return

    print(Fore.MAGENTA + "\n[+] Filtering suspicious paths in baseline...")
    before_suspicious = filter_suspicious(before_df)

    print(Fore.MAGENTA + "\n[+] Filtering suspicious paths in infected sample...")
    after_suspicious = filter_suspicious(after_df)

    print(Fore.MAGENTA + "\n[+] Detecting new suspicious hashes...")
    for i in range(3):
        print(Fore.YELLOW + f"Comparing hashes {'.' * (i+1)}", end='\r')
```

```
        sleep(0.5)

    diff_df = compare_hashes(before_suspicious, after_suspicious)

    if not diff_df.empty:
        output_file = "suspicious_amcache_diff.csv"
        diff_df.to_csv(output_file, index=False)
        print(Fore.GREEN + f"[✓] Done! Suspicious entries saved to: {output_file}")

        # Proceed with VirusTotal enrichment
        enrich_with_virustotal(diff_df, hash_column='SHA1')

    else:
        print(Fore.CYAN + "[i] No new suspicious hashes detected. No report
generated.")

if __name__ == "__main__":
    main()
```