HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
CYBER SECURITY K67

DEVELOPING WEB APPLICATION FOR SYSMON CONFIGURATION

PROJECT 2 REPORT

MEMBER LIST

Pham Hong Quan 20225551 Quan.PH225551@sis.hust.edu.vn

Supervisor: Nguyen Linh Giang, Assoc. Prof.

Hanoi — 2025

CONTENTS

1 ABSTRACT

System Monitor (Sysmon), a powerful tool from Microsoft's Sysinternals suite, enhances Windows system monitoring by logging detailed events critical for threat detection and incident response. However, its effectiveness relies heavily on well-crafted XML configuration files, which can be complex to author due to intricate rule ordering and filtering logic. This project develops a web-based application to simplify the creation, editing, and validation of Sysmon configurations, addressing challenges such as logical operator inconsistencies and rule precedence. Implemented using Node.js, HTML, CSS, and JavaScript, the application features an intuitive interface for importing, modifying, and exporting configurations in XML and TXT formats, with dynamic rule generation and drag-and-drop reordering. To validate its effectiveness, the project employs the Sliver C2 framework in a controlled environment to simulate adversarial behaviors, testing the generated configurations' ability to detect Indicators of Compromise (IoCs) such as process creations, network connections, registry modifications, and file operations. The results demonstrate the application's success in producing valid configurations that accurately capture Sliver IoCs while minimizing false positives. This work underscores the importance of accessible tools for Sysmon configuration and robust testing methodologies, contributing to enhanced endpoint monitoring and threat detection in cybersecurity.

2 PROJECT INTRODUCTION

In the evolving landscape of cybersecurity, effective system monitoring is paramount for detecting and responding to sophisticated threats. System Monitor (Sysmon), part of Microsoft's Sysinternals suite, provides granular logging of Windows system activities, including process creations, network connections, and registry changes, making it a cornerstone for threat hunting and forensic analysis. However, Sysmon's power is contingent on its configuration, defined through XML files that specify which events to log and how to filter them. Crafting these configurations is a complex task, requiring precise rule ordering, logical operator management, and alignment with organizational security needs. Errors in configuration can lead to missed detections or excessive noise, undermining security efforts. To address this challenge, this project develops a web-based application designed to streamline the authoring of Sysmon configuration files, making the process accessible to users with varying levels of expertise.

The application, built using Node.js for the backend and HTML, CSS, and JavaScript for the frontend, offers a user-friendly interface with features such as importing existing configurations, editing rules through dynamic forms, and exporting valid XML files. It incorporates best practices for Sysmon configuration, such as modular rule grouping and clear documentation, to ensure accuracy and maintainability. To evaluate the application's effectiveness, the project leverages the Sliver C2 framework, an open-source adversary emulation tool, to simulate real-world attack scenarios in a controlled environment. By generating and executing a Sliver implant on a Windows 10 virtual machine, the project tests the configurations' ability to detect key IoCs, including process executions, network connections, registry modifications, and file creations. This approach not only validates the application's utility but also demonstrates Sysmon's role in identifying adversarial tactics hich can be mapped to cyber adversary tactics, techniques, and procedures such as MITRE ATT&CK techniques.

The project's objectives are twofold: first, to reduce the complexity of Sysmon configuration authoring through an intuitive tool, and second, to validate the generated configurations against realistic threat scenarios. By combining software development with practical adversary emulation, the project contributes to the cybersecurity community's efforts to enhance endpoint visibility and threat detection. This report details the project's methodology, implementation, testing results, and insights, highlighting the significance of accessible configuration tools and robust testing frameworks in countering modern cyber threats.

3 Basic Information

3.1 Sysmon - A powerful Sysinernals tool

System Monitor (Sysmon) is a device driver that is part of the Sysinternals suite of tools [1]. Once installed on a Windows system, it persists across system reboots and continuously monitors and logs system activity to the Windows Event Log. Sysmon captures granular details about a wide range of event types, including but not limited to process creation, network connections, and file modifications. The events it generates can be collected via Windows Event Collection (WEC) or Security Information and Event Management (SIEM) agents, enabling detailed forensic analysis and threat detection. This capability facilitates the identification of malicious or anomalous behavior and provides insight into the techniques employed by intruders and malware within a network.

Sysmon includes the following capabilities: [1]

- Logs process creation events for both current and parent processes.
- Records hashes of process image files using SHA1, MD5, SHA256, or IMPHASH, with support for multiple hashes simultaneously.
- Includes a process GUID in process creation events to enable correlation even when process IDs are reused by the system.
- Embeds a session GUID in each event to correlate activities within the same logon session.
- Logs the loading of drivers and DLLs, including digital signatures and file hashes.
- Captures events where raw read access is performed on disks and volumes.
- Optionally logs network connections, detailing the source process, IP addresses, port numbers, hostnames, and port names.
- Detects and logs changes in file creation timestamps.
- Automatically reloads configuration upon changes in the system registry.
- Supports dynamic rule-based filtering to include or exclude specific events.
- Generates event data from early in the boot process, enabling detection of advanced kernel-mode threats.

And many more depends on the type of events being logged. Overall, Sysmon is a powerful system monitoring tool that enhances visibility into the behavior of a Windows system. It provides detailed logging of various system activities, such as process execution, network connections, file access, and driver loading. By generating rich and structured event data, Sysmon enables organizations to detect suspicious or anomalous behavior, investigate security incidents, and understand how threats operate within their environments. Its ability to persist across reboots, automatically adapt to configuration changes, and log activity from early in the boot process makes it especially valuable for both real-time monitoring and post-incident forensic analysis. For more information about each event types logged, please visit `https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon` or `https://sec.vnpt.vn/2024/12/sysmon-cong-cu-ho-tro-giam-sat-he-thong-mien-phi-va-hieu-qua/`

3.2 Sysmon configuration file

Configuration files in Sysmon are XML files used to customize how Sysmon logs events and system information. These files are utilized in conjunction with the **-i** (during installation), **-c** (for configuration updates) command-line options or **-s** (check upon the current Sysmon version & configuration), allowing users to easily deploy pre-defined configurations and filter the specific events to be monitored. Prior to authoring a configuration file, it is essential to carefully verify both the version and existing configuration of Sysmon to ensure compatibility and proper functionality.

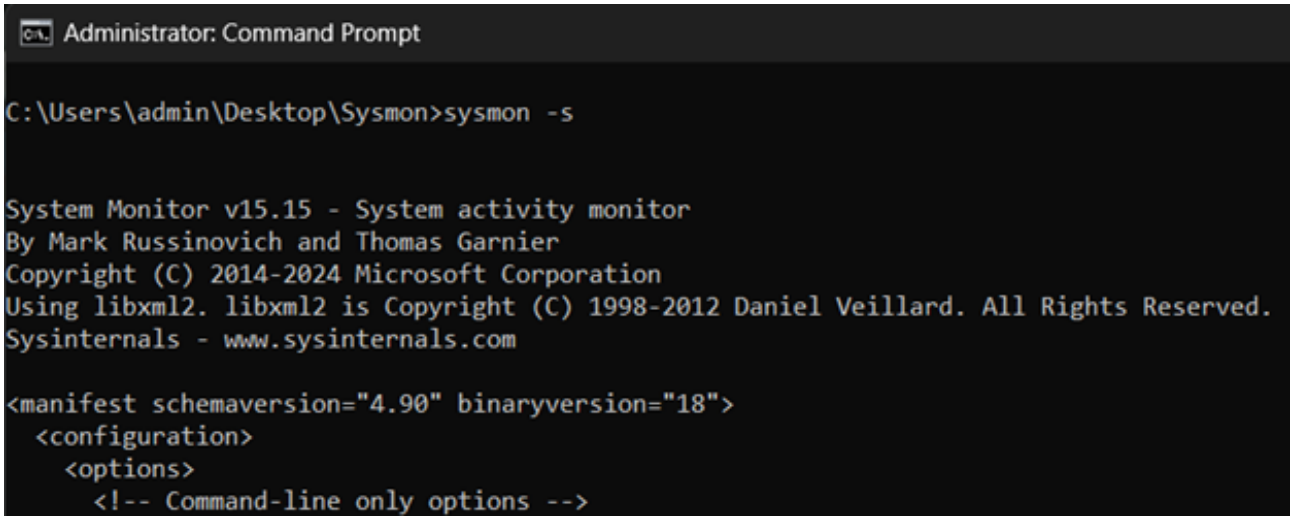

Figure 1: Getting sysmon version

Sysmon configuration files are composed of two principal components: **General Configuration** and **Event Filtering**. These components are defined within the root `<Sysmon>` XML element, which also includes a schemaversion attribute specifying the schema version used.

3.2.1 General Configuration

The general configuration section, located directly under the `<Sysmon>` tag, defines global parameters that govern Sysmon's behavior. These parameters include hashing algorithms, file archiving behavior, DNS lookup options, and more. Some of the key fields are:

- **HashAlgorithms**: Specifies the hashing algorithms used by Sysmon for image hashes. Supported values include `MD5`, `SHA1`, `SHA256`, `IMPHASH`, or `*` (to use all).
- **ArchiveDirectory**: Defines the directory where deleted executable files are archived using the Copy-On-Delete mechanism.
- **CheckRevocation**: A boolean flag that enables or disables the checking of certificate revocation status. Default is `True`.
- **CopyOnDeletePE**: When set to `True`, Sysmon saves copies of deleted executable files for further analysis. Default is `False`.
- **DnsLookup**: Enables or disables reverse DNS lookup for connections. Default is `True`.
- **FieldSizes**: Allows specifying the maximum field size for certain attributes. For example: `CommandLine:100`, `Image:50`.

A sample snippet for this section might look as follows:

```
<Sysmon schemaversion="4.90">
    <HashAlgorithms>SHA256</HashAlgorithms>
    <ArchiveDirectory>C:\SysmonArchive</ArchiveDirectory>
    <CheckRevocation>True</CheckRevocation>
</Sysmon>
```

Listing 1: Sample General Configuration in Sysmon

3.2.2 Event Filtering

Event filtering is encapsulated within the `<EventFiltering>` tag and defines specific rules for logging events based on Sysmon Event IDs. Each rule targets a particular event type (e.g., process creation, network connection, image load) and employs filter operators to determine which events are logged or excluded. Sysmon supports a wide range of filtering operators such as followed: [2]

Operator	Description	Example
`is`	Matches if the value exactly equals the specified string.	`<Image condition="is"> C:\Windows\System32\ipconfig.exe </Image>`
`is not`	Matches if the value does not exactly match the string.	`<Image condition="is not">powershell.exe</Image>`
`is any`	Matches if the value equals any of the specified options in the list.	`<GrantedAccess condition="is any">0x1234; 0x143a; 0x1dffff</GrantedAccess>`
`contains`	Matches if the value contains a specific substring.	`<CommandLine condition="contains"> http </CommandLine>`
`contains any`	Matches if the value contains any substring in the list.	`<Image condition="contains any">Appdata;\Users</Image>`
`contains all`	Matches only if the value contains all specified substrings.	`<Image condition="contains all">Appdata;\Users</Image>`
`excludes`	Matches if the value does not contain a specific substring.	`<CommandLine condition="excludes">svchost.exe -k</CommandLine>`
`excludes any`	Matches if the value does not contain at least one of the substrings in the list.	`<Image condition="excludes any">Dropbox.exe; winlogbeat.exe</Image>`

Operator	Description	Example
excludes all	Matches only if the value does not contain all the substrings.	<ImageLoaded condition="excludes all">C:\Users; \AppData\Local</ImageLoaded>
begins with, not begins with	Matches if the value starts (or does not start) with a specified prefix.	<Image condition="begin with">C:\Users</Image>
ends with, not ends with	Matches if the value ends (or does not end) with a specified suffix.	<Image condition="end with">redist.exe</Image>
image	Matches based on the image file name only.	<Image condition="image"> rundll32.exe </Image>
LessThan, MoreThan	Matches if a numeric or time value is less than or greater than a given threshold.	<UtcTime condition="MoreThan"> 2025-04-17T00:00:00 </UtcTime> <ProcessTerminate>

3.2.3 File configuration logic & ordering

Sysmon exhibits inconsistencies in how it processes logical **AND** and **OR** operations when multiple filter conditions are applied. Specifically, when multiple filters are defined for the same field, Sysmon applies **OR** logic—meaning that an event is matched if at least one condition is true. Conversely, when filters span across different fields, Sysmon enforces AND logic—requiring all conditions to be satisfied simultaneously.

Additionally, operators such as **contains all** and **excludes all** require Sysmon to evaluate all provided conditions, which may cause confusion when combined with other filters. To address this and ensure consistency, it is recommended to use the `<RuleGroup>` construct for each `EventType`. This approach helps isolate filtering logic per event type and guarantees that the evaluation of conditions is handled appropriately.

This distinction is particularly important when writing Sysmon configuration files, as improper use of logical grouping may result in either missed events or inaccurate logging of irrelevant activity. An example is as followed:

```
<Sysmon schemaversion="4.90">
  <EventFiltering>
    <RuleGroup name="" groupRelation="or">
      <ProcessCreate onmatch="exclude">
        <Image condition="is">C:\Windows\System32\plasrv.exe</Image>
        <Image condition="is">C:\Windows\System32\wifitask.exe</Image>
      </ProcessCreate>
    </RuleGroup>
  </EventFiltering>
```

```
10 </Sysmon>
```

Listing 2: Sample Configuration logic in Sysmon

Special attention must be given to the use of **include** and **exclude** rules, particularly in edge cases. If a process does not match any rule under **include**, it will not be logged—even if it appears in the **exclude** list. This behavior helps prevent redundant or missing logs during data collection.

Moreover, the execution order of **include** and **exclude** rules is critical. If the **exclude** rule is placed after the **include** rule, it will only act on the subset of events already matched by the **include** condition, and vice versa. The following example demonstrates this behavior.

Initially, Sysmon is configured to log only processes named `powershell.exe` and `cmd.exe`. After this inclusion step, it checks whether any of the matched processes should be excluded. Since `notepad.exe` and `calc.exe` are not part of the include rules from the beginning, they are never logged, regardless of their presence in the exclude list. As a result, Sysmon will only record events related to `cmd.exe` and `powershell.exe`. Even if a suspicious instance of `notepad.exe` (e.g., executing a PowerShell script via a macro) is present, it will still be excluded correctly. This configuration is considered valid because exclude operates only on the set of events already selected by include.

config_order.xml

```
<Sysmon schemaversion="4.90">
    <EventFiltering>
        <!-- Bước 1: Bao gồm tất cả các tiến trình PowerShell và CMD -->
        <RuleGroup name="Include PowerShell and CMD" groupRelation="or">
            <ProcessCreate onmatch="include">
                <Image condition="contains">powershell.exe</Image>
                <Image condition="contains">cmd.exe</Image>
            </ProcessCreate>
        </RuleGroup>

        <!-- Bước 2: Loại trừ những tiến trình hợp lệ (notepad, calc) -->
        <RuleGroup name="Exclude Common Applications" groupRelation="or">
            <ProcessCreate onmatch="exclude">
                <Image condition="is">C:\Windows\System32\notepad.exe</Image>
                <Image condition="is">C:\Windows\System32\calc.exe</Image>
            </ProcessCreate>
        </RuleGroup>
    </EventFiltering>
</Sysmon>
```

Figure 2: Correct configuration file

However, if the execution order is reversed - e.g., if the exclude rule is evaluated before include rules - a malicious process can bypass detection. For instance, an attacker could rename a malicious payload as `calc.exe` and execute it using PowerShell. Because `calc.exe` was excluded before the inclusion logic was applied, the filter fails to log the corresponding event, allowing the malicious activity to proceed undetected.

config_order.xml

```xml
<Sysmon schemaversion="4.90">
    <EventFiltering>
        <!-- Bước 1: Loại trừ những tiến trình hợp lệ trước -->
        <RuleGroup name="Exclude Common Applications" groupRelation="or">
            <ProcessCreate onmatch="exclude">
                <Image condition="is">C:\Windows\System32\notepad.exe</Image>
                <Image condition="is">C:\Windows\System32\calc.exe</Image>
            </ProcessCreate>
        </RuleGroup>

        <!-- Bước 2: Bao gồm tất cả các tiến trình PowerShell và CMD -->
        <RuleGroup name="Include PowerShell and CMD" groupRelation="or">
            <ProcessCreate onmatch="include">
                <Image condition="contains">powershell.exe</Image>
                <Image condition="contains">cmd.exe</Image>
            </ProcessCreate>
        </RuleGroup>
    </EventFiltering>
</Sysmon>
```

Figure 3: Incorrect configuration file

As a result, `calc.exe` get to run without being logged by Sysmon

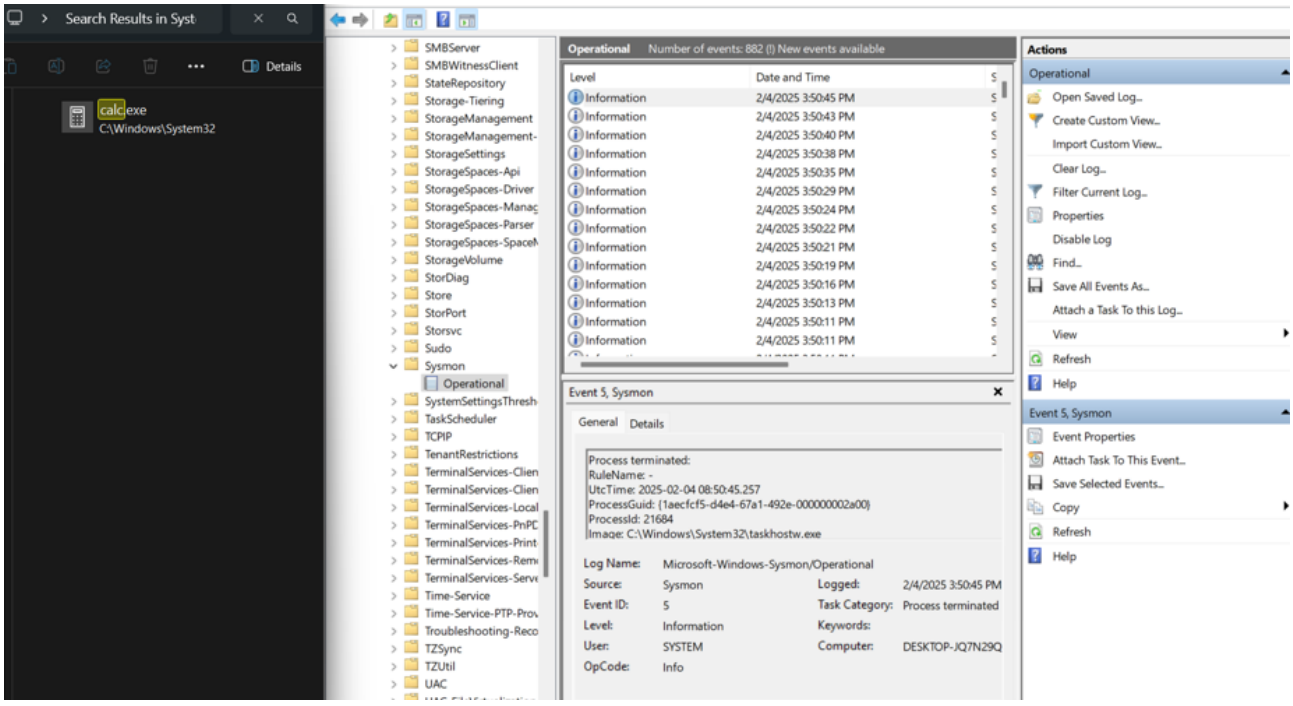

Figure 4: Sysmon failed to logged event

Also when authoring a Sysmon configuration file, it is not uncommon for users to encounter cases where certain events fail to be detected or logged. This issue may stem not only from incorrect configurations but also from the improper ordering of rules or filters within the file. Two key execution order principles must be considered:

1. **Order of Filter Evaluation**

The evaluation order of filters is governed by the internal schema of Sysmon, not by the sequence in which the filters are written in the configuration file. For example, even if the condition for `CommandLine` appears before the condition for `Image` in the XML, Sysmon may prioritize the `Image` condition during runtime based on the schema definition.

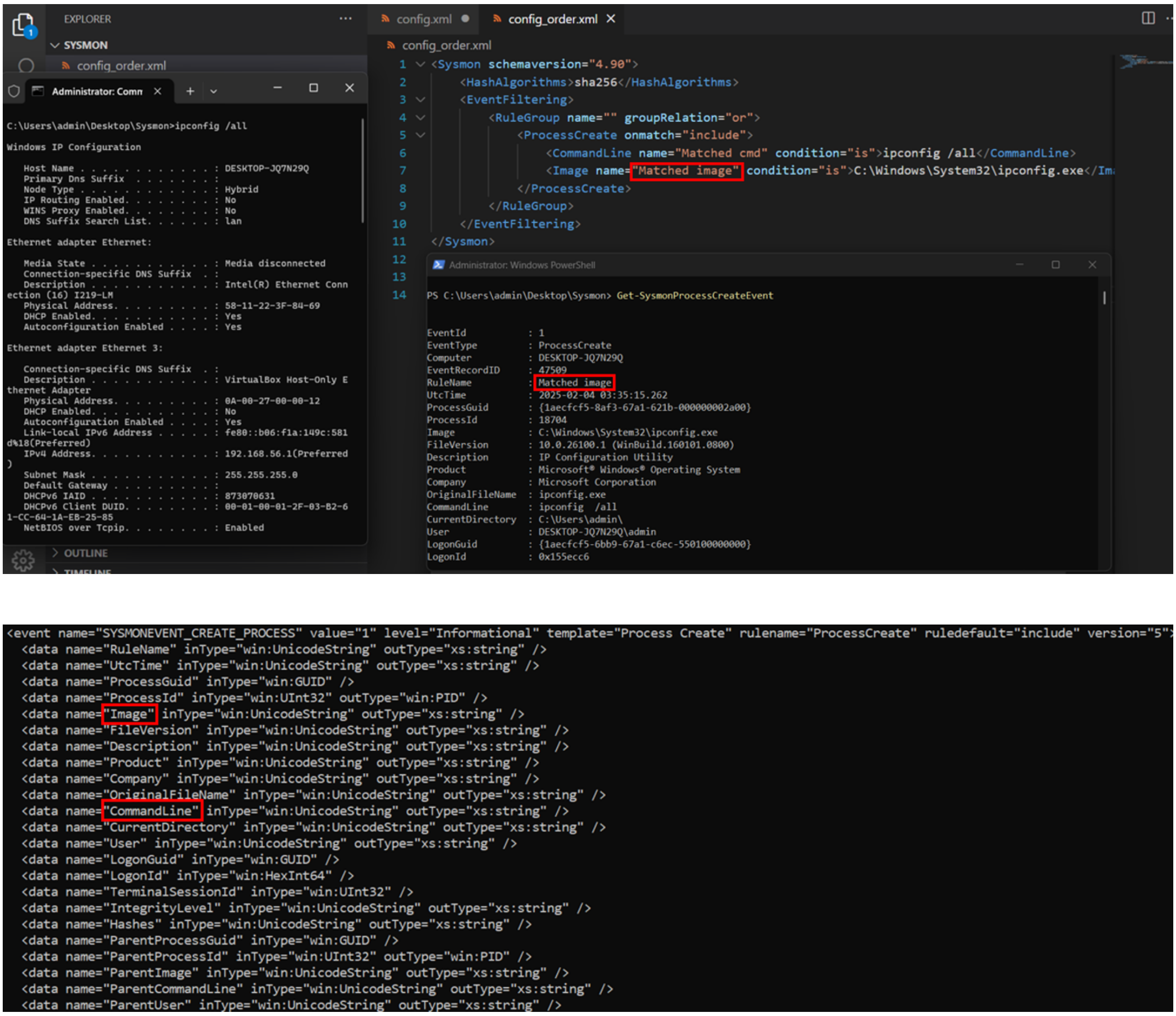

Figure 5: Order of executing base on Sysmon schema file

2. **Order of Rule Execution**
   In contrast, rule execution strictly follows the order in which the rules are declared in the configuration file. This means rules are processed sequentially from top to bottom. If an event satisfies multiple rules, the first matching rule encountered in the file will take precedence. In this example, although the contents of the two configuration files are identical, there is a notable difference in the ordering of elements. In the first configuration file, the `<Rule>` element appears before the two individual filters for `<CommandLine>` and `<Image>`. This difference in order once again leads to a variation in how Sysmon interprets and applies the rules to the corresponding event.

config_order.xml

```xml
<Sysmon schemaversion="4.90">
    <HashAlgorithms>sha256</HashAlgorithms>
    <EventFiltering>
        <RuleGroup name="IPConfig Detection" groupRelation="or">
            <ProcessCreate onmatch="include">
                <!-- Rule to detect high privilege IPConfig execution -->
                <Rule name="High Privilege IPConfig" groupRelation="and">
                    <OriginalFileName condition="is">ipconfig.exe</OriginalFileName>
                    <IntegrityLevel condition="is">High</IntegrityLevel>
                </Rule>
                <!-- Rule to detect specific IPConfig execution commands -->
                <CommandLine name="Matched CommandLine" condition="is">ipconfig /all</CommandLine>
                <Image name="Matched Image" condition="is">C:\Windows\System32\ipconfig.exe</Image>
            </ProcessCreate>
        </RuleGroup>
    </EventFiltering>
</Sysmon>
```

Select Administrator: Windows PowerShell

```
EventId           : 1
EventType         : ProcessCreate
Computer          : DESKTOP-JQ7N29Q
EventRecordID     : 51473
RuleName          : High Privilege IPConfig
UtcTime           : 2025-02-04 07:23:08.460
ProcessGuid       : {1aecfcf5-c05c-67a1-e42a-000000002a00}
ProcessId         : 17388
Image             : C:\Windows\System32\ipconfig.exe
FileVersion       : 10.0.26100.1 (WinBuild.160101.0800)
Description       : IP Configuration Utility
Product           : Microsoft® Windows® Operating System
Company           : Microsoft Corporation
OriginalFileName  : ipconfig.exe
CommandLine       : ipconfig  /all
CurrentDirectory  : C:\Users\admin\Desktop\Sysmon\
User              : DESKTOP-JQ7N29Q\admin
LogonGuid         : {1aecfcf5-6bb9-67a1-c6ec-550100000000}
LogonId           : 0x155ecc6
TerminalSessionId : 2
IntegrityLevel    : High
Hashes            : SHA256=9C552FA02A37BA6EA511A7A571B1D05671CE9C5589A6E180337ADD7BC35E3D0B
ParentProcessGuid : {1aecfcf5-bf6a-67a1-952a-000000002a00}
ParentProcessId   : 8472
ParentImage       : C:\Windows\System32\cmd.exe
ParentCommandLine : C:\WINDOWS\System32\cmd.exe
ParentUser        : DESKTOP-JQ7N29Q\admin
```

Figure 6: Configuration 1 - `<Rule>` before

config_order.xml

```xml
<Sysmon schemaversion="4.90">
    <HashAlgorithms>sha256</HashAlgorithms>
    <EventFiltering>
        <RuleGroup name="IPConfig Detection" groupRelation="or">
            <ProcessCreate onmatch="include">
                <!-- Rule to detect specific IPConfig execution commands -->
                <CommandLine name="Matched CommandLine" condition="is">ipconfig /all</CommandLine>
                <Image name="Matched Image" condition="is">C:\Windows\System32\ipconfig.exe</Image>
                <!-- Rule to detect high privilege IPConfig execution -->
                <Rule name="High Privilege IPConfig" groupRelation="and">
                    <OriginalFileName condition="is">ipconfig.exe</OriginalFileName>
                    <IntegrityLevel condition="is">High</IntegrityLevel>
                </Rule>
            </ProcessCreate>
        </RuleGroup>
    </EventFiltering>
</Sysmon>
```

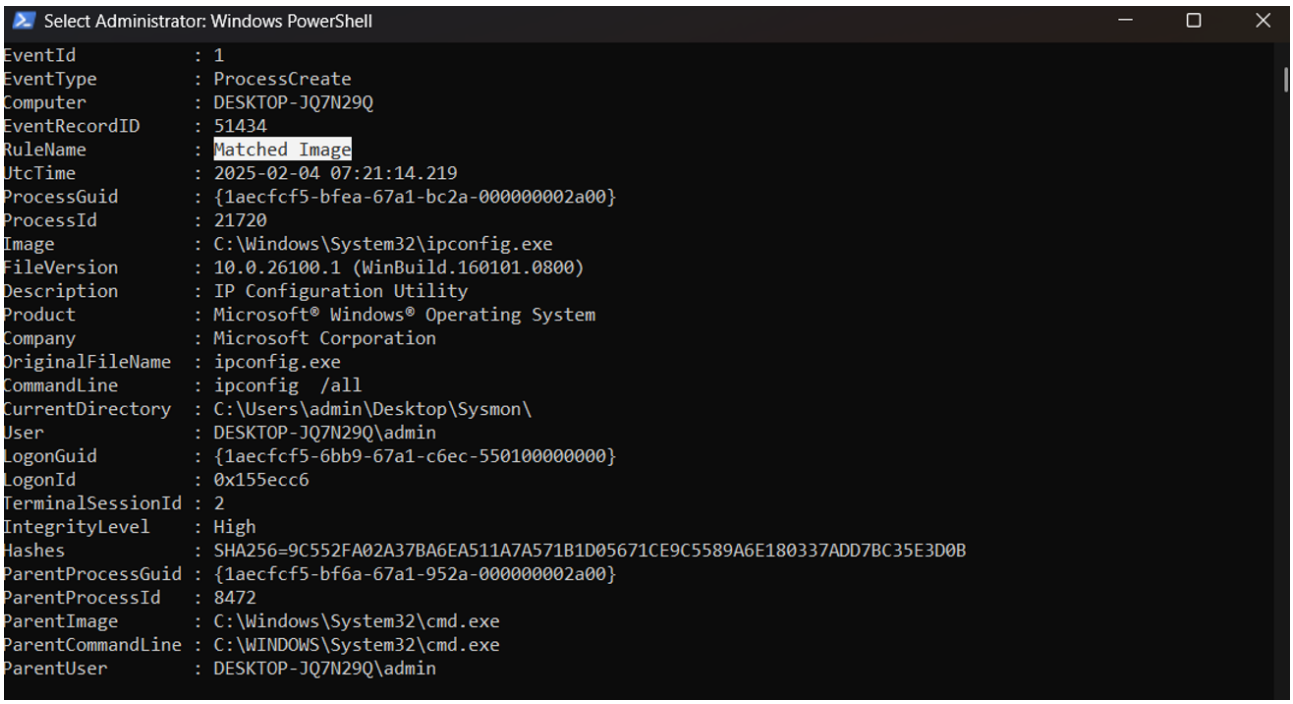

Figure 7: Configuration 2 - `<Rule>` after

In sum up, to correctly construct a Sysmon configuration file, the following principles should be adhered to:

1. **Define the File Structure**
   - All rules must be placed within the `<EventFiltering>` element to enable event filtering.
   - Use the `<RuleGroup>` element to group rules based on each specific event type (`EventType`).
   - Rules within a `<RuleGroup>` can be logically combined using the `groupRelation="or"` or `groupRelation="and"` attribute to control filter logic.
2. **Naming and Organization Conventions**
   - Assign meaningful and descriptive names to each `RuleGroup`, based on its monitoring purpose.
   - Clearly separate detection rules (e.g., `include` rules for identifying suspicious activity) from whitelisting rules (e.g., `exclude`).
   - Use XML comments (e.g., `<!- Comment ->`) to annotate and describe different parts of the configuration.
3. **Documentation and Notes**
   - State the purpose of each `RuleGroup` explicitly.
   - Provide brief explanations of why certain events are included or excluded.
   - Reference external documentation or test cases to justify and validate rule design.

3.3 Sliver - Adversary C2 emulation framework

Command and Control (C2) frameworks are essential tools used in offensive security to maintain communication between an attacker and compromised machines. These frameworks allow adversaries to remotely execute commands, upload payloads, exfiltrate data, and maintain persistence within a network. Among modern C2 platforms, Sliver stands out as an open-source, cross-platform, and stealthy red team framework developed by BishopFox [3]. Sliver supports multiple transport protocols (mTLS, HTTP/2, DNS), dynamic implant generation for various OS targets, and features such as in-memory shellcode execution, port forwarding, and process injection. There are four major components to the Sliver ecosystem:

- **Server Console** – The server console is the main interface, which is started when you run the `sliver-server` executable. The server console is a superset of the client console. All code is shared between the client/server consoles except server-specific commands related to client (e.g., operator) management. The server console communicates over an in-memory gRPC interface to the server.
- **Sliver Server** – The Sliver server is also part of the `sliver-server` executable and manages the internal database, starts/stops network listeners (such as C2 listeners, though there are other types). The main interface used to interact with the server is the gRPC (Remote Procedure Call) interface, through which all functionality is implemented.
- **Sliver Client** – The client console is the primary user interface used to interact with the Sliver server.
- **Implant** – The implant is the actual malicious code that runs on the target system you want remote access to.

In this project, we leverage Sliver not for malicious purposes but as a controlled adversary emulation framework to assist in testing the accuracy and coverage of Sysmon configuration files. Our environment consists of a Sliver server running on an Ubuntu virtual machine and a Windows 10 virtual machine acting as the target, both connected through a host-only network. Since the setup is local, we operate solely through the Sliver server console, without utilizing the standalone Sliver client. A custom Windows implant `implant.exe` is generated and executed on the target machine, allowing us to simulate adversarial behaviors such as process creation, registry-based persistence, and file operations. These activities are designed to trigger specific Sysmon events, which we monitor to evaluate the effectiveness of our configuration rules. This approach enables practical validation of the configurations built with our web application and ensures they accurately detect and log attacker-like activity in a controlled testing environment. For more information about setting up environment, please visit `https://sliver.sh/` or the main github page `https://github.com/BishopFox/sliver`

4 Web Application Overview

To assist users in authoring configuration files that adhere to best practices, we have developed a web-based application implemented using Node.js, along with standard HTML, CSS, and JavaScript for the front-end interface and styling. The overall project structure is described as followed

- **css**: This directory contains the styling files used to design the visual appearance of the web application. It defines layout, typography, and color schemes to ensure a user-friendly interface.
- **example**: This folder holds sample Sysmon configuration files in XML format. These files serve as templates or test cases that users can refer to or load into the application for validation and customization.
- **js**: The JavaScript folder includes the client-side scripts responsible for interactive functionality on the front-end. These scripts handle dynamic rendering, user input validation, and communication with the back-end.
- **node_modules**: This folder contains all third-party libraries and dependencies managed by `npm`, required for running the server and handling various Node.js functionalities.
- **index.html**: This is the main HTML file that serves as the entry point for the web interface. It structures the layout of the interface and integrates CSS and JavaScript resources for rendering the application.
- **package.json** and **package-lock.json**: These files define the project metadata, dependencies, and scripts necessary for building and running the application. They also ensure consistency across development environments.
- **README.md**: This markdown file provides an overview of the project, including installation instructions, usage guidelines, and relevant documentation for users and developers.
- **server.js**: This is the main server-side script that initializes and runs the Node.js back-end. It handles HTTP requests, serves static files, and may provide endpoints for file conversion or validation.

This application structure allows us to maintain a high level of organization and clarity within the codebase. By segregating different functionalities into distinct packages, each part of the application can be developed, tested, and maintained independently. Furthermore, the separation of concerns ensures that changes in one part of the application do not unintentionally affect other parts, thereby reducing the risk of introducing bugs. The following comprehensive use case diagram will visually represent the interactions between the user and the web application, further clarifying the flow and functionality of the application.

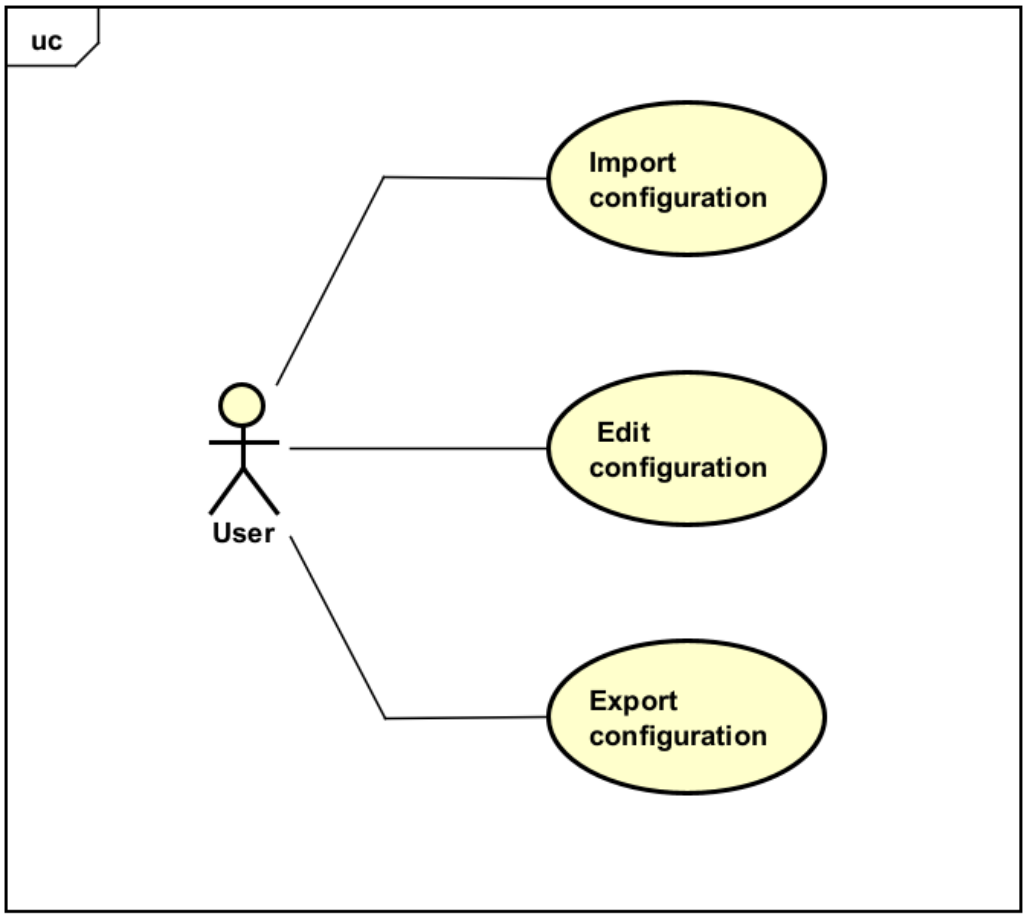

Figure 8: Use Case Diagram

5 Implementation Details

The core functionality of the web-based Sysmon configuration assistant is implemented using a modular front - end structure built with HTML, CSS, and JavaScript. The application is designed to abstract away the complexity of Sysmon's configuration logic, especially when dealing with filtering inconsistencies and edge cases.

1. "config.js" – Dynamic Field Mapping and UI Logic

The `config.js` module serves as the main controller for dynamic behavior within the application. Its responsibilities include:

- **Event Field Schema:** Defines a mapping between each Sysmon Event ID (1–29) and its respective fields. This mapping (`sysmonFieldsSchema`) enables the application to generate accurate rule input forms for each event.
- **Event Logic Mapping:** Associates user-friendly event names (e.g., `ProcessCreate`) with their corresponding Event IDs. This ensures the correct identification and categorization of event rules.

- **UI Initialization:** Functions such as `initializeRuleFunctions` and `initializeModalEvents` attach listeners to UI elements for rule creation, modal popups, and drag-and-drop reordering using the SortableJS library.
- **Import/Export:** Provides support for importing Sysmon configurations from XML or TXT formats, and exporting valid configurations in XML format. These features enable seamless editing and validation of existing configuration files.
- **Rule Generation:** Dynamically creates DOM elements representing `normal` and `compound` rules, including support for field selection, condition logic, and nested grouping (AND/OR).

2. "configData.js" – Data Parsing and Format Conversion

The `configData.js` file complements `config.js` by managing the parsing and conversion of Sysmon configuration data. It includes:

- **Event Metadata:** Defines a list of all supported Sysmon events along with their IDs and labels. This is used to populate dropdown menus and labels in the UI.
- **Format Parsers:** Implements robust parsers for Sysmon configurations in XML, JSON, and TXT formats. Each parser normalizes the input into a unified JSON schema that the UI can render.
- **JSON-to-XML Conversion:** The `generateSysmonXML()` function constructs a valid XML configuration from the live UI state. It respects the Sysmon schema and event filtering hierarchy.
- **Rule Reconstruction:** Functions like `addImportedNormalRule()` and `addImportedCompoundRule()` recreate UI components from imported data to allow for visual editing of pre-existing rules.
- **Helper Utilities:** Includes utility functions such as `escapeHTMLAttribute` and `getEventId` to assist with safe DOM handling and event lookups.

6 Application features showcase

6.1 Import configuration

User when running application will be greeted with a blank configuration screen to edit configuration. Then user will be able to import configuration to be appeared on screen.

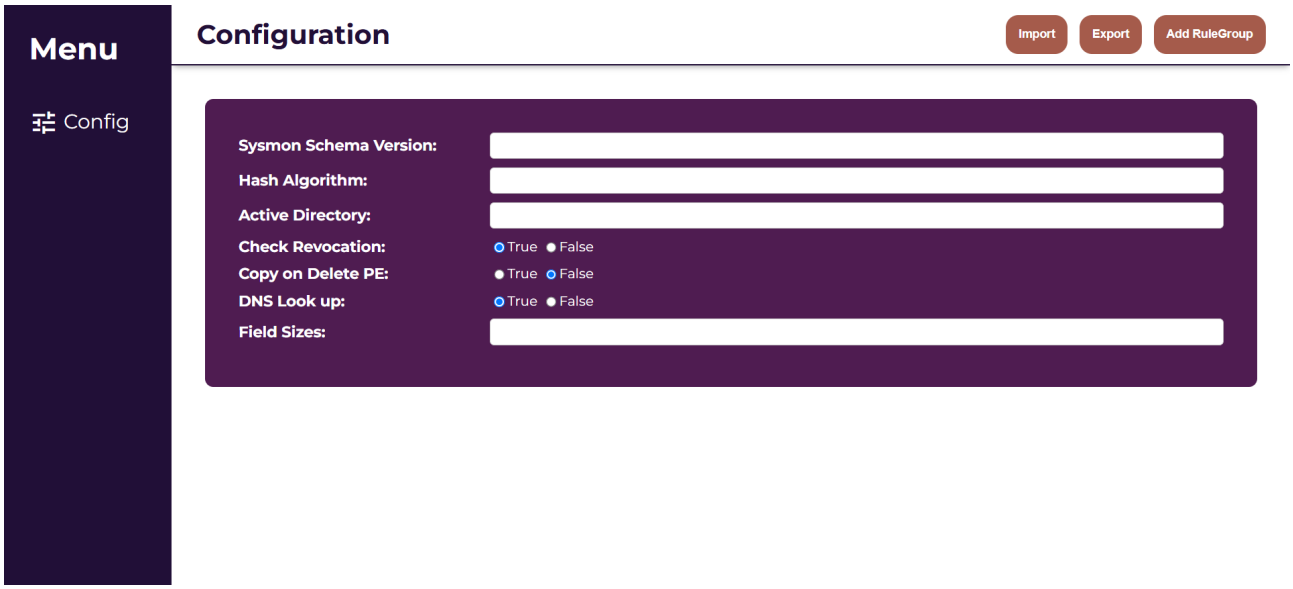

Figure 9: Main application scene

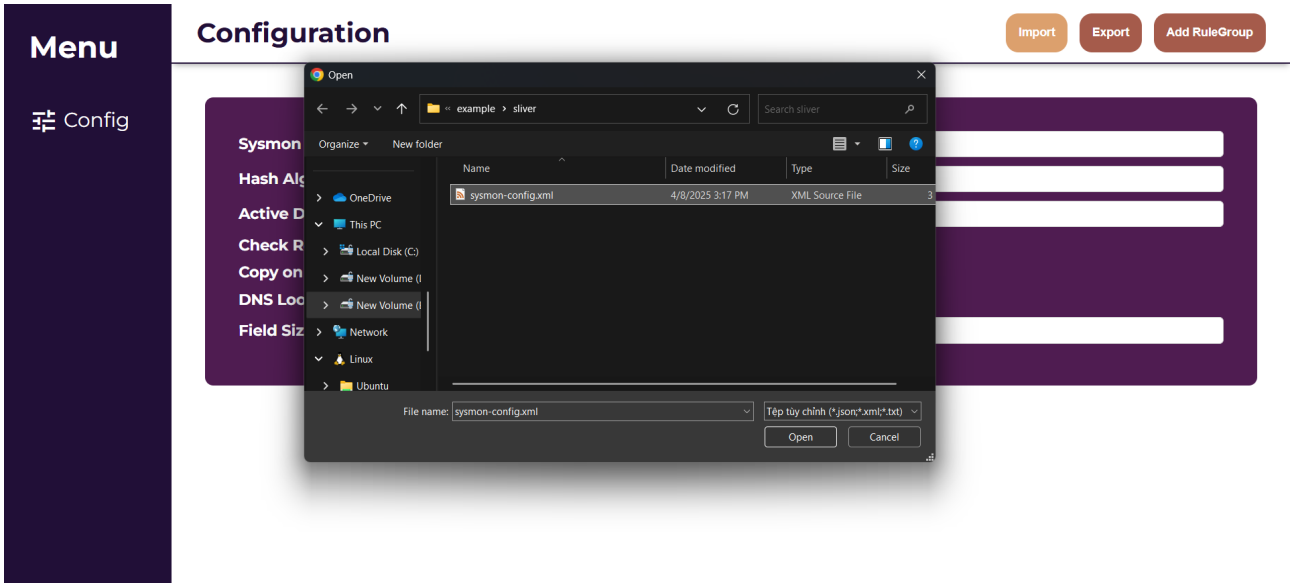

Figure 10: Importing xml configuration file

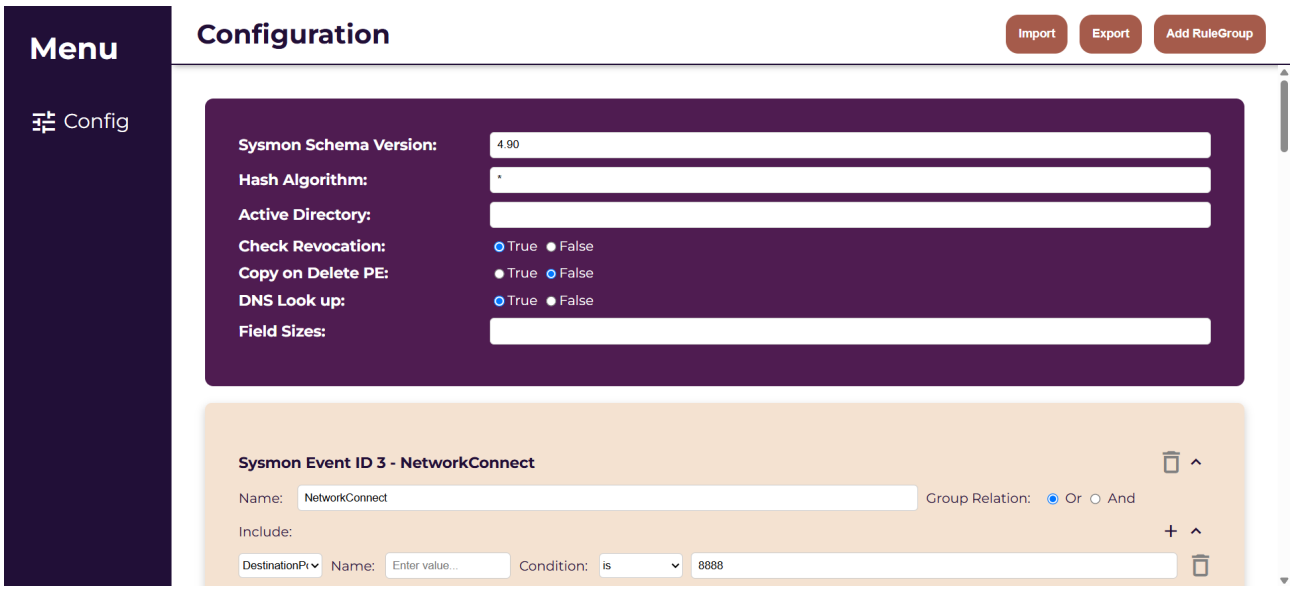

Figure 11: Successfully import configuration file

The web application support 2 type of formats as mentioned above: `txt` and `xml`. Sysmon configuration file is usually written in `xml` format but in case where user has already applied the configuration but lost the file, they can retrieve the configuration by submitting the through `txt` format by typing `sysmon -c > [filename].txt` in terminal window.

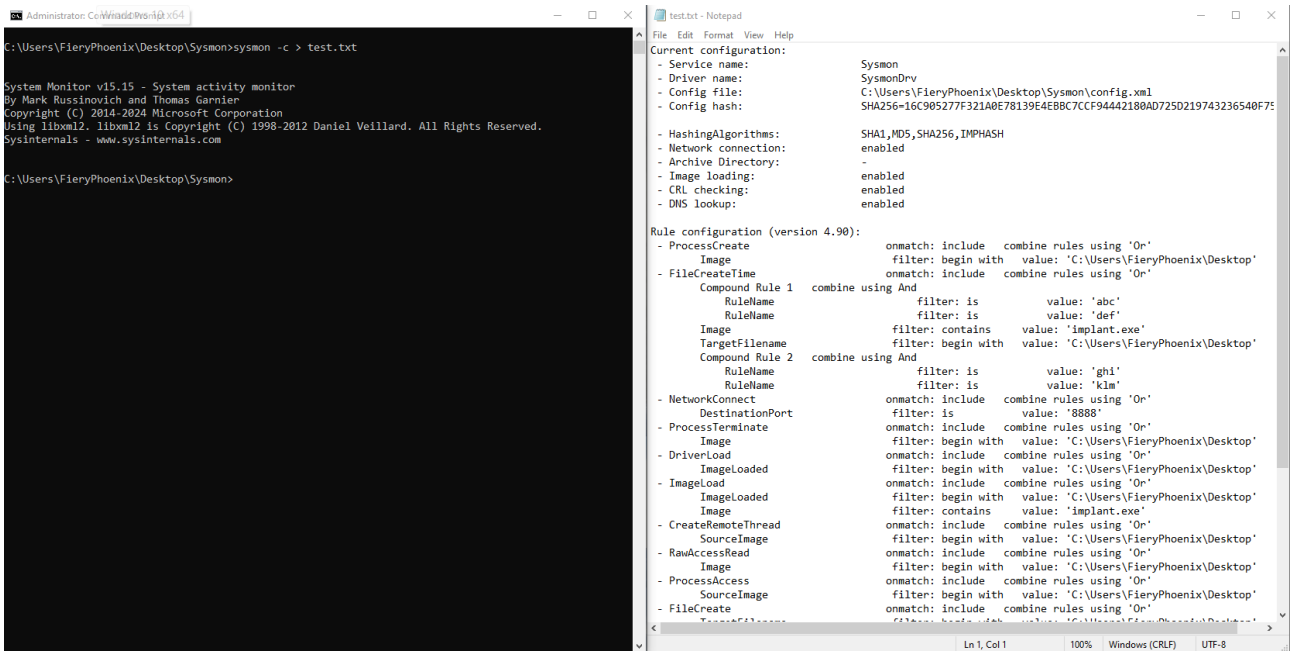

Figure 12: Acquire sysmon configuration through terminal

6.2 Edit configuration

User will be given with many capabilities to create a sysmon configurationfile, including:

6.2.1 Add RuleGroup (Add event)

Base on the convention, each event should be wrapped around by a rulegroup, user will be able to add event type supported by Sysmon. This rulegroup will be placed at the bottom as Sysmon configuration file is ordered-type.

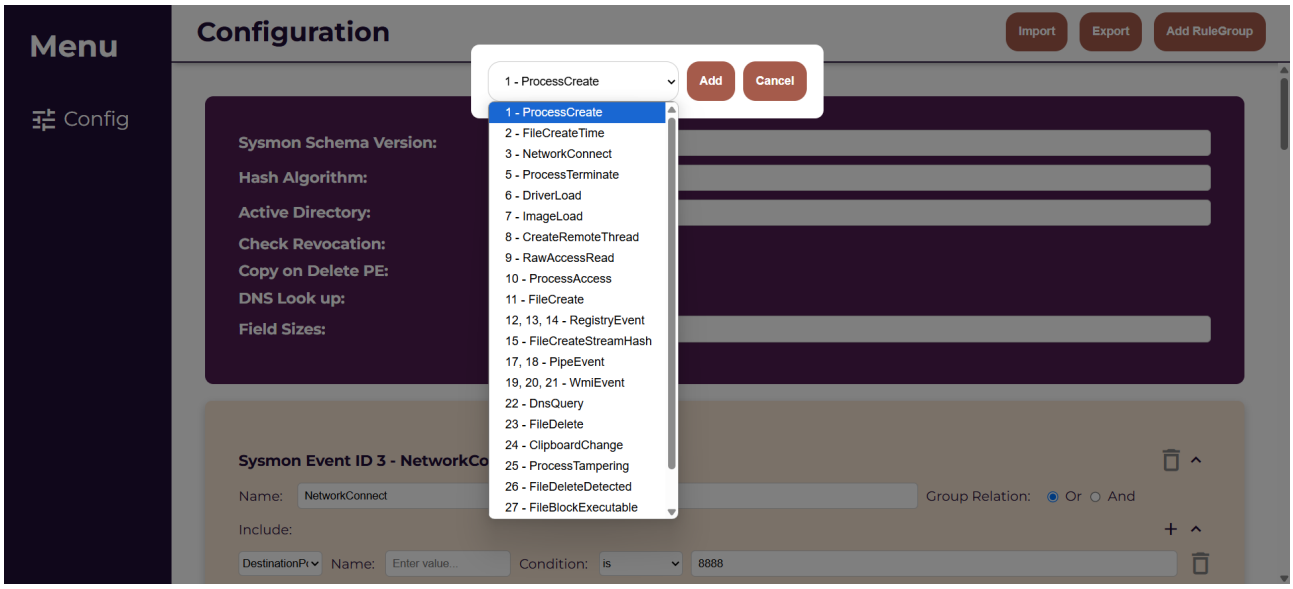

Figure 13: Adding new rule group (event)

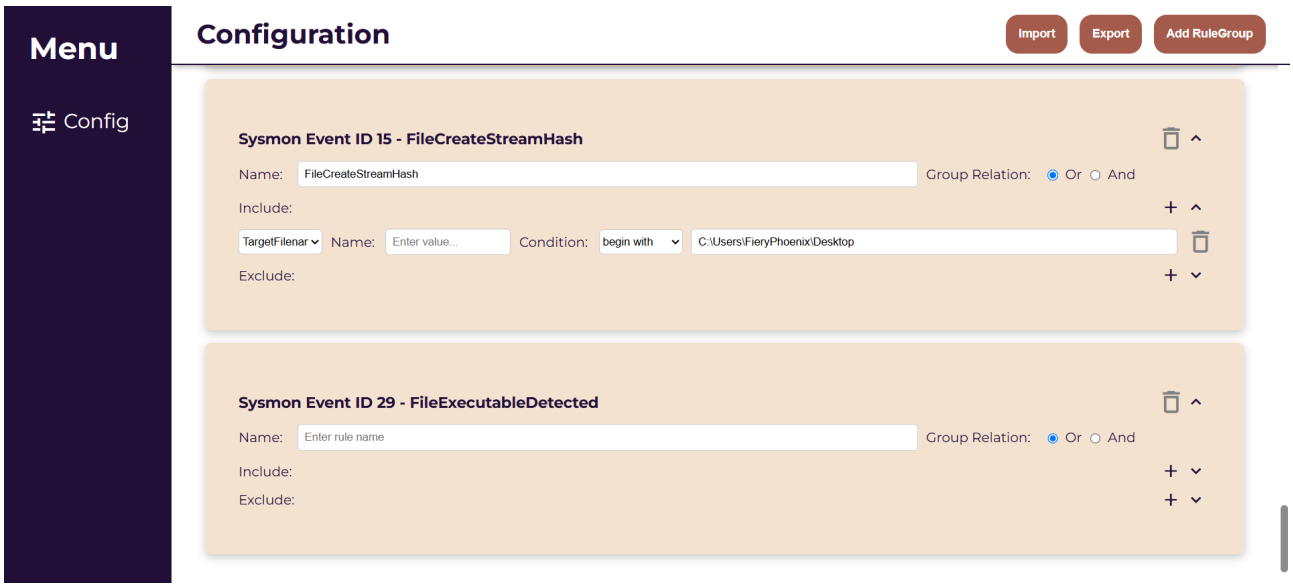

Figure 14: Successfully add new rulegroup

6.2.2 Add Rule / Compound Rule

For each event type, along with it is a subset of rules / compound rules that can be followed and edited. User can add new rule by click on "plus" icon, edit by filling text holder and delete by clicking "trash" icon. If adding compound rule user can also add sub-rules to it.

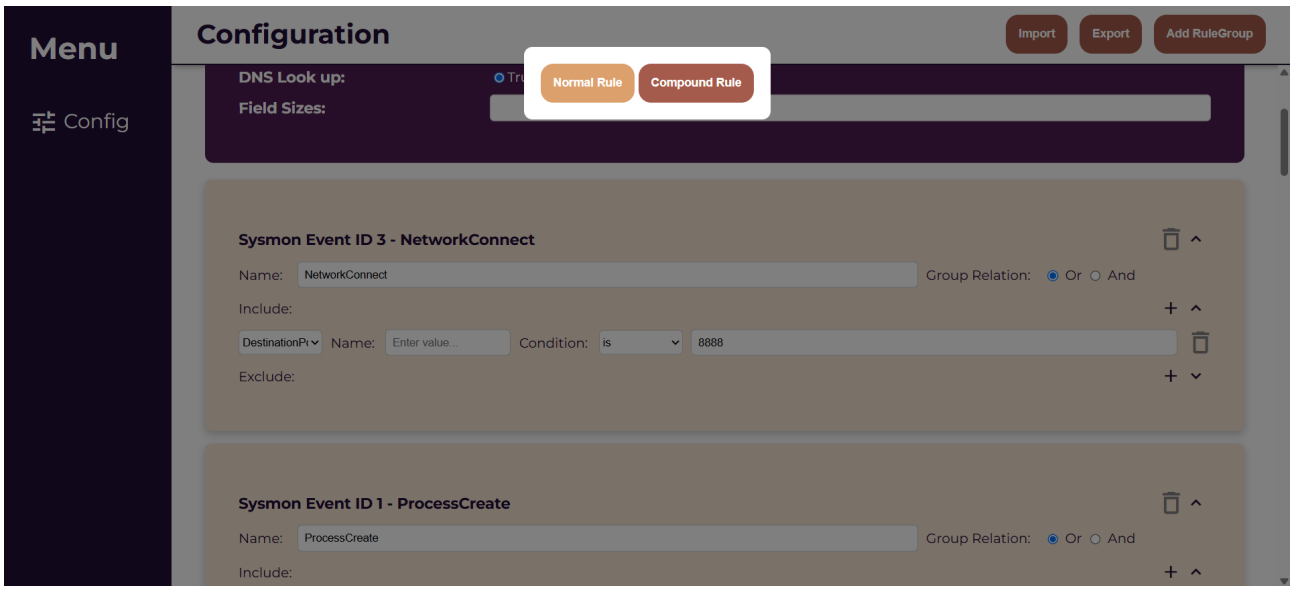

Figure 15: Add new compound rule to event

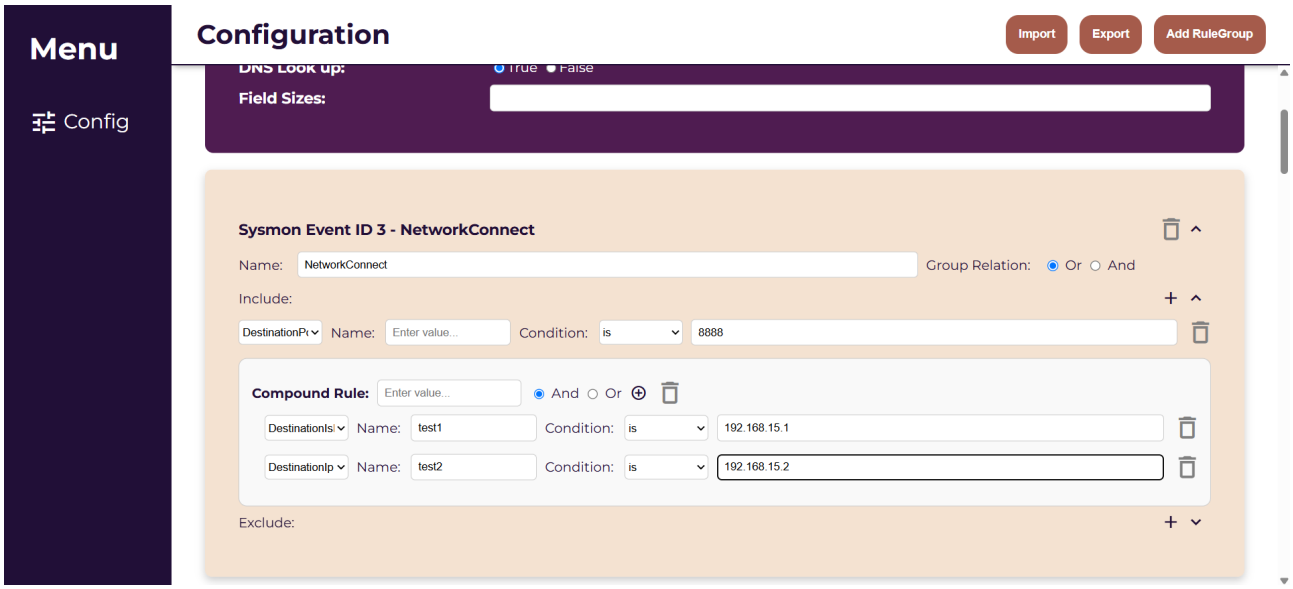

Figure 16: Successfully add new compound rule

6.2.3 Arrange rulegroup / rule / compound rule

Because Sysmon configuration depend on the order of each rulegroup, rule and compound rule, we also implement dragging element that helps user arranging order efficiently.

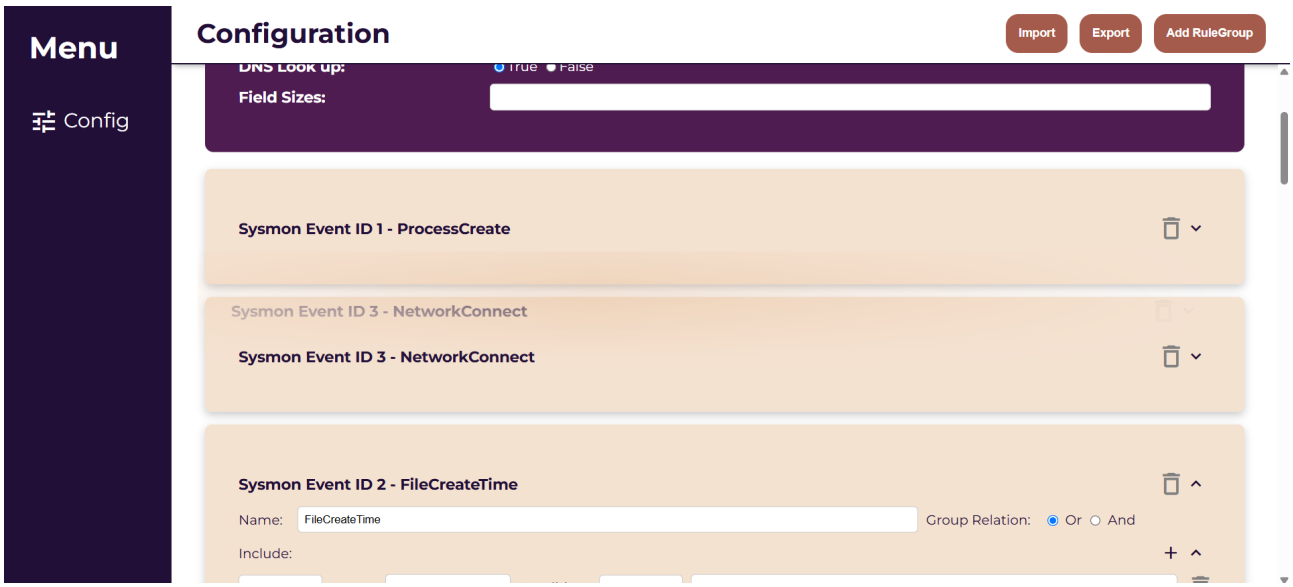

Figure 17: Dragging element in application

6.3 Export configuration

After finish editing the configuration, user can export the file into xml format, which is the format Sysmon use to load the configuration.

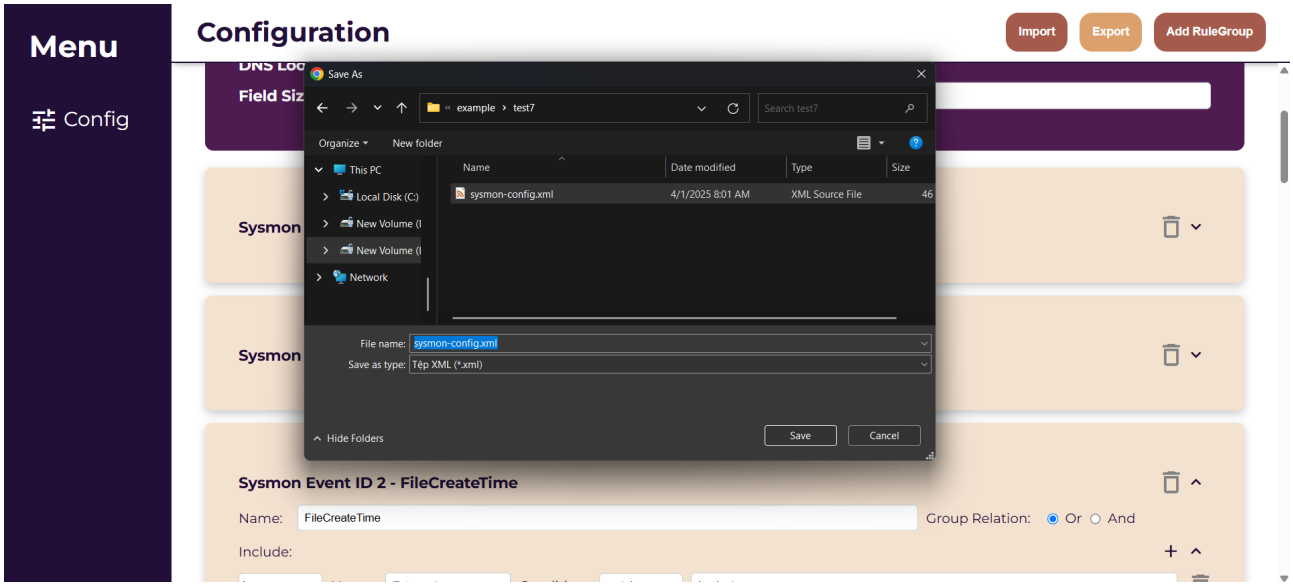

Figure 18: Export configuration

Figure 19: Final configuration in xml format

7 APPLY SYSMON IN DETECTING SLIVER IOC

Indicators of Compromise (IOCs) are forensic artifacts pieces of data that suggest a system may have been breached or is currently under attack. They act like digital "fingerprints" left behind by malware or attackers. Different types of Indicators of Compromise are used in cybersecurity [4]. Some of these include:

- **File-based Indicators** – These are associated with a specific file, such as a hash or file name.
- **Network-Based Indicators** – Indicators associated with a network, such as an IP address or domain name.
- **Behavioral Indicators** – These indicators are associated with the behavior of a system or network, such as unusual network traffic or unusual system activity.
- **Artifact-Based Indicators** – These are indicators associated with the artifacts left behind by an attacker, such as a registry key or a configuration file.

To identify Indicators of Compromise (IoCs) related to the Sliver framework, we focus on four key aspects: **process creation**, **network connections**, **file modifications**, and **persistence mechanisms**. Each of these areas provides forensic visibility into the behavior of the implant during execution.

It is important to note that Sliver does not natively implement persistence mechanisms, as confirmed in the official GitHub issue tracker [5]. Therefore, to simulate persistence behavior, we manually connect to the victim machine through command server (in our case Ubuntu) and modify the Windows Registry through shell to execute the implant on user login.

Before analyzing events, we must first construct a valid Sysmon XML configuration file that filters relevant activities. To effectively isolate behaviors associated with the implant, we begin by logging all system events, then excluding default system processes and directories. We include only those events in which the image path or working directory matches the expected location of the implant. The link to configuration file used to achieve that is at `https://github.com/bananagobananza/SysmonConfigurationBuilder/blob/main/example/sliver/sysmon-config.xml`

In our experiment, the `implant.exe` file was deployed on the user's **Desktop** folder. As such, our configuration file is designed to filter and include only those events that originate from image paths beginning with `C:\Users\<username>\Desktop`, thereby capturing actions specifically linked to the Sliver implant. If we look closely at the XML configuration, we adopt a technique where certain event types are included without specifying any filtering conditions. This instructs Sysmon to include the corresponding event type in the system log and capture all instances of that event. This approach is particularly useful during initial testing, when the objective is to gather a comprehensive view of all activity related to a specific event ID before applying more restrictive filters.

With the Sysmon configuration file properly constructed and deployed, we proceed to analyze the event logs generated during the execution of the Sliver implant.

7.1 Initial setup

To start, we create `implant.exe` and run it in victim's machine. In the command server we create a domain and open port for communication between command server and implant.

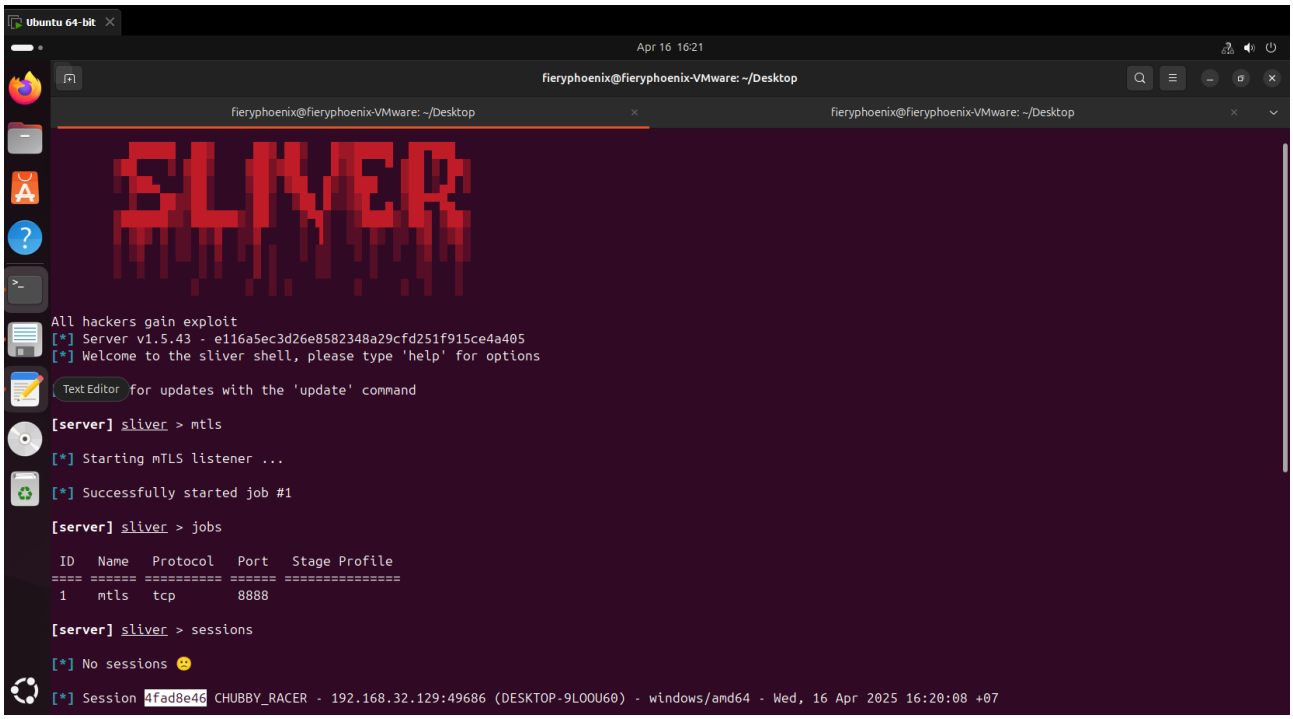

Figure 20: Command server initialization

The command server open port **8888** with IP **192.168.32.128** in order to listen and give control to implant. Next up, we implement persistent to victim's machine by connect to victim's shell and modify the registry key of implant file.

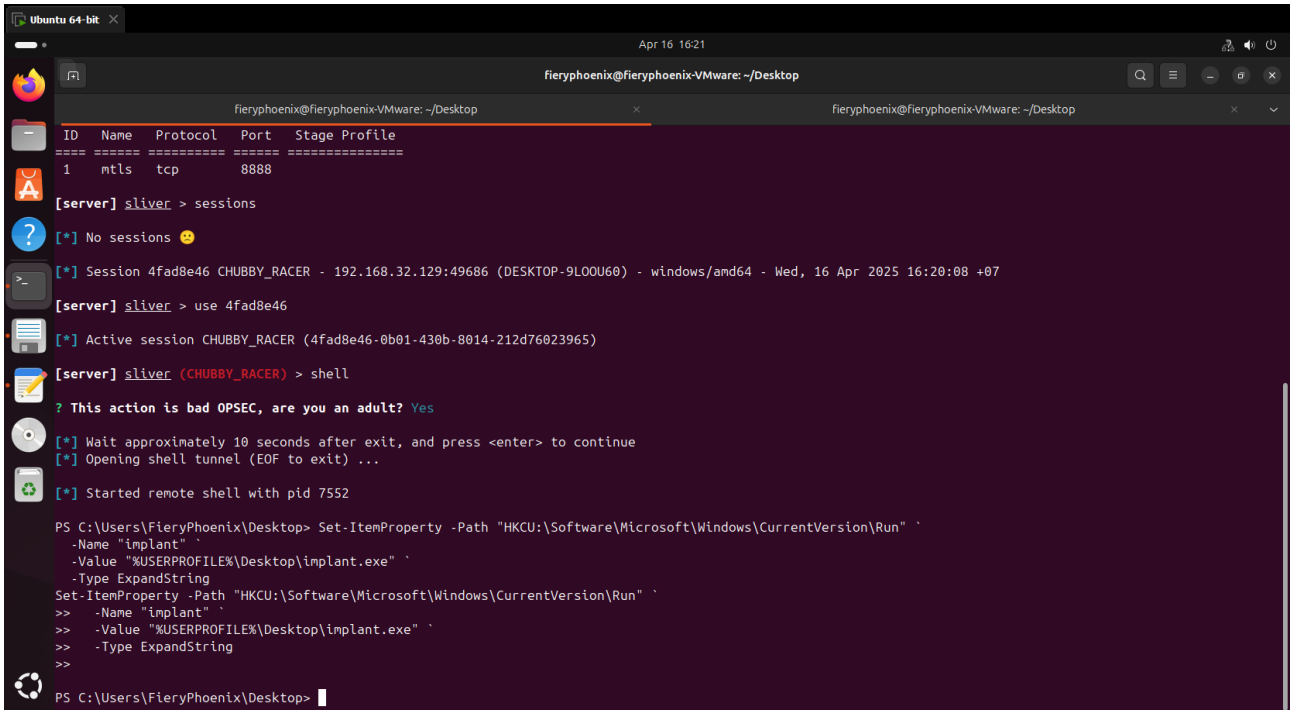

Figure 21: Connect to implant and modify registry keys

Finally we upload `test.txt` file to victim's machine via command `upload` and finish the set up.

```
[server] sliver (CHUBBY_RACER) > upload test.txt

[*] Wrote file to C:\Users\FieryPhoenix\Desktop\test.txt

[server] sliver (CHUBBY_RACER) >
```

Figure 22: Upload file to victim's machine

7.2 Identifying IoC of Sliver

Now we move to victim's machine and watch sysmon log in event viewer. Right at the start we see `implant.exe` create a process and try to load as many modules of itself as possible:

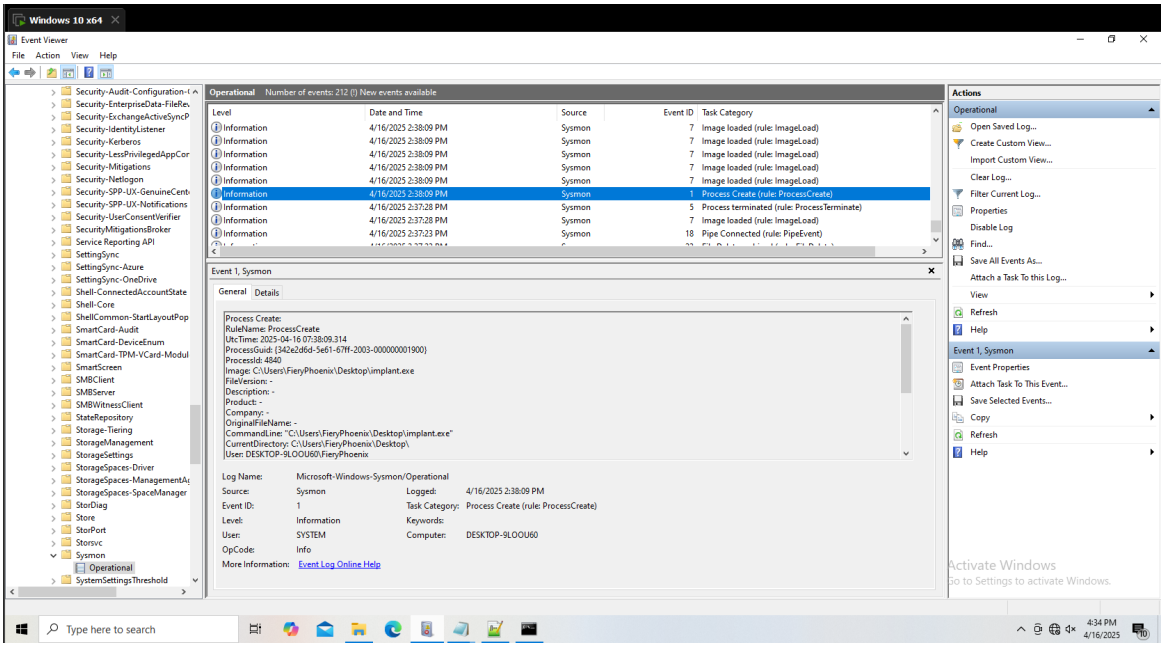

Figure 23: Sysmon event ID 1: Process Creation

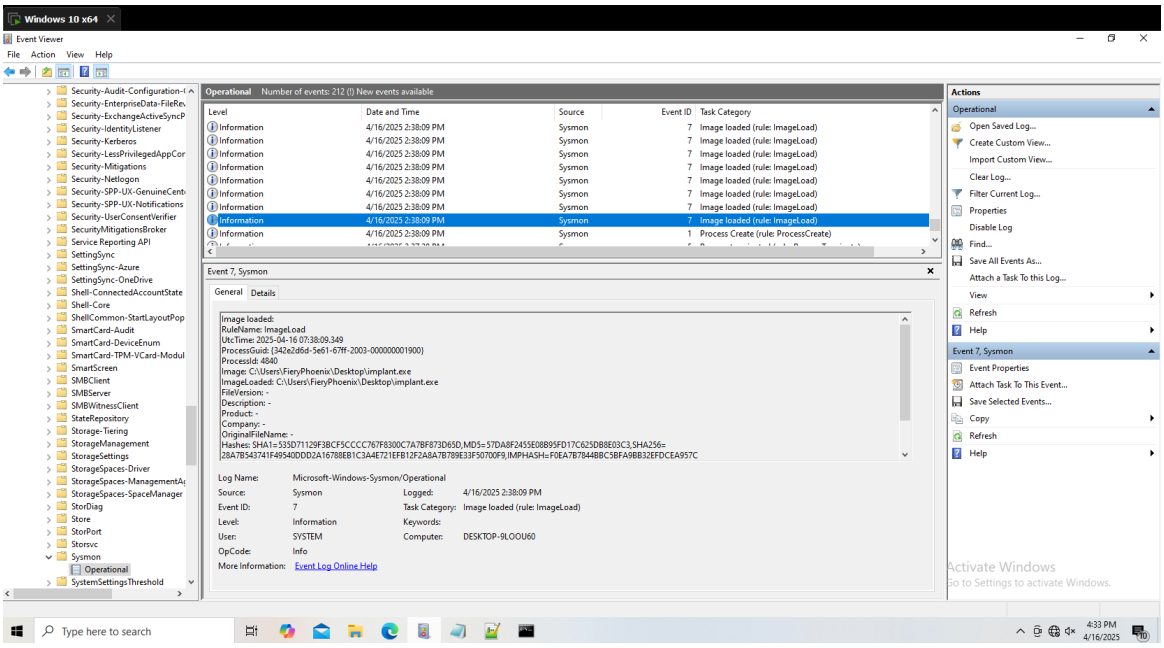

Figure 24: Sysmon event ID 7: Image Loaded

Next, the implant attempts to establish a connection to the command and control (C2) server. As shown in the configuration, the Sliver server is listening on port **8888** at IP address **192.168.32.128**. This activity is observable in Sysmon under Event ID 3 (NetworkConnect).

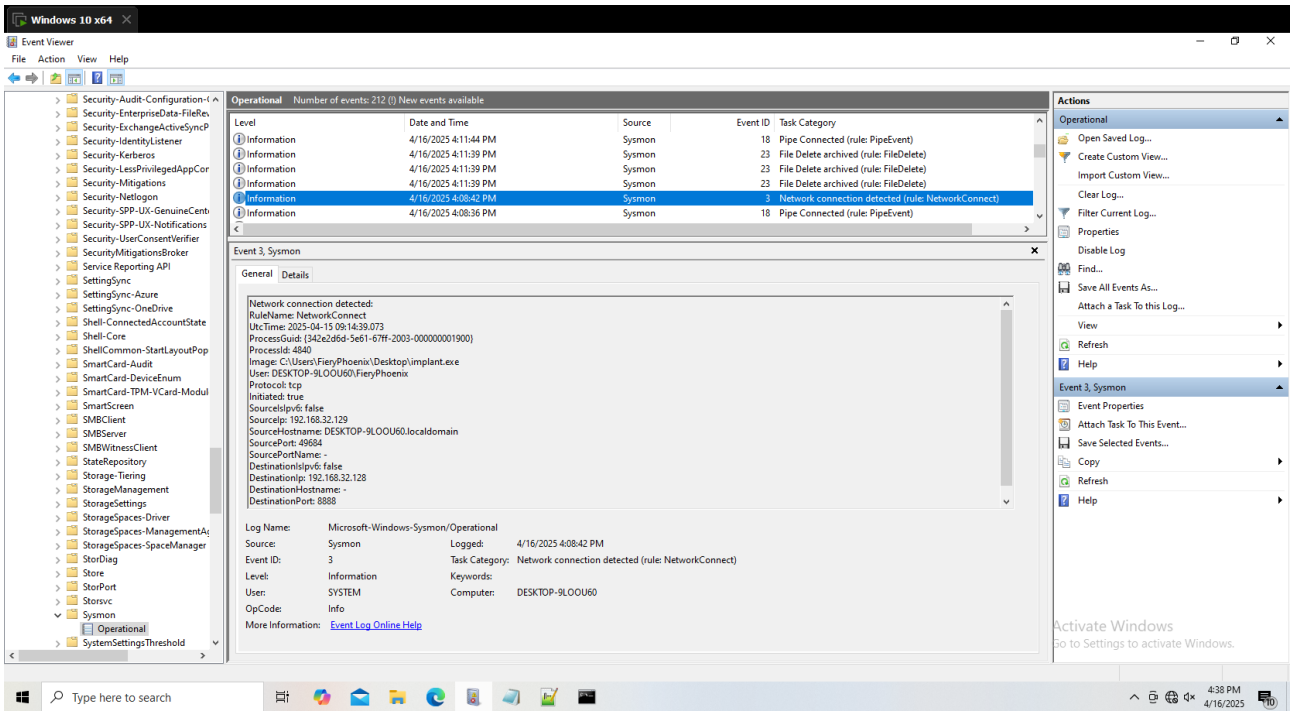

Figure 25: Sysmon event ID 3: Network connected

After that, the implant achieve persistent through modification of registry, this is also logged by Sysmon at event ID 13.

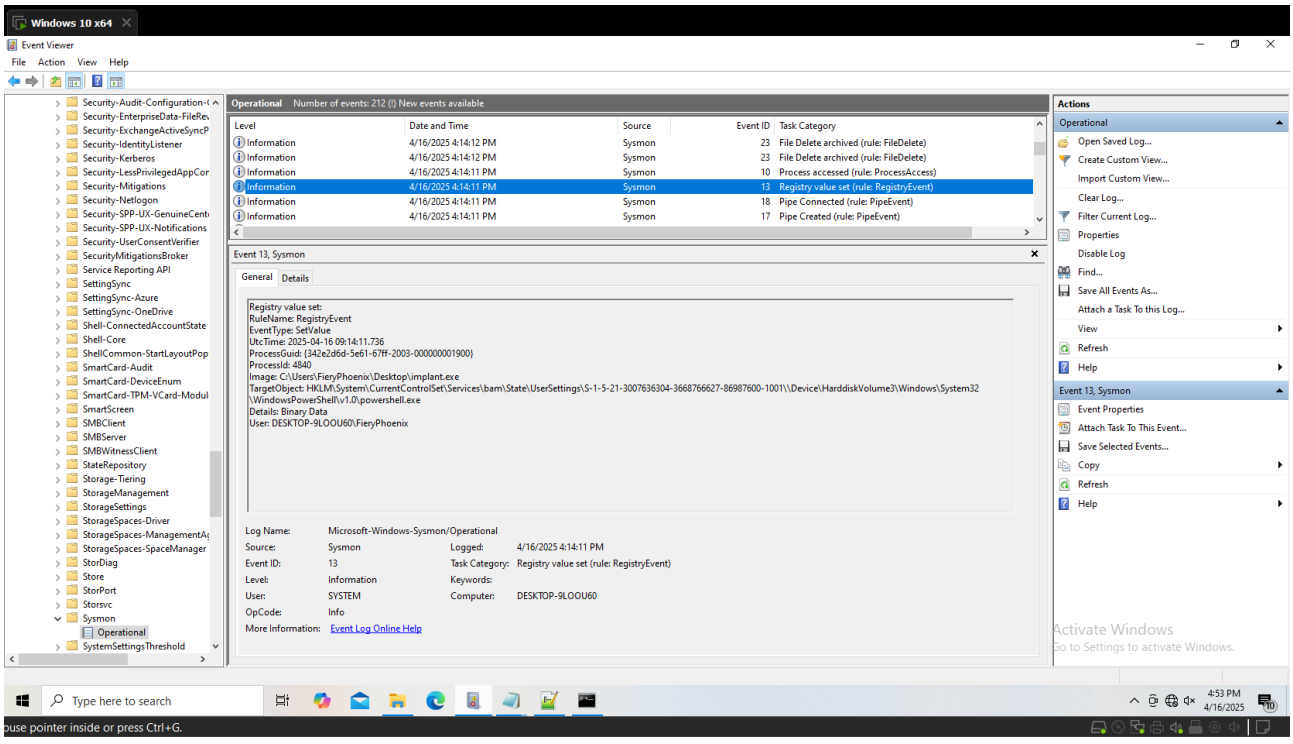

Figure 26: Sysmon event ID 13: RegistryEvent (Value Set)

Finally, sysmon also notice the creation of a new file name test.txt in Desktop.

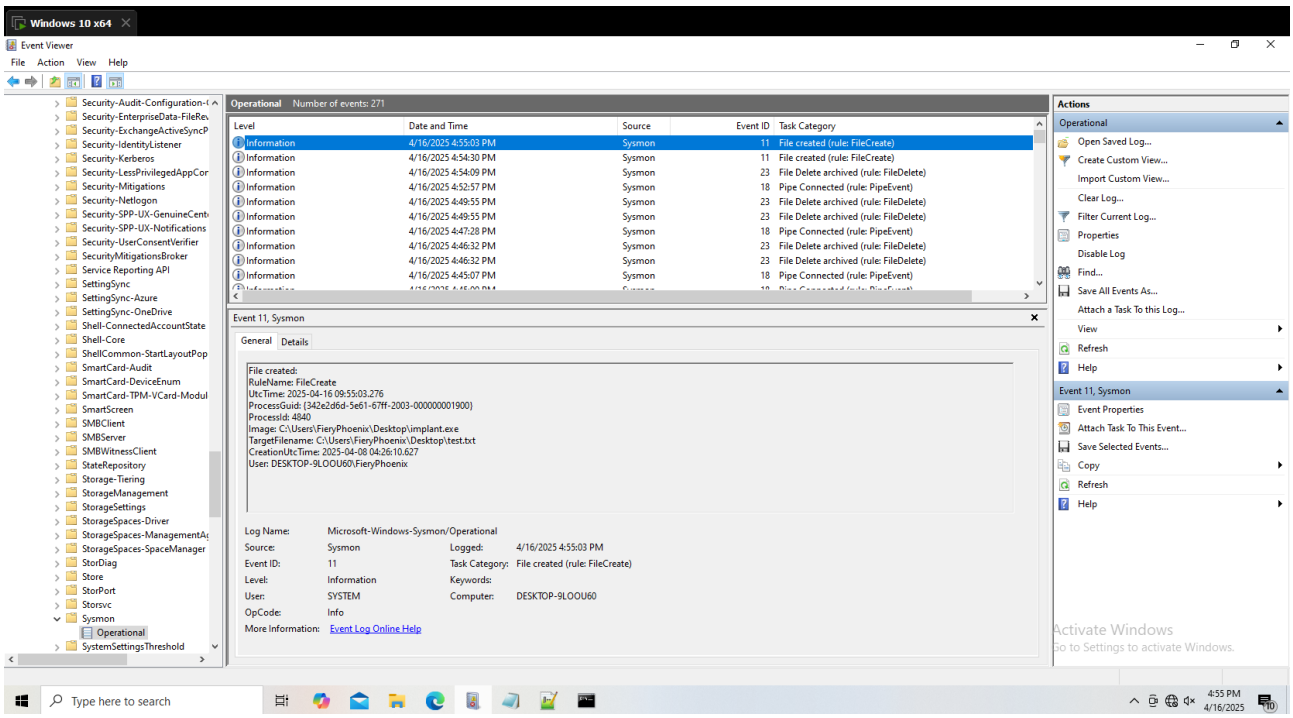

Figure 27: Sysmon event ID 11: FileCreate

Through the deployment and controlled execution of the Sliver implant, we identified several key Indicators of Compromise (IoCs) that align with common attacker techniques and behaviors. These IoCs were observed and recorded using Sysmon, mapped across multiple event IDs:

- **Process Creation (Event ID 1)**: Upon execution, `implant.exe` spawns a new process on the victim machine. This is a clear indicator of initial execution activity.
- **Module Loading (Event ID 7)**: The implant loads numerous modules and libraries, which may indicate reflective DLL injection or preparation for in-memory execution techniques.
- **Network Connection (Event ID 3)**: The implant establishes an outbound connection to the command and control server at IP `192.168.32.128`, port `8888`, using mTLS as the communication protocol. This connection facilitates remote control and command execution.
- **Registry Modification (Event ID 13)**: To simulate persistence, the implant modifies the Windows Registry key `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` to ensure the implant is executed on user login.
- **File Creation (Event ID 11)**: A file named `test.txt` is uploaded to the Desktop using Sliver's `upload` command. This file write operation is logged and serves as evidence of post-exploitation activity.

These artifacts collectively form the behavioral and artifact-based IoCs of Sliver in this controlled test. Monitoring for these events provides valuable detection opportunities for blue teams and defenders seeking to identify the presence or activity of Sliver-like frameworks in their environments.

8 Discussion - Building good configuration file

Crafting an effective Sysmon configuration file is a critical task for cybersecurity professionals aiming to enhance system monitoring and threat detection on Windows systems. Sysmon, a component of Microsoft's Sysinternals suite, provides detailed logging of system activities, such as process creations, network connections, and registry modifications, which are invaluable for identifying malicious behaviors and supporting incident response. However, the effectiveness of Sysmon hinges on its configuration, typically defined in an XML file, which determines which events are logged and how they are filtered. A "good" configuration strikes a balance between comprehensive threat detection, minimal false positives, and acceptable system performance, tailored to the specific needs of the organization's environment and threat landscape.

The foundation of a good Sysmon configuration lies in aligning logging rules with the organization's security objectives and the specific roles of the systems being monitored. A one-size-fits-all approach is ineffective, as the requirements for a high-security server handling sensitive financial data differ significantly from those of a general-purpose workstation. For instance, servers may require verbose logging of all process creations (Event ID 1) and network connections (Event ID 3) to detect advanced persistent threats (APTs), while workstations might prioritize lighter logging to minimize performance impact. To achieve this alignment, administrators should start with established configuration templates, such as those provided by SwiftOnSecurity [6] or Sysmon-modular [7]. These templates are designed to cover a broad spectrum of security events, mapping to MITRE ATT&CK techniques to detect common adversarial tactics, such as credential dumping (T1003) or timestomping (T1070.006). For example, a rule might log process creations involving suspicious command-line arguments, like those associated with `ntdsutil.exe`, while excluding trusted processes like `chrome.exe` to reduce noise.

Noise reduction is a critical aspect of building a good Sysmon configuration. Logging too many events can overwhelm security teams with false positives, making it difficult to identify genuine threats, while logging too few events risks missing critical indicators of compromise. Effective configurations include targeted include rules to capture high-risk activities and comprehensive exclude rules to filter out legitimate behaviors. For instance, monitoring file creation time changes (Event ID 2) in directories like `C:`, which are often used for malicious activities, is essential, but excluding activities from trusted applications like OneDrive or antivirus software prevents unnecessary alerts. Resources like Sysmon-modular provide pre-built exclusion lists for common software, such as Microsoft Office and Adobe products, which administrators can adapt to their environment. However, care must be taken to avoid overly broad exclusions that could create blind spots. For example, excluding all activities from `svchost.exe` might miss malicious instances of this process, which is commonly abused by adversaries.

Performance optimization is another key consideration in Sysmon configuration. While verbose logging enhances detection capabilities, it can strain system resources, particularly on high-traffic servers or resource-constrained endpoints. Features like DNS lookup for network connections or file archiving for deleted executables provide valuable context for

forensic analysis but can significantly increase CPU and disk usage. A good configuration disables such features by default unless they are critical to the organization's security strategy. For example, the Sysmon-modular template with medium verbosity, as seen in configurations like `config.xml`, disables DNS lookup and certificate revocation checks to minimize overhead while still capturing essential events. Administrators must assess their systems' performance constraints and adjust settings accordingly, testing configurations in a controlled environment to ensure they do not degrade system functionality. Tools like Windows Performance Monitor can help evaluate the impact of Sysmon logging on CPU, memory, and disk resources.

Rule ordering is a subtle but critical factor in Sysmon configurations. Sysmon processes include rules before exclude rules within each event type, meaning that incorrect ordering can lead to missed detections. For instance, if an exclude rule for `calc.exe` is placed before an include rule targeting suspicious process creations, malicious instances of `calc.exe` might be overlooked. A good configuration carefully structures rules to prioritize specific detections while ensuring exclusions do not inadvertently suppress critical events. Documentation from Sysmon-modular emphasizes this principle, recommending that administrators validate rule ordering using Sysmon's `-c` command to preview logged events. Additionally, configurations should be modular, grouping rules by event type (e.g., process creation, network connection) and purpose (e.g., threat detection, compliance) to simplify maintenance and updates.

Customization is essential to address organization-specific threats and operational requirements. While standard configurations provide a strong starting point, they may not fully account for unique risks, such as industry-specific attack vectors or internal threats. Administrators should analyze their threat landscape, using threat intelligence feeds or past incident data to identify relevant indicators of compromise. For example, an organization in the healthcare sector might add rules to detect unauthorized access to patient data directories, while a financial institution might focus on monitoring for ransomware-related file modifications (Event ID 11). Compliance requirements, such as those mandated by GDPR or PCI-DSS, may also dictate specific logging needs, such as tracking access to sensitive registry keys (Event ID 13). Custom rules should be tested incrementally to avoid introducing errors or excessive logging, using tools like Sysmon's event viewer or third-party log analysis platforms to verify their effectiveness.

Regular maintenance and updates are vital to ensure a Sysmon configuration remains effective over time. The threat landscape evolves rapidly, with adversaries developing new techniques to evade detection. Configurations must be revisited periodically to incorporate new MITRE ATT&CK mappings, update exclusion lists for newly installed software, and address emerging threats. For example, the rise of living-off-the-land (LotL) attacks, which leverage legitimate tools like PowerShell, may necessitate stricter rules for monitoring script executions (Event ID 1). Community resources, such as the Sysmon-config repository [6] or Olaf Hartong's Sysmon-modular project [7], provide updates and new rules that administrators can integrate into their configurations. Automated tools, like web-based configuration editors, can simplify this process by allowing users to import, modify, and export XML files without deep technical expertise, as highlighted in recent Sysmon management solutions.

Testing and validation are integral to building a good Sysmon configuration. Before deploying a configuration in a production environment, administrators should test it in a lab setting, simulating both legitimate and malicious activities to assess its detection capabilities and performance impact. Tools like the Atomic Red Team framework can simulate MITRE ATT&CK techniques to verify that rules capture expected events without generating excessive false positives. Log analysis platforms, such as Splunk or Elastic Stack, can help correlate Sysmon events with other telemetry to evaluate the configuration's effectiveness in a broader security context. Validation should also include stakeholder feedback, ensuring that the configuration meets the needs of security analysts, incident responders, and compliance teams.

In conclusion, building a good Sysmon configuration file requires a strategic approach that balances detection, performance, and maintainability. By starting with established templates, aligning rules with organizational needs, and optimizing for performance, administrators can create configurations that enhance threat detection without overwhelming resources. Careful attention to noise reduction, rule ordering, and customization ensures that Sysmon captures relevant events while minimizing false positives. Regular updates and rigorous testing keep the configuration aligned with evolving threats and operational requirements. Ultimately, a well-crafted Sysmon configuration serves as a cornerstone of an organization's cybersecurity strategy, empowering security teams to detect and respond to threats effectively in an increasingly complex threat landscape.

9 CONCLUSION

The development and evaluation of the web-based Sysmon configuration assistant, coupled with its application in detecting Indicators of Compromise (IoCs) from the Sliver C2 framework, yielded significant insights into both system monitoring and adversary emulation. The project successfully demonstrated the feasibility of simplifying Sysmon configuration authoring through a user-friendly interface, while also validating its effectiveness in a controlled testing environment. However, several aspects of the implementation, testing methodology, and practical applicability warrant further discussion to contextualize the results and identify avenues for improvement.

The web application effectively addressed the complexity of Sysmon configuration files, which, as outlined in Basic information section, rely on intricate XML structures and precise rule ordering to achieve accurate event filtering. By abstracting the configuration logic into an intuitive interface, the application enabled users to import, edit, and export Sysmon configurations without deep expertise in XML or Sysmon's schema. Features such as dynamic rule generation, drag-and-drop reordering, and support for multiple input formats (XML and TXT) significantly reduced the barrier to entry for creating valid configurations. The use of modular JavaScript components, such as `config.js` and `configData.js`, ensured robust handling of event schemas and filtering logic, mitigating common pitfalls like incorrect rule precedence or logical operator misuse (e.g., AND/OR inconsistencies). This aligns with best practices for Sysmon configuration, as emphasized in community resources like the Sysmon Community Guide [8], which advocate for structured and well-documented rule sets.

The controlled testing with Sliver provided a practical validation of the Sysmon configurations generated by the application. As detailed in Sliver section, the configuration successfully captured key IoCs, including process creation (Event ID 1), network connections (Event ID 3), registry modifications (Event ID 13), and file creation (Event ID 11). These events mapped directly to Sliver's behaviors, such as establishing mTLS connections to the C2 server at 192.168.32.128:8888 and modifying registry keys for persistence. The ability to detect these artifacts underscores the application's utility in real-world scenarios, where blue teams rely on Sysmon to identify adversarial tactics, techniques, and procedures (TTPs) as defined by frameworks like MITRE ATT&CK. The initial broad logging approach, followed by targeted filtering for implant-related paths (e.g., `C:\Users\<username>\Desktop`), proved effective in isolating relevant events while minimizing noise from legitimate system processes.

Despite these successes, several limitations emerged during the project. First, the testing environment was constrained to a single Windows 10 virtual machine and a local Sliver server, which may not fully represent the complexity of enterprise networks. Real-world deployments often involve diverse endpoints, varying Windows versions, and noisy environments with legitimate applications generating high event volumes. The configuration's reliance on path-based filtering (e.g., Desktop directory) could miss Sliver implants deployed in less predictable locations, such as `C:\ProgramData` or `C:\Windows\Temp`, which are commonly abused by attackers. Future iterations should incorporate more dynamic filtering criteria,

such as behavioral patterns or command-line arguments, to enhance detection robustness.

Second, the web application's usability, while improved over manual XML editing, could benefit from additional features to support advanced users. For instance, integrating real-time validation against the Sysmon schema or providing templates based on community configurations (e.g., Sysmon-modular or SwiftOnSecurity) could streamline rule creation. The drag-and-drop functionality, while innovative, requires users to understand Sysmon's rule precedence, which may still pose a learning curve. Automated suggestions for rule ordering or conflict detection could further enhance usability, particularly for complex configurations with multiple `RuleGroup` elements.

The use of Sliver as an emulation framework highlighted both its strengths and limitations. Sliver's flexibility in generating custom implants and supporting multiple protocols (mTLS, HTTP/2) made it an ideal choice for simulating modern adversary behaviors. However, its lack of native persistence mechanisms, as noted in the Sliver GitHub issue tracker [5], required manual registry modifications to simulate realistic attack scenarios. This manual intervention limited the scope of automated testing and may not fully capture the stealth of advanced persistent threats (APTs). Future experiments could leverage other frameworks, such as Cobalt Strike or Covenant, to test the configuration against a broader range of TTPs, including in-memory execution and obfuscated payloads.

From a practical perspective, the project underscores the importance of iterative configuration tuning. The initial broad logging approach, while useful for identifying Sliver IoCs, generated significant event volumes that could overwhelm Security Information and Event Management (SIEM) systems in production. Refining exclusions for legitimate processes (e.g., `svchost.exe`, `explorer.exe`) and prioritizing high-value events (e.g., process injection, credential dumping) are critical for operational deployment. Additionally, integrating the application with SIEM platforms or threat intelligence feeds could enable real-time updates to rules based on emerging threats, enhancing its relevance in dynamic security environments.

Looking forward, several improvements could elevate the project's impact. First, expanding the testing environment to include diverse operating systems (e.g., Windows Server, Windows 11) and network configurations would validate the configuration's generalizability. Second, incorporating machine learning or heuristic-based rule suggestions could automate the identification of optimal filtering criteria, reducing manual effort. Finally, open-sourcing the application and integrating it with community-driven repositories, such as Olaf Hartong's Sysmon-modular [7], could foster collaboration and accelerate adoption among security practitioners.

In conclusion, the project demonstrated a practical approach to simplifying Sysmon configuration authoring and validated its effectiveness in detecting Sliver IoCs. While the web application and testing methodology achieved their objectives, addressing the identified limitations—such as testing scope, usability enhancements, and configuration tuning will be crucial for real world applicability. These findings contribute to the broader discourse on endpoint monitoring and adversary emulation, emphasizing the need for accessible tools and robust testing to counter evolving cyber threats.

REFERENCES

[1] Microsoft. *Sysmon*. URL: https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon.

[2] Microsoft. *Sysmon*. URL: https://github.com/trustedsec/SysmonCommunityGuide.

[3] BishopFox. *Sliver*. URL: https://sliver.sh/.

[4] SentinelOne. *What are Indicators of Compromise (IoCs)?* URL: https://www.sentinelone.com/cybersecurity-101/threat-intelligence/what-are-indicators-of-compromise-iocs-a-comprehensive-guide/.

[5] BishopFox. *Persistence Module*. URL: https://github.com/BishopFox/sliver/issues/9.

[6] SwiftOnSecurity. *sysmon-config*. URL: https://github.com/SwiftOnSecurity/sysmon-config.

[7] olafhartong. *sysmon-modular*. URL: https://github.com/olafhartong/sysmon-modular.

[8] SysmonCommunityGuide. *TrustedSec Sysmon Community Guide*. URL: https://github.com/trustedsec/SysmonCommunityGuide/.