


# Hunting for Lateral Movement using Event Query Language

 [elastic.co/blog/hunting-for-lateral-movement-using-event-query-language](https://elastic.co/blog/hunting-for-lateral-movement-using-event-query-language)

March 18, 2021



Lateral Movement describes techniques that adversaries use to pivot through multiple systems and accounts to improve access to an environment and subsequently get closer to their objective. Adversaries might install their own remote access tools to accomplish Lateral Movement, or use stolen credentials with native network and operating system tools that may be stealthier in blending in with normal systems administration activity.

Detecting Lateral Movement behaviors often involves the design of detections at both the source and the target system, as well as the correlation of more than one type of event (such as network events with process execution events) in order to capture the remote execution context.

In this blog, we explore some examples of techniques and leverage the capabilities of Elastic's Event Query Language (EQL) to design behavioral hunts and detections.

## How Lateral Movement works

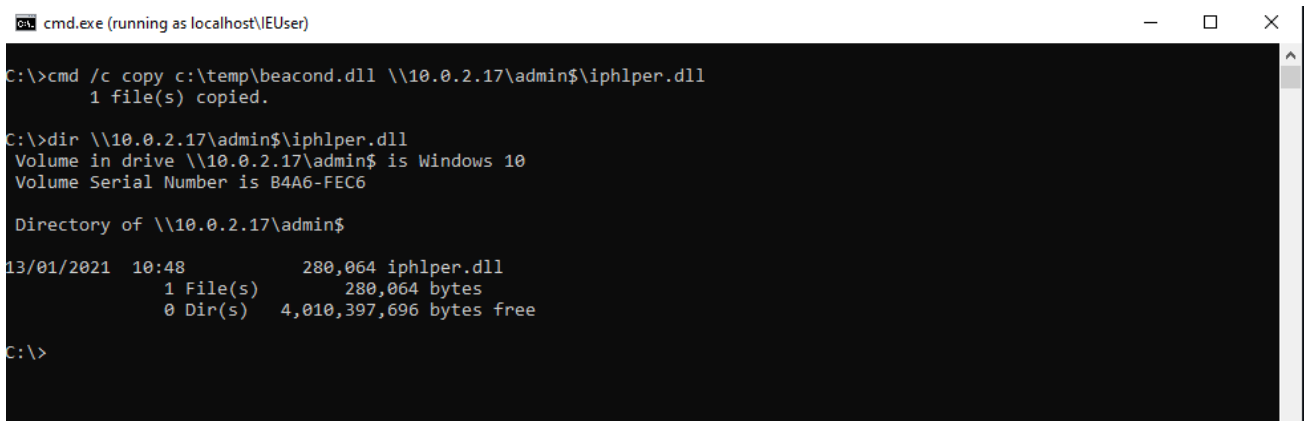
Lateral Movement is usually composed of the following high-level steps:

1. Remote authentication to the target host (valid access credentials are required)
2. Staging the command to execute to the remote host or to another resource accessible by the target host such as internet URL or a Network File Share
3. Remotely triggering the execution (immediate or scheduled) of the staged program on the target host via accessible remote services and protocols (Service, Task Scheduler, WinRM, WMI, Remote Registry).
4. Clean up the staged payload and any other relevant artifacts to avoid suspicion (optional)

Note that staging a program (step 2) is not always necessary, as there are usually exposed services that allow for remote interaction with the target host such as PowerShell Remoting and Remote Desktop (RDP).

## Lateral Tool Transfer

Files may be copied from one system to another to stage adversary tools or other files over the course of an operation. A commonly abused vector is SMB/Windows Admin Shares via the use of built-in system commands such as `copy` , `move` `copy-item` , and others:



```

cmd.exe (running as localhost\IEUser)
C:\>cmd /c copy c:\temp\beacond.dll \\10.0.2.17\admin$\iphelper.dll
1 file(s) copied.

C:\>dir \\10.0.2.17\admin$\iphelper.dll
Volume in drive \\10.0.2.17\admin$ is Windows 10
Volume Serial Number is B4A6-FEC6

Directory of \\10.0.2.17\admin$

13/01/2021  10:48          280,064 iphelper.dll
             1 File(s)          280,064 bytes
             0 Dir(s)  4,010,397,696 bytes free

C:\>

```

Figure 1: File copy via system command

From the source machine, there are alternative methods of copying the file without having to execute suspicious commands. Still, it's important to look for low-hanging detection opportunities.

Figure 2 below shows an EQL query that looks for the following behavior that is consistent with an attacker transferring a file to a remote host:

- Execution of a command interpreter with a `process.args` keyword array related to file copy ( `copy` , `move` ) and a hidden file share (prefixed by a `$` sign such as `c$admin$` )
- Staging data from a shadow copy volume (often associated with credential access via staging of NTDS.dit or Registry SAM key to access stored account password hashes)

```
process where event.type == "start" and
  process.name : ("cmd.exe", "powershell.exe", "robocopy.exe", "xcopy.exe") and
  process.args : ("copy*", "move*", "cp", "mv") and
  process.args : ("*$*", "*HarddiskVolumeShadowCopy*")
```

Figure 2: Hunting EQL for file transfer via hidden file share from source machine  
On the target machine, we've observed that all files copied via server message block (SMB) are represented by a file creation event by the virtual process `System` (always has a static `process.pid` value equal to `4` and represents the Windows kernel code and loaded kernel mode drivers):

```
{
  "_index": ".ds-logs-endpoint.events.file-default-000004",
  "_type": "_doc",
  "_id": "kXYPpXcB0RzSN0EU6_TE",
  "_version": 1,
  "_score": null,
  "_source": {
    "agent": {
      "id": "b07fedf1-c13f-6a4a-7f74-0f3dbd655f0e",
      "type": "endpoint",
      "version": "7.10.0"
    },
    "process": {
      "Ext": {
        "ancestry": [
          "YjA3ZmVkZjEtYzEzZi02YTRhLTdmNzQtMGYzZGJkNjU1ZjB1LTAtMTMyNTc0NDcyMzYuNDkxNTM4OTAw"
        ]
      },
      "name": "System",
      "pid": 4,
      "entity_id": "YjA3ZmVkZjEtYzEzZi02YTRhLTdmNzQtMGYzZGJkNjU1ZjB1LTAtMTMyNTc0NDcyMzYuNDkxNTM4OTAw"
    },
    "message": "Endpoint file event",
    "@timestamp": "2021-02-15T09:35:57.042811Z",
    "file": {
      "Ext": {
        "windows": {
          "zone_identifier": -1
        }
      },
      "path": "C:\\Windows\\iphlpiper.dll",
      "extension": "dll",
      "name": "iphlpiper.dll"
    },
    "ecs": {
      "version": "1.5.0"
    },
    "data_stream": {
      "namespace": "default",
      "type": "logs",
      "dataset": "endpoint.events.file"
    }
  }
}
```

Figure 3: File creation event details depicted in Kibana's Discover view as a result of file transfer over SMB

A file creation event alone is not enough (the `System` process may create files that are related to local activity) to conclude that this activity pertains to a Lateral Movement attempt. Thus, we need to correlate it with *incoming* SMB network events by the same process:

```

sequence by host.id with maxspan=30s
[network where event.type == "start" and process.pid == 4 and destination.port == 445 and
network.direction == "incoming" and network.transport == "tcp" and
source.address != "127.0.0.1" and source.address != ":::1"
] by process.entity_id
[file where event.type in ("creation", "change") and process.pid == 4] by process.entity_id

```

Figure 4: Hunting EQL for file transfer via hidden file share from target host  
The above query looks for an incoming remote network event to tcp port 445 (SMB) followed by immediate file creation or modification (can be limited to executable file extension to reduce false positives) and both events are performed by the same (`process.entity_id`) virtual `System` process.

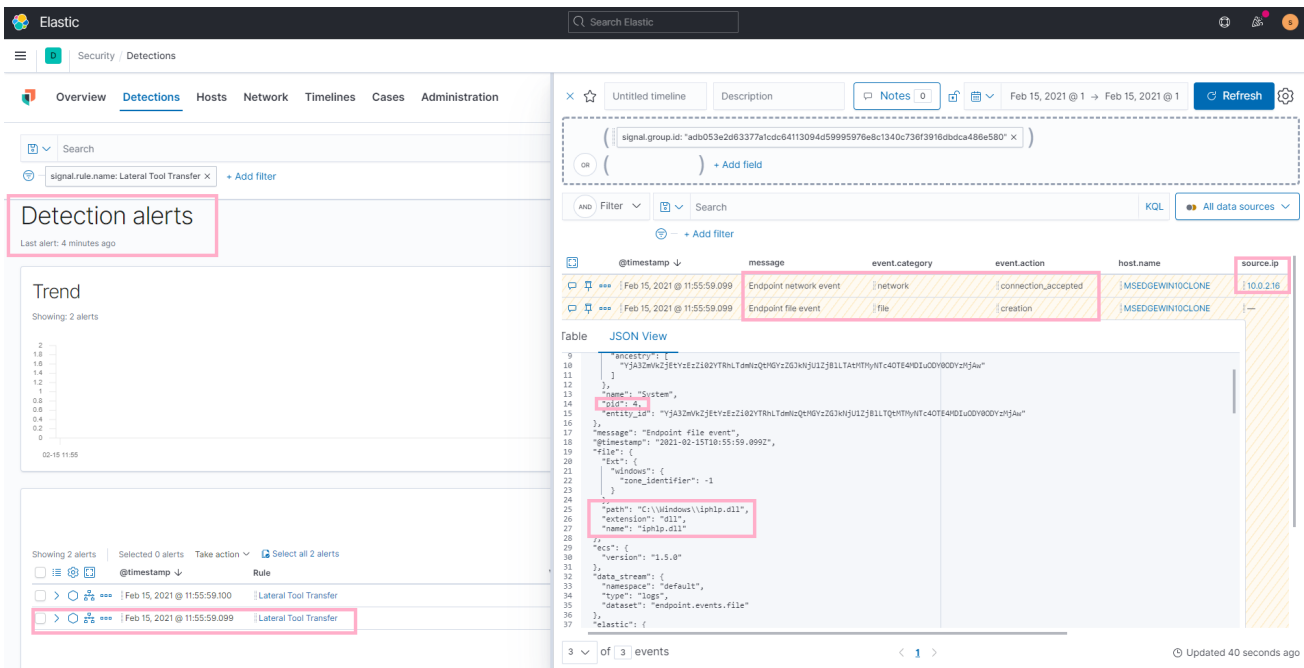


Figure 5: Detection alert example for Lateral Tool Transfer from target host  
The above alert contains details about the file that was copied as well as the `source.ip` address of the Lateral Movement activity. The same logic triggers on `PSEXEC`, a remote execution utility often abused by adversaries for the same purpose:

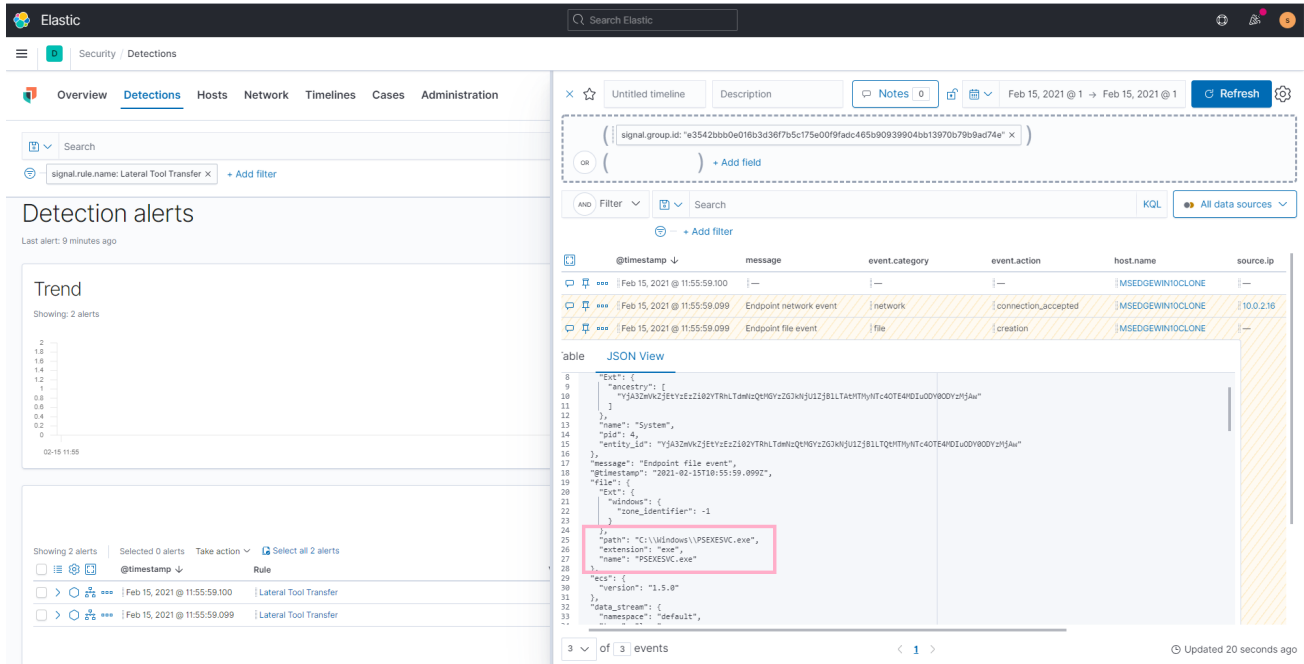


Figure 6: Lateral Tool Transfer triggering on PSEXEC from target host

We can also leverage EQL correlation to capture instances where a file that was copied via SMB is immediately executed:

```
sequence with maxspan=1m
[file where event.type in ("creation", "change") and
process.pid == 4 and file.extension : "exe"] by host.id, file.path
[process where event.type in ("start", "process_started")] by host.id, process.executable
```

Figure 7: Hunting EQL for remote execution via file shares

The above EQL looks for a sequence of events where a file is created/modified by the virtual **System** process followed by a process event where the process.executable is equal to the file.path. Below is an alert example:

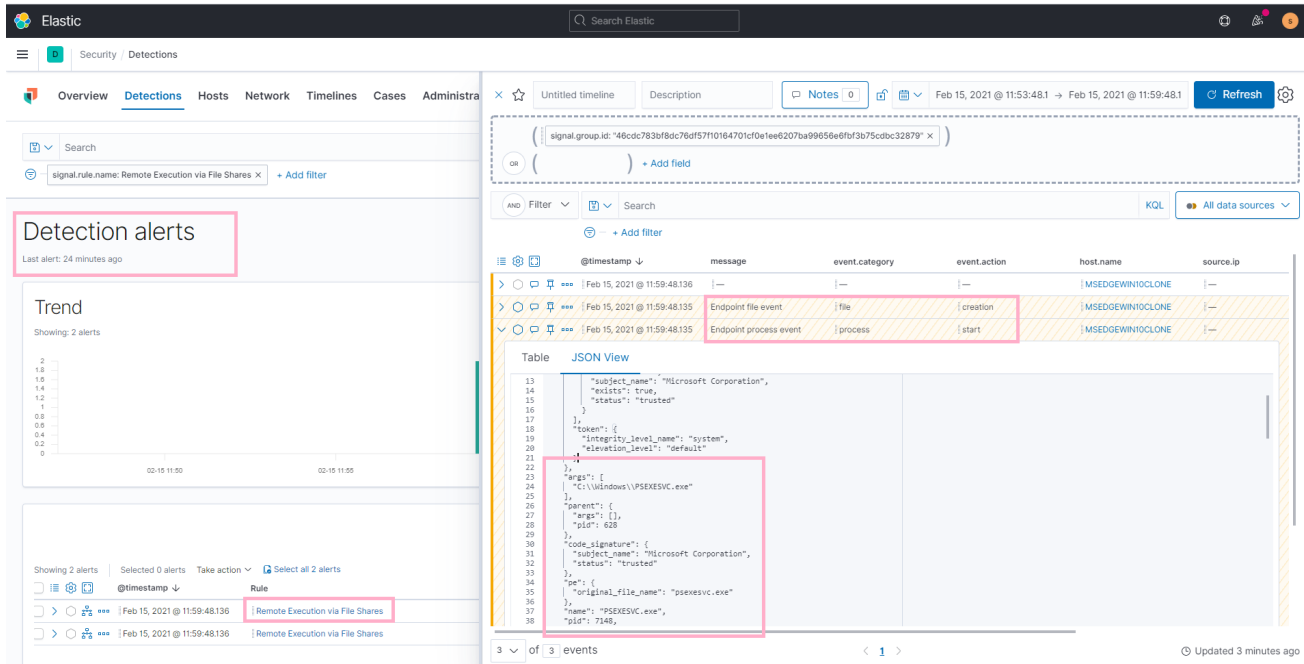


Figure 8: Detection alert for remote execution via file shares from target host  
 Another example where file transfer over SMB can be abused for remote execution is copying a malicious executable, script, or shortcut to the Startup folder of a target host. This will cause the program referenced to be automatically executed when a user logs in, and in the context of that user:

```

sequence by host.id with maxspan=30s
[network where event.type == "start" and process.pid == 4 and destination.port == 445 and
network.direction == "incoming" and network.transport == "tcp" and
source.address != "127.0.0.1" and source.address != ":::1"
] by process.entity_id
[file where event.type in ("creation", "change") and
process.pid == 4) and
file.path : "*\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\*" ] by process.entity_id

```

Figure 9: Hunting EQL for Lateral Movement via startup folder  
 Below is an example of a detection alert for Lateral Movement via the Startup folder:





Both `schtasks.exe` and direct usage of a custom implementation will cause a process to load the Task Scheduler COM API ( `taskschd.dll` ), followed by an outbound network connection where both the `source.port` and the `destination.port` are equal or greater than RPC dynamic ports ( `49152 to 65535` ) and from the same `process.entity_id`, which can be translated to this EQL query:

```
sequence by host.id, process.entity_id with maxspan=30s
[library where dll.pe.original_file_name : "taskschd.dll"]
[network where network.direction == "outgoing" and destination.port >= 49152 and
source.port >= 49152 and destination.address not in ("127.0.0.1", ":::1")]
```

Figure 12: Hunting EQL query for outbound task scheduler activity on source host  
Of course, matches to this query can be related to scheduled tasks discovery as well.  
Below is an example of an alert where we can observe the username, source, and destination IP, as well as the process name used to perform a remote task activity:

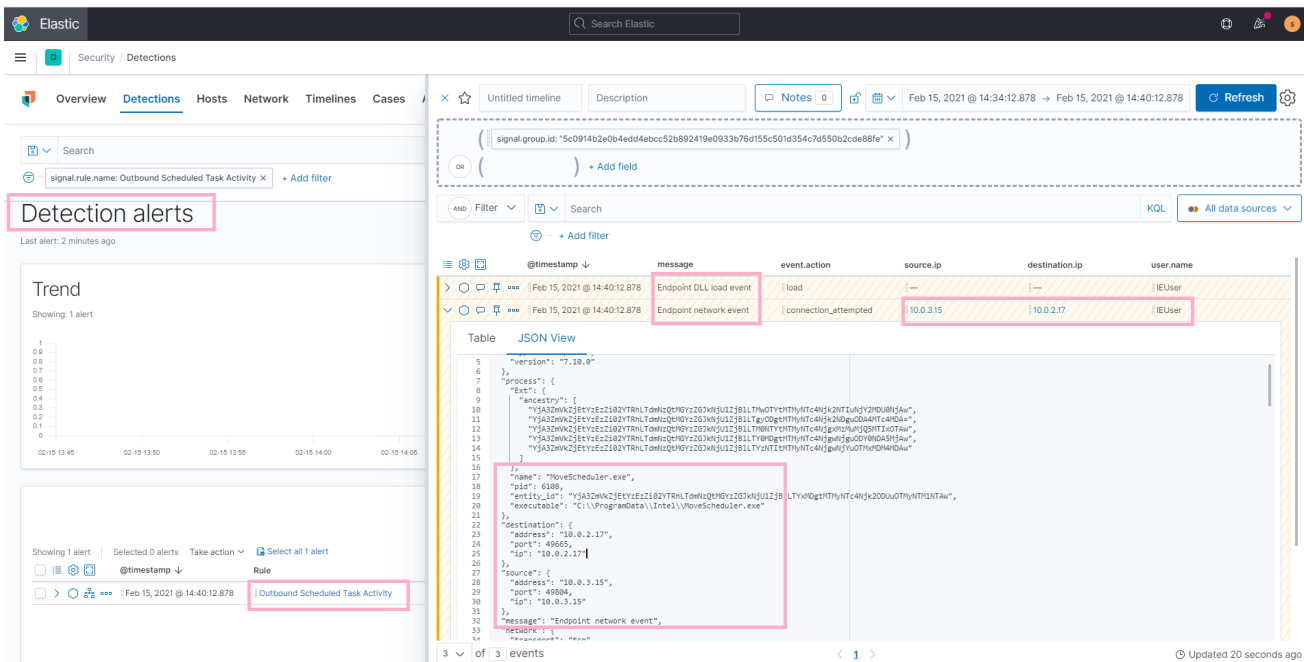


Figure 13: Detection alert for Lateral Movement via Scheduled Task on source host  
On the *target* host, we can hunt for remote scheduled task creation/modification via two options:

1. Incoming DCE/RPC (over TCP/IP) network event by the Task Scheduler service ( `svchost.exe` ) followed by a file creation of a task XML configuration file ( `C:\Windows\System32\Tasks\task_filename` )



2. Incoming DCE/RPC (over TCP/IP) network event by the Task Scheduler service ( `svchost.exe` ) followed by a registry change of a task cache Action value ( `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks\{GUID}\Actions` )

Option A provides us with the task name (equal to the file.name of the changed/created file), and Option B provides us with the task action itself (equal to the base64 decoded data of the Action registry value where the task scheduler service caches the task action configuration):

```
sequence by host.id, process.entity_id with maxspan = 1m
[network where process.name : "svchost.exe" and
network.direction == "incoming" and source.port >= 49152 and destination.port >= 49152 and
source.address != "127.0.0.1" and source.address != "::1"
]
[file where wildcard(file.path, "C:\\Windows\\System32\\Tasks\\*")]
```

Figure 14: Hunting EQL query for task creation on target host (Option A)

```
sequence by host.id, process.entity_id with maxspan = 1m
[network where process.name : "svchost.exe" and
network.direction == "incoming" and source.port >= 49152 and destination.port >= 49152 and
source.address != "127.0.0.1" and source.address != "::1"
]
[registry where registry.path : "HKLM\\SOFTWARE\\Microsoft\\Windows
NT\\CurrentVersion\\Schedule\\TaskCache\\Tasks\\*\\Actions"]
```

Figure 15: Hunting EQL query for task creation on target host (Option B)

Option B has the advantage of providing details about the task action, which tend to be useful while triaging (set to execute a program from a suspicious path, LOLBAS process, etc.).

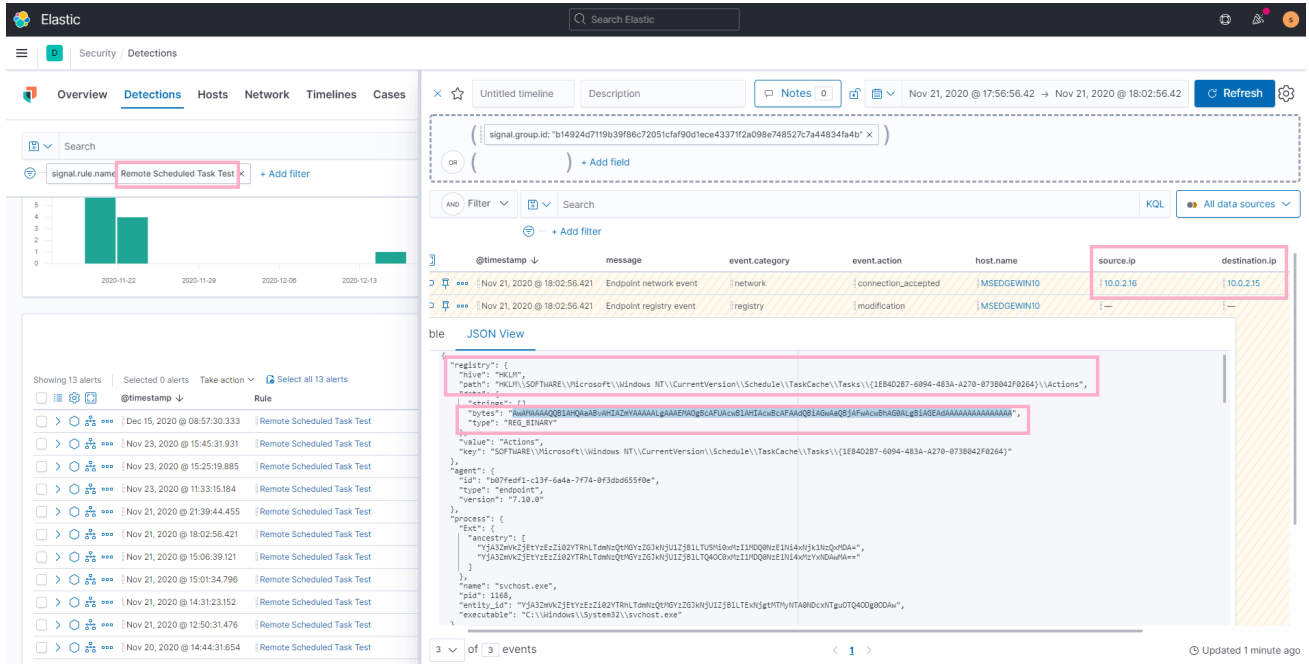


Figure 16: Detection alert for Lateral Movement via Scheduled Task on target host  
Decoding the registry Action base64 encoded data provides us details about the created task action:

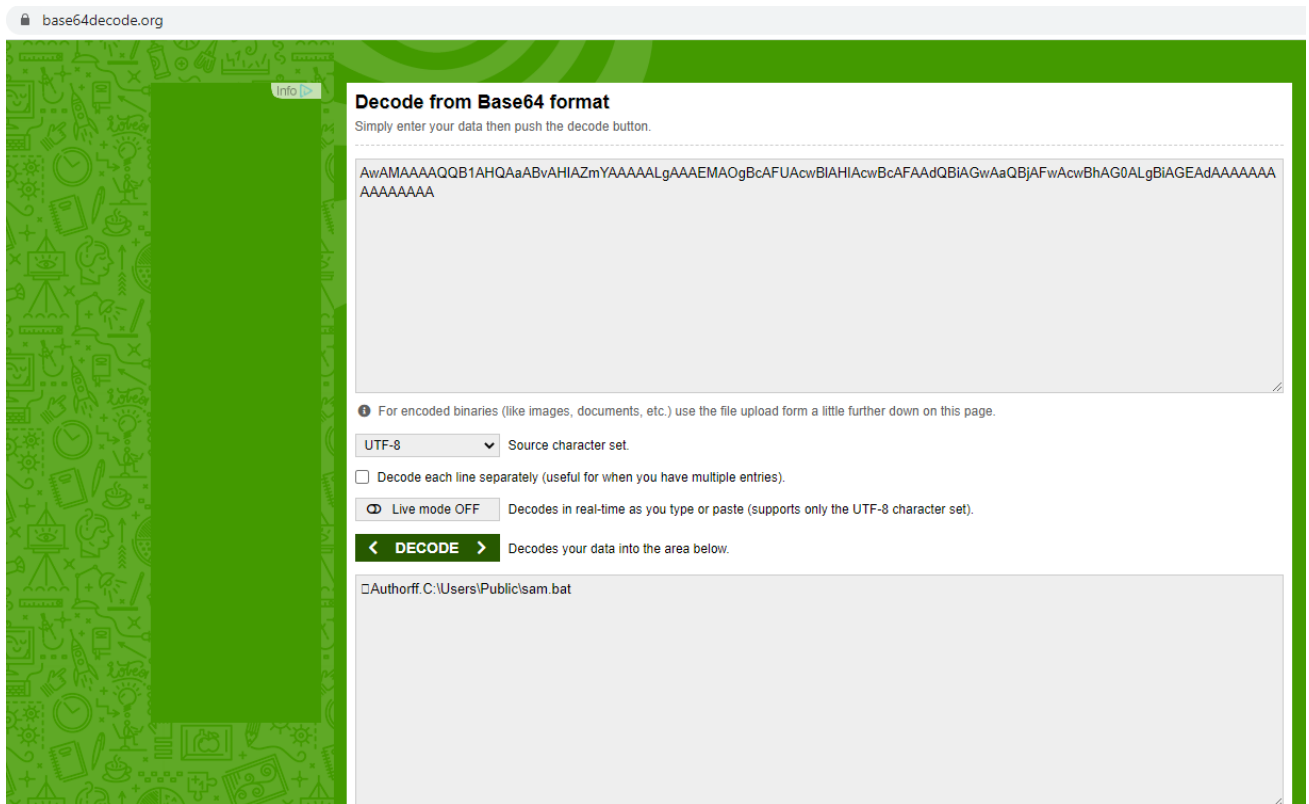
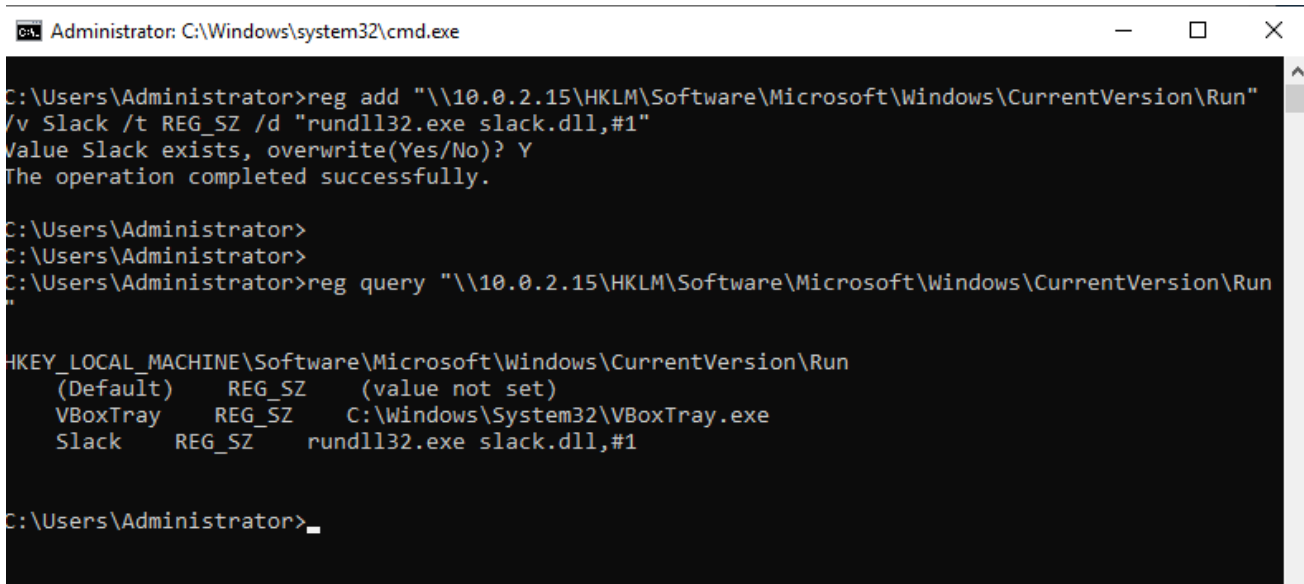


Figure 17: Base64 decoded data of the scheduled task action registry value

## Remote Registry

Adversaries may leverage the Remote Registry service for defense evasion or remote execution. One simple scenario is to modify the `Run` key registry on a remote system to cause the execution of a program upon system startup or user logon:



```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\Administrator>reg add "\\10.0.2.15\HKLM\Software\Microsoft\Windows\CurrentVersion\Run" /v Slack /t REG_SZ /d "rundll32.exe slack.dll,#1"
Value Slack exists, overwrite(Yes/No)? Y
The operation completed successfully.

C:\Users\Administrator>
C:\Users\Administrator>
C:\Users\Administrator>reg query "\\10.0.2.15\HKLM\Software\Microsoft\Windows\CurrentVersion\Run"
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
(Default) REG_SZ (value not set)
VBoxTray REG_SZ C:\Windows\System32\VBoxTray.exe
Slack REG_SZ rundll32.exe slack.dll,#1

C:\Users\Administrator>
```

Figure 18: Remote modification of the Run registry key via reg utility

We can hunt for this behavior from the source machine by looking for the execution of `reg.exe` with process.args containing `\\*`, but the same action can be achieved via API calls avoiding `process_command_line`-based detections.



```
{
  "args": [
    "reg",
    "add",
    "\\10.0.2.15\\HKKEY_USERS\\S-1-5-21-3461283602-4096304019-2269080069-1000\\Software\\Microsoft\\Windows\\CurrentVersion\\Run",
    "/v",
    "Slupdate",
    "/t",
    "REG_SZ",
    "/d",
    "C:\\Windows\\System32\\Rundll32.exe smssvc.dll,Start"
  ],
  "parent": {
    "args": [
      "cmd"
    ],
    "name": "cmd.exe",
    "pid": 8120,
    "args_count": 1,
    "entity_id": "YjA3ZmVkJEYzEzZ102YTRhLTdmNzQ0MGYzZGJkNjU1ZjB1TGxhMjAtMTMyNTE1OTQ0MjMwNjE5MjAw",
    "command_line": "cmd",
    "executable": "C:\\Windows\\System32\\cmd.exe"
  },
  "code_signature": {
    "trusted": true,
    "subject_name": "Microsoft Windows",
    "exists": true,
    "status": "trusted"
  },
  "pe": {
    "original_file_name": "reg.exe"
  },
  "name": "reg.exe",
  "pid": 8100,
  "args_count": 9,
  "entity_id": "YjA3ZmVkJEYzEzZ102YTRhLTdmNzQ0MGYzZGJkNjU1ZjB1TGxhMjAtMTMyNTE1OTQ0MjMwNjE5MjAw",
  "command_line": "reg add \\10.0.2.15\\HKKEY_USERS\\S-1-5-21-3461283602-4096304019-2269080069-1000\\Software\\Microsoft\\Windows\\CurrentVersion\\Run" /v Slupdate /t REG_SZ /d \\C:\\Windows\\System32\\Rundll32.exe smssvc.dll,Start",
  "executable": "C:\\Windows\\System32\\reg.exe",
  "hash": {
    "sha1": "429df8371b437209d79dc97978c33157d1a71c4b",
    "sha256": "19316d4266d0b776d9b2a05d5903d8cb0f0ea1520e9c2a7e6d5960b6fa4dcdf",
    "md5": "0a93acac33151793f8d52000071c0b06"
  }
}
```

Figure 19: Example of Reg.exe process execution event on source host

Note that `Reg.exe` is not performing any network connection — instead, it's the virtual `System` process that issues an outbound network connection to the target host on port 445 (DCE/RPC over SMB).

On the target host we can see the following sequence of key events:

1. Incoming network connection on tcp port 445 (DCE/RPC over SMB) by the virtual `System` process (`process.pid` equal 4)
2. `RemoteRegistry` service process starts ( `svchost.exe` with `process.args` containing the string `RemoteRegistry` )
3. `RemoteRegistry` service process performs the registry change

@timestamp	message	event.category	event.action	host.name	source.ip	destination.ip	user.name
Nov 21, 2020 @ 17:54:51.532	Endpoint network event	network	connection_accepted	MSEDGEWIN10	10.0.2.16	10.0.2.15	SYSTEM
Nov 21, 2020 @ 17:54:51.532	Endpoint process event	process	start	MSEDGEWIN10	—	—	LOCAL SERVICE
Nov 21, 2020 @ 17:54:51.532	Endpoint registry event	registry	modification	MSEDGEWIN10	—	—	LOCAL SERVICE

Figure 20: Remote Registry-relevant events on target host

The following EQL hunt can be used to correlate (2) and (3) by `host.id` and `process.entity_id` of the Remote Registry service:

```
sequence by host.id, process.entity_id with maxspan=30s
  [process where event.type == "start" and process.name == "svchost.exe" and
  process.args == "RemoteRegistry"]
  [registry where true]
```

Figure 21: Hunting EQL to detect Remote Registry modification via Regsvc on target host  
If we include (1) in the above sequence to capture the `source.ip` address, it may trigger on unrelated incoming SMB connections as the only common element between the three events limited to the `host.id` value.

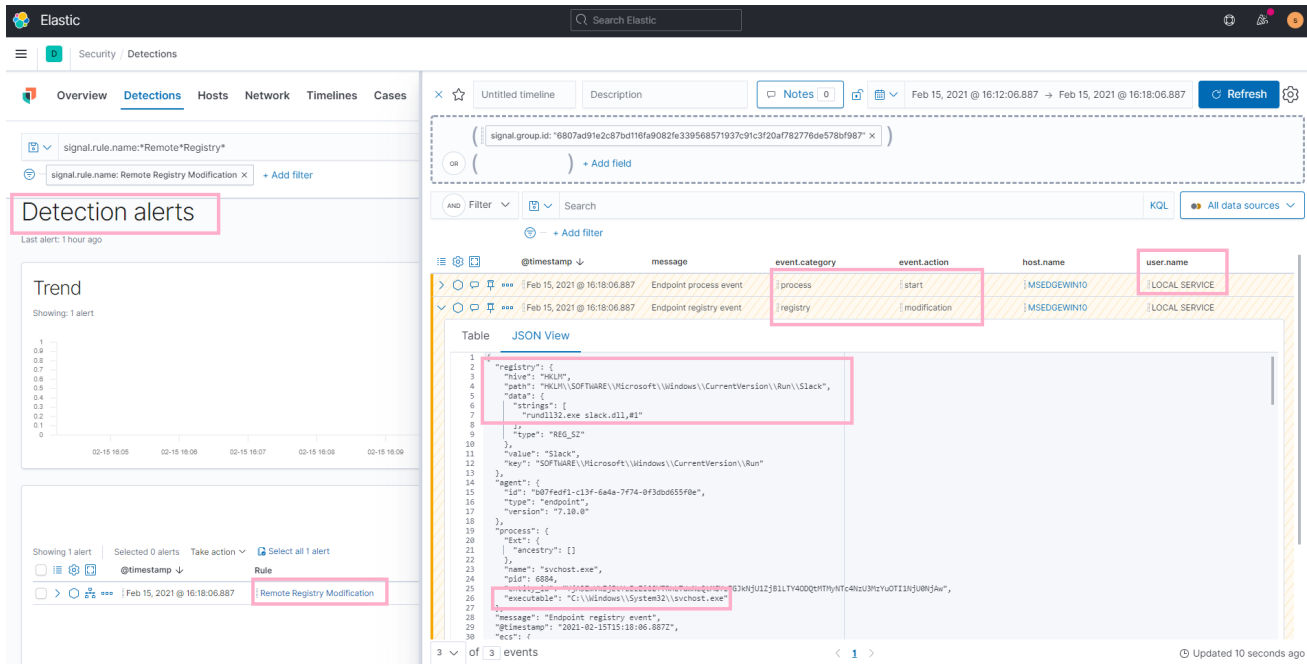


Figure 22: Detection alert for Remote Registry modification via Regsvcs on target host  
Adversaries may attempt to achieve the same outcome via the Windows Management Instrumentation (WMI) registry provider (StdReg), which behaves differently:

1. WMI Service ( `svchost.exe` with `process.args` containing `winmgmt` string) accepts an incoming DCE/RPC (over TCP/IP) network connection where both source.port and the destination.port are greater than or equal to RPC dynamic ports ( `49152 to 65535` )
2. A new instance of the WMI Provider Host (process.name equal to `WmiPrvSe.exe` with user.name equal to `Local Service` or user.id equal to `S-1-5-19` ) is started
3. The started WMI Provider Host loads the registry provider `StdProv.dll` module
4. The WMI Provider Host performs the registry change

We can express the correlation of (1), (2) and (4) with the following hunting EQL:

```

sequence by host.id with maxspan = 1s
[network where network.direction == "incoming" and event.action == "connection_accepted" and
source.port >= 49152 and destination.port >= 49152 and not source.address in ("127.0.0.1", ":::1")]
[process where event.type == "start" and process.name == "WmiPrvSE.exe" and user.name == "LOCAL SERVICE"]
[registry where process.name == "WmiPrvSE.exe" and user.name == "LOCAL SERVICE"]

```

Figure 23: Hunting EQL for Remote Registry modification via Regsvcs on target host  
If logging of the `StdProv.dll` module loading is enabled, we can also add (3) to the sequence to reduce potential false positives:

```

sequence by host.id with maxspan = 10s
[network where network.direction == "incoming" and
event.action == "connection_accepted" and source.port >= 49152 and destination.port >= 49152 and
not source.address in ("127.0.0.1", "::1")]
[process where event.type == "start" and process.name == "WmiPrvSE.exe" and
user.name == "LOCAL SERVICE"]
[library where dll.name == "stdprov.dll" and process.name == "WmiPrvSE.exe" and
user.name == "LOCAL SERVICE"]
[registry where process.name == "WmiPrvSE.exe" and user.name == "LOCAL SERVICE"]

```

Figure 24: Hunting EQL for Remote Registry modification via Regsvc on target host (library event)

Below an example of a detection alert where we can see the remotely modified registry details and the remote source.ip:

The screenshot shows the Elastic Security Detections interface. A detection alert is displayed with the following details:

- Signal Group ID:** 9ffa4bda334217afe85a37fb94829a850adb335f0930474b959182dc449ba0
- Alert Title:** Incoming Registry Modification Using WMI
- Event Table:**

Timestamp	Message	Event Category	Event Action	Host Name	Source IP	Destination IP
Dec 5, 2020 @ 22:45:59.823	Endpoint network event	network	connection_accepted	IMSEDEWIN10	10.0.2.17	10.0.2.15
Dec 5, 2020 @ 22:45:59.824	Endpoint process event	process	start	IMSEDEWIN10	-	-
Dec 5, 2020 @ 22:45:59.824	Endpoint registry event	registry	modification	IMSEDEWIN10	-	-
- JSON View Details:**

```

{
  "registry": {
    "hive": "HKLM",
    "path": "HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\wm54",
    "data": {
      "strings": [
        "powershell.exe"
      ],
      "type": "REG_SZ"
    },
    "value": "wm54",
    "key": "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"
  },
  "agent": {
    "id": "b07fedf1-c13f-6a4a-7f74-0f3dbd655f9e",
    "type": "endpoint",
    "version": "7.10.0"
  },
  "process": {
    "ext": {
      "ancestry": [ ]
    },
    "name": "WmiPrvSE.exe",
    "pid": 2268,
    "parent": "C:\\Windows\\System32\\wbem\\WmiPrvSE.exe",
    "executable": "C:\\Windows\\System32\\wbem\\WmiPrvSE.exe"
  }
}

```

Figure 25: Detection alert for Remote Registry modification via the WMI on target host

## Sharp Remote Desktop

SharpRDP is a Lateral Movement tool that leverages the Remote Desktop Protocol (RDP) for authenticated command execution and without the need for graphical interaction.

Once authenticated, SharpRDP sends virtual keystrokes to the remote system via a method called SendKeys to open up a Run dialog on the target host and then enter a specified command, which will be executed on the target host.

The main indicator from the source host is an unusual process (hosting SharpRDP code) loading the Remote Desktop Services ActiveX Client that implements RDP client functionality ( MSTscAx.dll ) followed by an outbound network connection to RDP tcp port



3389 and both events from the same `process.entity_id`:

```
sequence by host.id, process.entity_id with maxspan=30s
[library where dll.pe.original_file_name : "mstscax.dll" and
not process.executable : ("C:\\Windows\\System32\\mstsc.exe", "C:\\Windows\\SysWOW64\\mstsc.exe" )]
[network where network.direction == "outgoing" and destination.port == 3389 and
source.port >= 49152 ]
```

Figure 26: Hunting EQL for suspicious RDP Client

Below an example of results matching our hunting EQL where we can see an unusual process (other than `mstsc.exe` and similar known RDP clients) loading the Remote Desktop Services ActiveX Client ( `MSTscAx.dll` ) as well as the outbound network connection:

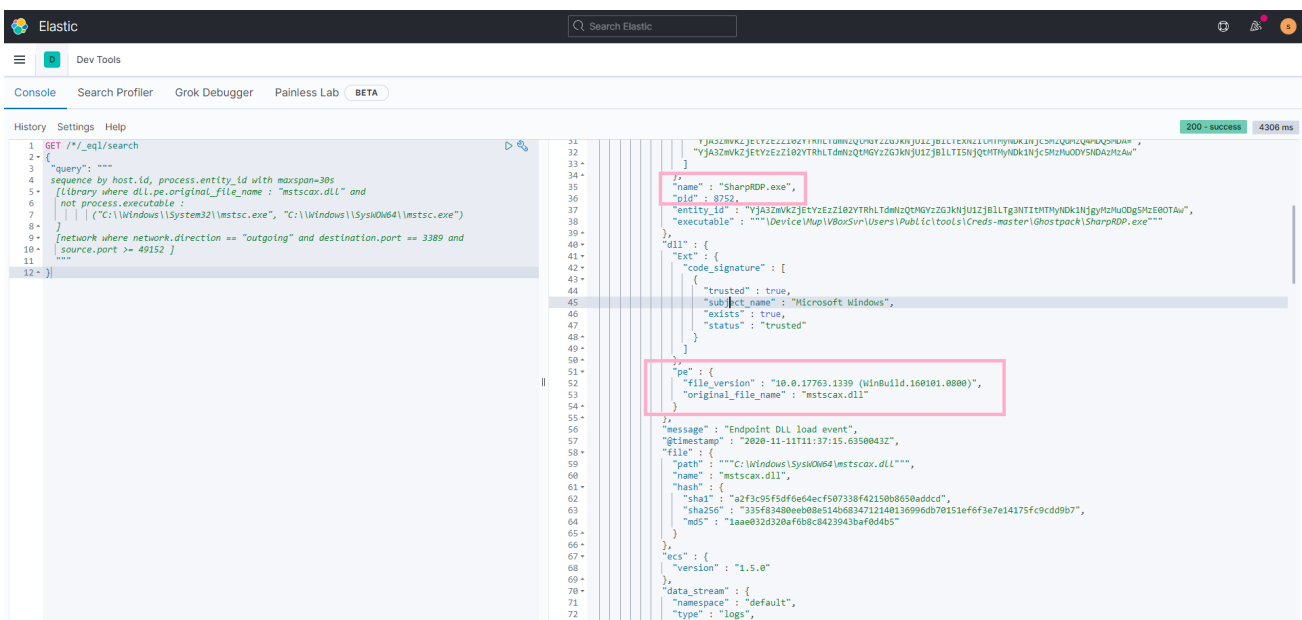


Figure 27: Results example for suspicious RDP Client EQL hunt

On the target host, the following key events occur within a one-minute time window:

1. An incoming network connection is accepted by the RDP service ( `TermService svchost.exe` ) on port `3389`
2. Under the `RunMRU` registry key, a new (or update to an existing) string value is set to `cmd` , `powershell` , `taskmgr` or `tsclient` (depending on the chosen SharpRDP `execution method`), which is caused by the typed command in the `Run dialog` via the `SendKeys` method
3. Depending on the `execution method`, a new process (attacker command) is created with `process.parent.name` of `cmd.exe` , `powershell.exe` , `taskmgr.exe` , or a random executable running from the `tsclient` mountpoint (shared drive from the RDP client host with the RDP target server)

For (2), note that when running anything from the Run dialog, a registry entry will be created at `HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU` showing what was entered into the Run dialog box.

The above sequence of events can be expressed with the following EQL:

```

/* Incoming RDP followed by a new RunMRU string value set to cmd, powershell, taskmgr or tsclient,
followed by process execution within 1m */

sequence by host.id with maxspan=1m
[network where event.type == "start" and process.name : "svchost.exe" and destination.port == 3389
and
network.direction == "incoming" and network.transport == "tcp" and
source.address != "127.0.0.1" and source.address != "::1"
]
[registry where process.name : "explorer.exe" and
registry.path : ("HKEY_USERS\\*\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\RunMRU\\*")
and
registry.data.strings : ("cmd.exe*", "powershell.exe*", "taskmgr*", "\\tsclient\\*.exe\\*")
]

[process where event.type in ("start", "process_started") and
(process.parent.name : ("cmd.exe", "powershell.exe", "taskmgr.exe") or process.args :
("\\tsclient\\*.exe")) and
not process.name : "conhost.exe"
]

```

Figure 28: Hunting EQL for SharpRDP behavior on the target host  
 Example of a detection alert and its composing event details on the target host:

The screenshot shows the Elastic Security Detections interface. On the left, a list of alerts is displayed, with the rule 'Potential SharpRDP C' selected. The main panel shows the details of a detection alert, including the signal group ID and the event details. The event details are shown in a table with columns for @timestamp, message, event.category, event.action, host.name, source.ip, and destination.ip. The event.action is 'connection\_accepted'. Below the table, the JSON view of the event is displayed, showing the event's structure and details.

@timestamp	message	event.category	event.action	host.name	source.ip	destination.ip
Nov 13, 2020 @ 12:45:56.413	Endpoint network event	network	connection_accepted	MSEDGEWIN0	10.0.2.16	10.0.2.15

```

{
  "name": "svchost.exe",
  "pid": 3389,
  "parent_pid": "10.0.2.15",
  "parent_name": "svchost.exe",
  "parent_exe_path": "C:\\Windows\\System32\\svchost.exe",
  "destination": {
    "address": "10.0.2.15",
    "port": 3389,
    "ip": "10.0.2.15"
  },
  "source": {
    "address": "10.0.2.16",
    "port": 52727,
    "ip": "10.0.2.16"
  },
  "message": "Endpoint network event",
  "network": {
    "transport": "tcp",
    "type": "ipsec",
    "direction": "incoming"
  },
  "@timestamp": "2020-11-13T11:43:56.413Z",
  "@ecs": {}
}

```

Figure 29: Detection alert for SharpRDP on target host (TermService network connection)

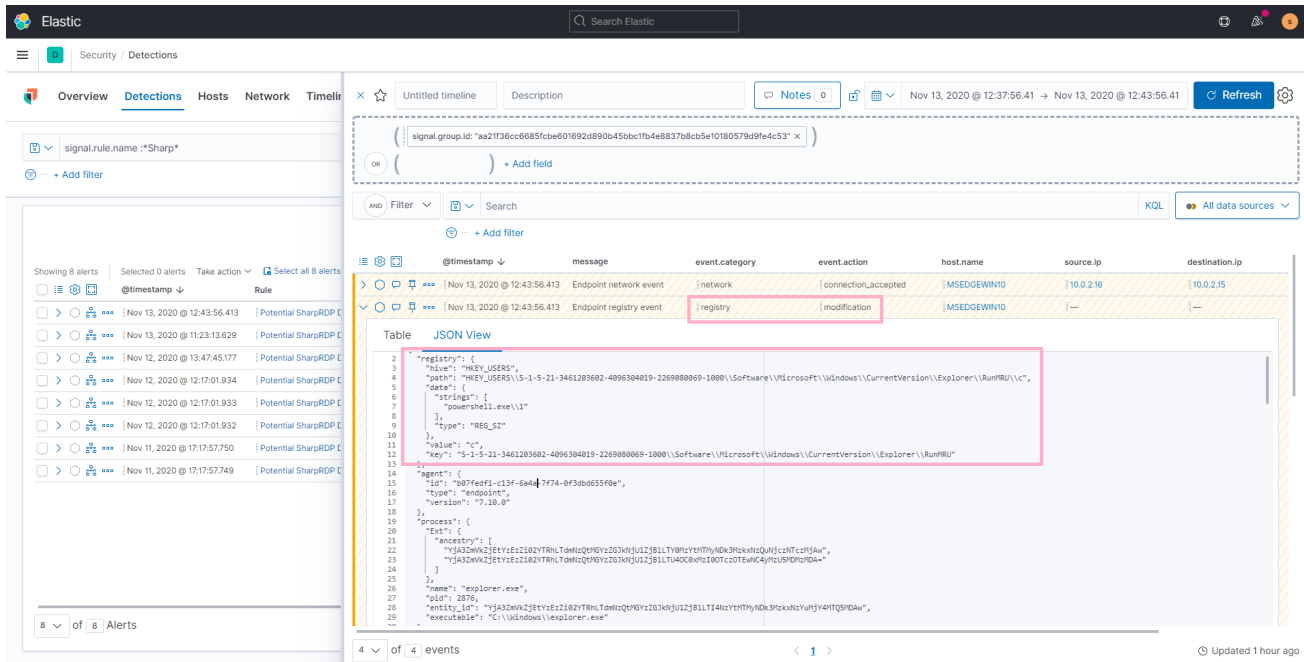


Figure 30: Detection alert for SharpRDP on target host (RunMRU set to Powershell)

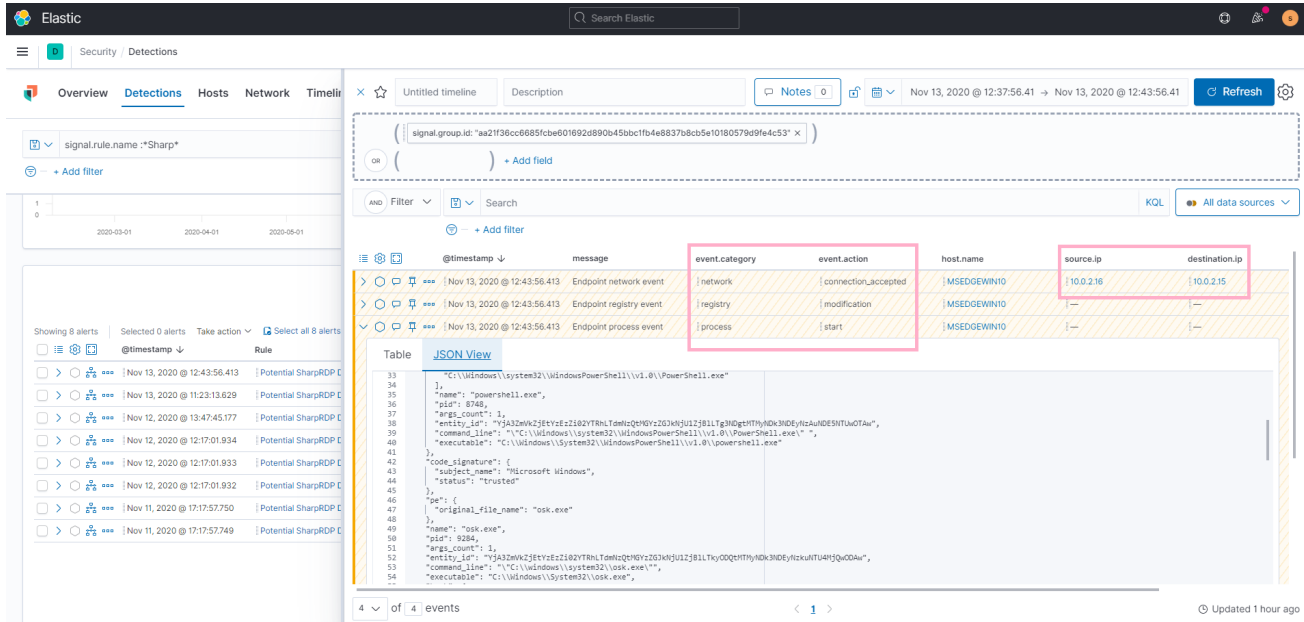


Figure 31: Detection alert for SharpRDP on target host (PowerShell child process)

## Wrapping up

Event Query Language (EQL) correlation capabilities enable us to capture complex behavior for a variety of Lateral Movement techniques. The high-level steps are:

1. **Understand the theory** and the building blocks of a certain technique (network protocols, loaded modules, services, process names, and arguments)
2. **Identify the key events** and their order that compose a certain behavior (both source and target host)

3. **Identify the common values** that can be used for correlation ([sequences](#)) — identifying more commonalities can reduce false positives
4. **Identify enrichment possibilities**, such as extra events in the sequence that can be useful during alert triage
5. **Assess the window of time** for correlation — using a shorter time window (for example, 30 seconds instead of 1 second) can reduce false positives, but can also introduce false negatives caused by network latency or slow system
6. **Test using different methods and tools** and tune the hunting logic accordingly, or, in some instances, duplicate logic to capture edge cases

Some of the [EQL](#) detection rules used as examples can be found in the [Elastic detection-rules repository](#):

If you're new to [Elastic Security](#), you can experience our latest version on [Elasticsearch Service](#) on Elastic Cloud.

### **We're hiring**

Work for a global, distributed team where finding someone like you is just a Zoom meeting away. Flexible work with impact? Development opportunities from the start?