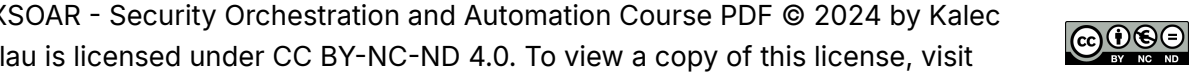
XSOAR - Security Orchestration and Automation Course - Kalec Blau

Version 1.0.0



📌 The following document is a course guide that contains all related information. For any questions or suggestions, do not hesitate to contact me.

🔒 Feel free to share this material with anyone who needs it for learning XSOAR, as long as the authorship is mentioned :)

If it has been helpful to you, please support me by purchasing my course on Udemy using this link.

https://www.udemy.com/course/xsoar-security-orchestration-and-automation-course/?referralCode=8F469AAD51A79BAA5950

Thank you very much for your support!

Kalec Blau ⚡

Happy Automating!

COUPON: HAPPYAUTOMATING50

https://www.udemy.com/course/6116421/?couponCode=HAPPYAUTOMATING50

Unlimited redemptions :)

0. Course Introduction

In this course, we will cover several key topics. First, we will explore the concept of SOAR (Security Orchestration, Automation, and Response), XSOAR and its various use cases.

Next, we will create our own XSOAR instance and perform the necessary configurations to get it up and running. As we progress, we will learn about the different components of XSOAR, such as incident types, integrations, and instances.

Finally, we will develop automations and playbooks to automate incident response tasks.

1. Introduction to SOAR and XSOAR

Key Points Summary

- Concepts: SOAR and XSOAR.
- Introduction to the XSOAR incident lifecycle.

1.1. Introduction to SOAR

SOAR stands for Security Orchestration, Automation, and Response. It's a suite of tools and processes that help organizations automate and streamline their security operations. The goal of SOAR is to improve the efficiency and effectiveness of security operations by integrating different security tools, automating repetitive tasks, and enabling faster and more accurate incident response.

Key components of SOAR

1. **Orchestration**: Connecting and integrating various security tools and systems to work together seamlessly.
2. **Automation**: Using scripts and predefined workflows to automate repetitive tasks, such as alert triage, data collection, and incident response actions.
3. **Response**: Enabling security teams to respond to threats quickly and effectively with the right information and actions.

Benefits

1. **Increased Efficiency**: Automating repetitive tasks frees up security analysts to focus on more complex issues.
2. **Improved Accuracy**: Automated processes reduce the chances of human error, leading to more reliable and consistent responses.

3. **Faster Incident Response**: Automation and orchestration allow for quicker detection and mitigation of security incidents, minimizing potential damage.
4. **Enhanced Collaboration**: Integrated tools and streamlined workflows improve communication and collaboration among security team members.
5. **Scalability**: SOAR platforms can handle large volumes of alerts and incidents, making it easier to scale security operations as the organization grows.

Use Case Examples

1. **Phishing Attack Response**:
   - **Scenario**: A user reports a suspicious email.
   - **SOAR Solution**: The SOAR platform automatically analyzes the email, checks for known indicators of compromise (IOCs), and quarantines the email if it is malicious. It then updates the security team and relevant users about the action taken.
2. **Malware Detection and Containment**:
   - **Scenario**: An endpoint detects a potential malware infection.
   - **SOAR Solution**: The SOAR system automatically isolates the infected endpoint from the network, collects relevant data (logs, file samples), and runs further analysis. It then triggers an alert to the security team with a detailed report of the incident.
3. **User Account Compromise**:
   - **Scenario**: Suspicious activity is detected on a user's account.
   - **SOAR Solution**: The SOAR platform automatically resets the user's password, logs the user out of all sessions, and notifies the user and the security team. It also triggers an investigation into the root cause of the compromise.

1.2. Introduction to XSOAR

XSOAR stands for Extended Security Orchestration, Automation, and Response. It is a product offered by Palo Alto Networks, designed to extend the capabilities of traditional SOAR platforms. XSOAR integrates with a wide range of security tools and provides advanced orchestration, automation, and response capabilities, along with additional features such as threat intelligence and case management.

Editions of XSOAR

1. **Community Edition**:
   - A free version with limited features designed for individual users or small teams to get started with SOAR capabilities.

2. **Enterprise Edition**:
    - A comprehensive version with full features, including advanced automation, extensive integrations, and enterprise-grade support, aimed at larger organizations with complex security needs.

2. XSOAR Installation and Setup

Key Points Summary

- Register for a free license.
- Install XSOAR server in a virtual machine and apply the free license.
- UI and configuration overview

2.1. XSOAR Installation

System requirements

Refer to the link below to see system requirements for production and development environments

- https://docs-cortex.paloaltonetworks.com/r/Cortex-XSOAR/6.5/Cortex-XSOAR-Administrator-Guide/System-Requirements

For learning purposes, the following requirements are used.

OS	Ubuntu Server 20.04
Processors	2
RAM	4 GB
Hard Disk	30 GB

If you are going to use this space for a long time, it is advisable to add more disk space.

Software used

- **Virtualization:** VMware Workstation 17 Player
    - https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html
- **SSH Client:** MobaXterm
    - https://mobaxterm.mobatek.net/download.html

Installation steps

1. Register and download the demisto-server.sh and DemistoLicense.lic using the link below.
   a. Link: https://start.paloaltonetworks.com/sign-up-for-community-edition-cxo.html
   b. You should receive the license file through the email you used for registering.
   c. Optionally, copy the download link for later use with wget command.
2. Download Ubuntu 20.04 ISO Image.
   a. Link: https://ubuntu.com/download/alternative-downloads
3. Create a Virtual Machine and install Ubuntu 20.04.
   a. Optionally, update system.

```
sudo apt-get update && sudo apt-get upgrade -y
```

4. Connect to the VM using SSH and transfer the installer file (demisto-server.sh).

```
wget -O demisto.sh “https://download.demisto.com/download-params?token=x
```

5. Convert the installer to an executable file.

```
chmod +x demisto-server.sh
```

6. Execute the installer.

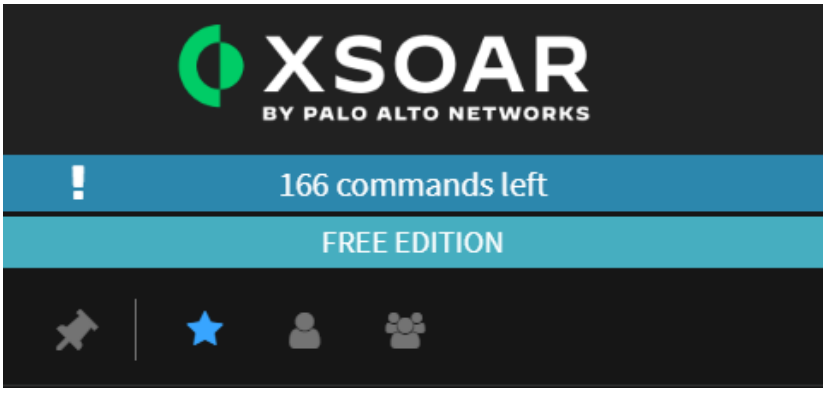
```
sudo ./demisto-server.sh
```

7. Accept the EULA and complete the information.
   a. The Server HTTPS port (default is 443).
   b. Not using Elasticsearch.
   c. Admin user (default is admin).
   d. Type the password (default is admin).

8. Confirm the XSOAR server and Docker are up and running.

```
systemctl status demisto
systemctl status docker
```

9. Optionally, check logs in /var/log/demisto.
10. Finally, install the free license, access https://serverURL:port.
    a. Go to Settings → About → License.
    b. Click Upload license and select the .lic file.



❓ **What happens at the end of the 30-day trial?**
When your trial period is about to expire, a message will pop up at the top-left of your screen telling you that your licence will expire in x days. When your trial period is done, your license will be rolled down to a limited version.

- 166 daily automation commands
- Rolling 30-day incident history
- 5 active feeds/100 indicators per feed
- Incident closure report

❓ **I did not receive my welcome email with the download link. When can I get it?**
Once the request is received, the Cortex XSOAR Product team validates that the requestor should have access to the Community Edition. Once this is approved, emails will be automatically sent out with directions for downloading and installing the Community Edition.

❓ **Who should I reach out to when I have technical questions?**
You can join the Palo Alto Networks LIVE Community forum where you can pose your questions to Cortex XSOAR experts.

https://live.paloaltonetworks.com/t5/cortex-xsoar/ct-p/Cortex_XSOAR

2.2. XSOAR UI and Configuration

- User Account Configuration
  - Change name, password email, profile picture...
- Access Management
  - Invite users
  - Create and assign Roles
- Password Policy
  - Adjust password complexity requirements
- Server Troubleshooting
  - Download log bundle
- System Diagnostics
  - Server performance and metrics

3. XSOAR Concepts

3.1. Incidents, types and fields

Key Points Summary

- Concepts: Incidents, Incident Types, Incident Fields and Custom Fields and its configuration.
- Incident Layout Overview.
- Best practices for Incident Fields.

- **Incidents**: These are the primary entities in Cortex XSOAR that represent events or issues that require investigation and resolution. They can be created:

    - Integrations: From alerts generated by various security tools (SIEM, EDR..).
    - Manually.
    - API.

- **Incident Types**: These define different categories or classes of incidents, such as phishing, malware, DDoS attacks, etc. Each type can have specific fields, workflows, and automations associated with it.

- **Incident Fields**: These are data attributes associated with an incident, such as incident ID, severity, status, description, reporter, timestamps, and custom fields specific to the incident type.

- **Incident Layouts**: These define the user interface layout for displaying incident details within Cortex XSOAR. Layouts can be customized to show relevant information for different incident types, helping analysts quickly access the most pertinent data during their investigation.

Each incident in Cortex XSOAR is organized into several sections.

- **Incident Info**: This section provides an overview of the incident, displaying key details and metadata.
- **War Room**: This section is a collaborative space where analysts can discuss, investigate, execute commands and respond to the incident.
- **Work Plan**: This section outlines the automated and manual tasks required to handle the incident, typically driven by a playbook.
- **Evidence Board**: This section is used to gather and organize evidence related to the incident. It helps in structuring the investigation and supports post-incident analysis.
- **Related Incidents**: This section displays other incidents that might be related to the current one, helping analysts identify patterns and correlations.
- **Canvas**: This section provides a visual workspace where analysts can map out the incident and its related components.

👍 **Best practices**

- **Use Built-in Fields When Possible**: Before creating a custom field, check if there is a built-in field that can serve your purpose. Built-in fields are optimized for performance and integration within the system.
- **Keep It Simple**: Only create custom fields when absolutely necessary. Too many fields can clutter your incident forms and make them harder to use.
- **Consistent Naming Conventions**: Use clear and consistent naming conventions for custom fields to ensure they are easily understood by all team members.

📒 **Summary**

- **Incidents** are the main entities representing security events.
- **Incident Types** categorize incidents and define their structure and automations.
- **Fields**, both built in and **custom**, capture and organize the data associated with incidents and incident types. Fields can be associated with one or several incident types.

3.2. Integrations and Instances

Key Points Summary

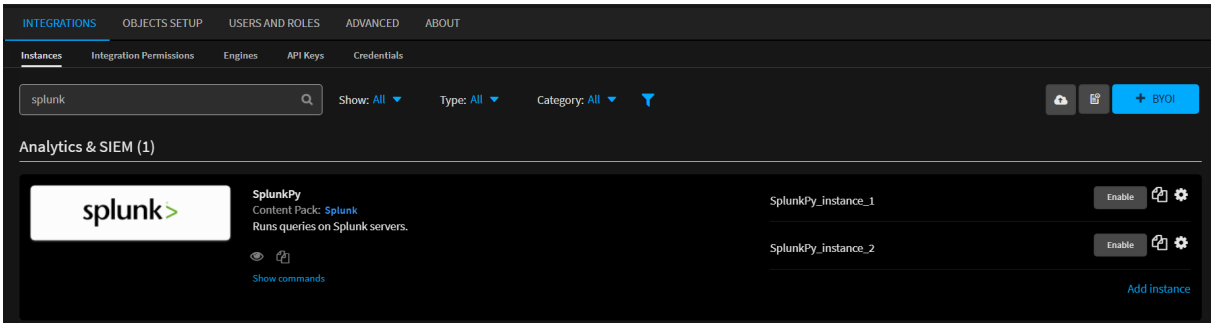
- Concepts: Integrations, Instances, Marketplace.
- DEMO: Install an Integration from the Marketplace.
- DEMO: Create an Integration Instance.

- **Integrations**: These are connectors that allow Cortex XSOAR to interact with external security tools and services (e.g., SIEMs, firewalls, threat intelligence platforms). Integrations facilitate data ingestion and enable automated actions on these tools.
  - **Categories:** Analytics and SIEM, Authentication, Case Management, Data Enrichment, Threat Intelligence, Database, Endpoint, Forensics and Malware Analysis, IT Services, Messaging, Network Security, Vulnerability Management.

- **Integration Instances**: These are specific configurations of integrations, including connection details, credentials, and other settings necessary for Cortex XSOAR to communicate with the integrated tool or service. You can have multiple instances of an integration to connect to different environments or tenants.

The Marketplace in Cortex XSOAR is a centralized platform where users can find, download, and install various integrations, automations, content packs, and other useful tools that enhance the functionality of Cortex XSOAR. Think of it as an app store specifically for security operations and automation.

Example



In case you are using the “Sample Incident Generator”, remember to disable it in order to avoid being flooded with incidents and hence, running out of daily free commands.

3.3. Classifiers and Mappers

Key Points Summary

- Concepts: Classifiers and Mappers.
- DEMO: Create a Classifier and a Mapper, assign it to an integration instance and test the results.

- **Classifiers**: These determines the type of incident that is created from a specific integration. When an incident is ingested from an external source, the classifier determines the incident type based on the fields and values present in the incoming data.

- **Mappers**: These map fields from external systems to Cortex XSOAR incident fields, ensuring that data is properly translated and stored within the platform. Mappers work alongside classifiers to standardize data from different sources.
    - Any fields that you do not map, are automatically mapped to XSOAR labels. While this information can still be accessed, it is always easier to work with fields.

Example

Splunk Alert Field	XSOAR Field
_raw	Raw Event
error	Error Message
app_id	Application ID

📒 **Summary**

- **Classifiers** determines the incident type from incoming data.
- **Mappers** map external fields to XSOAR built in or custom fields.
- Both can be assigned to a specific integration instance.

❗ In case you are using the “Sample Incident Generator”, remember to disable it in order to avoid being flooded with incidents and hence, running out of daily free commands.

3.4. Playbooks and Context

Key Points Summary

- Concepts: Playbook, Context, Inputs, Outputs, Playbook Debugger, Quiet Mode.
- Playbook designer canvas overview.
- DEMO: Create a simple Playbook that generates a random password and print the results.

Playbooks are seen more in depth in the Playbook Development module.

- **Playbooks**: These are automated workflows that guide the incident response process in Cortex XSOAR. Playbooks can include tasks, decision points, integrations, and automations to standardize and streamline how incidents are handled.

- **Context:** The context in Cortex XSOAR refers to the data that is shared and accessible across different tasks within a playbook and between playbooks. It acts as a shared storage where information is saved and retrieved as the playbook executes. Context data helps in maintaining the state of the incident and facilitates data sharing between different parts of the workflow. Context is presented as JSON format.

- **Inputs:** These are the parameters or data required by a playbook or a specific task within a playbook to execute. Inputs can come from various sources such as:
  - **Incident Fields**: Data fields associated with the incident, like the incident type, severity, or custom fields.
  - **Previous Tasks**: Outputs from preceding tasks in the playbook can serve as inputs for subsequent tasks.
  - **Manual Inputs**: Data provided manually by analysts when triggering a playbook or during playbook execution, often through prompts or forms.

- **Outputs:** These are the results produced by a playbook or specific task within a playbook. Outputs can be used in several ways:
  - **Context Data**: Outputs can be stored in the context to be used later in the playbook or by other playbooks.

- **Incident Fields**: Outputs can update incident fields, providing new or updated information about the incident.
- **Task Results**: Outputs can directly influence the flow of the playbook, determining the next steps or decisions in the workflow.

How Context, Inputs, and Outputs work together

1. **Playbook Initialization**: When a playbook starts, it may receive initial inputs from the incident fields, manually provided data, or context data from other playbooks or tasks.
2. **Task Execution**: Each task within the playbook will use the provided inputs to perform its designated function. The task may involve running a script, executing a command, querying an integrated system, or waiting for human input.
3. **Generating Outputs**: After executing, the task will produce outputs based on its actions. These outputs can include results of data queries, analysis findings, or any other relevant information.
4. **Storing in Context**: The outputs are stored in the context under specific context keys and paths. This makes the data available for subsequent tasks in the playbook or for use in other playbooks.

❗ When you enable Quiet Mode, War Room entries are not created and inputs and outputs are not stored in the Work Plan. Quiet Mode improves performance by increasing playbook speed and saving database size (recommended in production environments). Turn it off when you are debugging.

3.5. Automation Scripts, Commands and CLI

Key Points Summary

- Concepts: Automation Scripts, Built in , integration and reputation Commands, Playground, XSOAR CLI, Cortex XSOAR IDE.
- DEMO: Create a simple Automation Script that checks password complexity.
- DEMO: Explore the Playground and test commands.

💡 Automations are seen more in depth in the Automation Development module.

- **Automations**: These are scripts or actions that can be executed automatically as part of a playbook or manually by analysts. Automations can perform a wide range of tasks, such as data enrichment, notification sending, or executing commands on integrated systems.

- **Playground**: This is a safe, isolated environment within Cortex XSOAR where analysts can test commands, integrations, and scripts without affecting real incidents or production data.

- **CLI (Command Line Interface)**: Cortex XSOAR provides a CLI that allows users to execute commands and interact with the platform directly, often used for troubleshooting, testing, and scripting.
  - `!` Integration commands, automations, and built-in commands. For example, add evidence, assign an analyst, etc.

    ```
    !Print value="Hello World!"
    ```

  - `/` System commands /operations. For example, add notes, close an investigation, etc.

    ```
    /clear_playground
    ```

  - `@` User tagging. Send notifications to administrators, teams, analysts, etc.

    ```
    @admin (Admin) Hello!
    ```

- **Built-in Commands:** Cortex XSOAR comes with a wide range of built-in commands that you can use out-of-the-box. These commands cover a variety of functions like querying threat intelligence sources, interacting with ticketing systems, and performing data enrichment. Built-in commands save you time by providing ready-made solutions for common tasks, allowing you to focus on creating more strategic automation.

```
!GeneratePassword min_digits=5
```

- **Reputation Commands:** Reputation Commands are a subset of built-in commands specifically designed to fetch reputation data from various threat intelligence sources. For example, you might use a reputation command to check the threat level of an IP address or a domain.

```
!ip ip=8.8.8.8
```

IsPasswordValid

```
def main():
    try:
        # Get the input password
        password = demisto.args().get('password')

        # Define password complexity requirements
        requirements = {
            "length": len(password) >= 8,
            "uppercase": any(char.isupper() for char in password),
            "lowercase": any(char.islower() for char in password),
            "digit": any(char.isdigit() for char in password),
            "special": any(char in "!@#$%^&*()-_+=" for char in password
        }

            # Check if password meets all requirements
        is_valid = all(requirements.values())

        # Return results
        results = CommandResults(
            outputs_prefix='isValid',
            outputs=is_valid,
            )
        return_results(results)
    except Exception as e:
        return_error(f'Failed to execute Script. Error: {repr(e)}')

if __name__ in ('__main__', '__builtin__', 'builtins'):
    main()
```

```
!isPasswordValid password=1234 #False
!isPasswordValid password=1At$P4ss09 #True
```

3.6. Lists

Key Points Summary

- Concepts: Lists.
- DEMO: Create and use a lists to enrich your data.

- Lists: A list is a data container where you can globally store data within Cortex XSOAR. Despite the name lists, the data can be in any text format, and can even store files or an HTML template that you can define to use as part of a Communication task. Use case examples:
  - A list of approved IP addresses that are allowed to access your network (whitelist).
  - A list of malicious domains that should be blocked (blacklist).
  - A list of file extensions that you want to exclude in an automation (filter).
  - A list (JSON format) to map your customers name to an ID (mapper + enrichment).

Sample Data

```
{
    "192.168.1.1": "VM1",
    "192.168.1.2": "VM2",
    "192.168.1.3": "VM3"
}
```

List Commands

XSOAR provides a set of built-in commands to interact with lists:

Command	Description
`getList`	Retrieves the contents of the specified list.
`createList`	Creates a list with the supplied data.
`addToList`	Appends the supplied items to the specified list.
`setList`	Adds the supplied data to the specified list and overwrites existing list data.
`removeFromList`	Removes a single item from the specified list.

Accessing list content

In a playbook, you can access the data in a list via the context button under Lists, or by using the path `${lists.<list_name>}`.

```
# Access the list's content from an Automation Script
 my_list = demisto.executeCommand("getList", {"listName": list_name})[0]
```

HostnameLookup

```
def main():
    try:
        # Get the IP Address given by the user
        ip = demisto.args().get("ip")

        # Get lists content
        contents = demisto.executeCommand("getList", {"listName": "IP to
        contents_dict = json.loads(contents)

        # Retrieve the hostname
        hostname = contents_dict.get(ip, "Not found")

        # Return results
        results = CommandResults(
            outputs_prefix='hostname',
            outputs=hostname,
        )

        return_results(results)
    except Exception as e:
        return_error(f'Failed to execute Script. Error: {str(e)}')

if __name__ in ('__main__', '__builtin__', 'builtins'):
    main()
```

3.7. Threat Intelligence

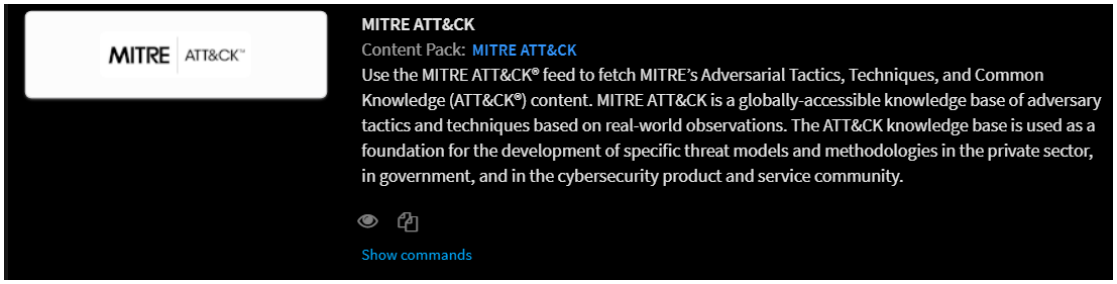
Key Points Summary

- Concepts: Indicators, Indicator Types, Threat Intelligence Integrations, Indicator Exclusion List, Indicator Extraction Rules.
- DEMO: Configure an Indicator Extraction Rule to auto extract and enrich indicators automatically.
- DEMO: Configure Indicator extraction and enrichment in a playbook task.

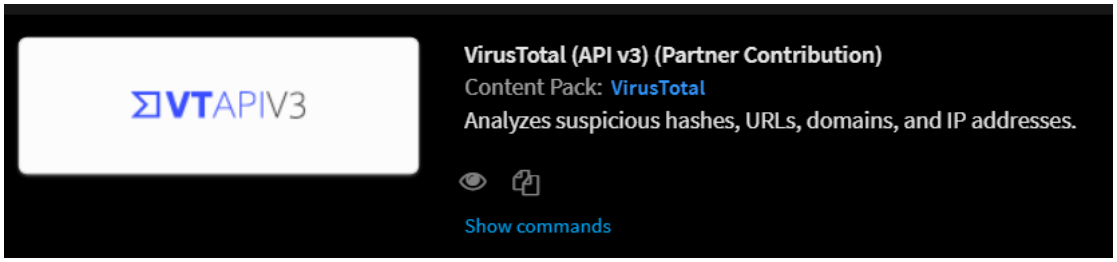
- **Threat Intelligence Indicators (TI Indicators)**: These are pieces of data related to potential security threats, such as IP addresses, domains, file hashes, and URLs. Cortex XSOAR can ingest and manage TI indicators to enrich incidents and support threat hunting and detection activities.
  - They are extracted using regex (pattern determined by Indicator Type).
  - They are enriched by running reputations commands (manually or determined by Indicator Type).

- **Indicator Types:** Indicator Types categorize the various forms of indicators based on their nature and usage. The following is a list of some of the indicator types:
  - IP Address
  - Domain
  - URL
  - File
  - Email

- **Threat Intelligence Integrations:** Threat Intelligence Integrations are the connections to various threat intelligence sources that provide data on current threats.
  - **Data Enrichment & Threat Intelligence:** Data enrichment refers to the process of enhancing raw data by adding relevant information from additional sources. It involves integrating and augmenting the initial data set with more context or details.
    - Example: MITRE ATT&CK Integration.



- **Threat Intel Feeds:** Threat intel feeds are streams of data provided by various sources that contain information about known and emerging threats. These feeds offer a continuous supply of indicators and threat-related information.
  - Example: Virus Total Integration.



- **Indicator Extraction Rules:** Indicator Extraction Rules define how indicators are identified and extracted. These rules specify patterns (regular expressions) and formats to look for, ensuring that relevant indicators are automatically detected and processed. After extraction, the indicator can be enriched using commands (such as the !whois command). These can be applied to Incident Types.

  Indicator extraction supports the following modes:

Mode	Description
None	Indicators are not extracted automatically.
Inline	Indicators are extracted synchronously within the context. Extraction occurs before playbook tasks run, ensuring robust information per indicator. This may delay playbook execution as extraction and enrichment happen in real-time and are placed into the incident context.
Out of band	Indicators are extracted asynchronously, running in parallel to other actions. Extracted data is not immediately available for task inputs or outputs. Used when indicators are not needed for immediate playbook flow, potentially improving system performance. Extracted indicators do not appear in the context.
System default	Uses default extraction modes based on different scenarios: - Incident creation: Default is Inline, extracting from all fields at creation. - Incident field change: Default is Out of band, extracting asynchronously.

Mode	Description
	- Tasks: Default is None, no automatic extraction. - Manual: Default is Out of band, extraction from CLI commands.

Indicator built-in commands

```
What is 8.8.8.8?
!extractIndicators text="ip: 8.8.8.8" auto-extract=inline
!enrichIndicators indicatorsValues=8.8.8.8
```


Reputation commands, such as `!ip` and `!domain`, can only be used after you configure and enable a reputation integration instance, such as Virus Total and Whois.

3.8. Jobs

Key Points Summary

- Concepts: Jobs (Time triggered and Triggered by delta in feed).
- DEMO: Create a time triggered job to delete expired threat intelligence indicators daily.

- **Jobs**: These are scheduled tasks within XSOAR that enables you to run playbooks that run at defined intervals or based on certain events. There are two types:
  - **Time triggered:** These jobs run at specific times. For instance, you can schedule a job to run every night to remove expired indicators from your system.
  - **Triggered by delta in feed:** These jobs are event-driven. They run when there are changes to a feed. For example, you can set up a job to run a playbook whenever a Threat Intelligence Management (TIM) feed finishes fetching new indicators.

Jobs Use Cases

- Daily or monthly security reports.
- Scheduled monitoring of VPN connectivity.
- Proactive threat investigations using uploaded STIX files containing IOCs.

- Monitoring and renewal of expiring SSL certificates.
- Scanning for security vulnerabilities in applications.
- Validation of adherence to policies and regulations.
- Continuous assessment of security system performance.
- Managing access for privileged users, including onboarding and offboarding.
- Auditing the security configurations of endpoints.
- Centralizing and analysing system logs for security insights.

3.9. How to search in XSOAR

Key Points Summary

- Concepts: Search syntax.
- DEMO: Search for data in XSOAR.

! The search follows the Bleve query syntax

Search Query Examples

```
# Search for active incidents with Information severity
status:Active and severity:Informational
```

```
# Search for active incidents or Informational incidents
status:Active or severity:Informational
```

```
# Search for all incidents that are not closed and are not in the job ca
-status:closed -category:job
```

```
# Search playbooks whose name starts by Block
name:Block*
```

```
# Search for automations whose code contains "import json"
script:*import json*
```

```
# Search for incidents that were created on or after July 1, 2024
created:>="2024-07-01T00:00:00 +0200"
```

```
# Search for indicator values that contain "www" and end with ".com" usi
value:"/w{3}..*.com/"
```

Search Command Examples

```
# Search for incident id 523
/search incident.id:523
```

```
# Search indicators that ends with ".com"
/search indicator.value:*.com
```

3.10. Marketplace and Content Packs

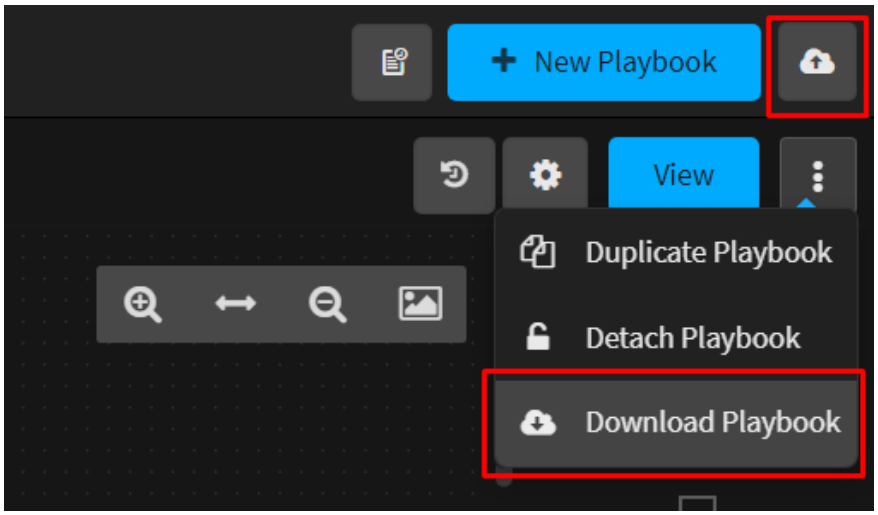
Key Points Summary

- Concepts: XSOAR Marketplace and Content Packs.
- DEMO: Explore the Marketplace.
- DEMO: Create your own Content Pack to import it in another XSOAR environment.

- **Marketplace**: In XSOAR, the marketplace refers to a centralized location where users can discover, download, and manage integrations, playbooks, automations, and other extensions for the XSOAR platform. It acts as a repository of pre-built content that enhances the functionality and capabilities of XSOAR.

- **Content Packs**: Content packs in XSOAR are bundles of integrations, playbooks, incident types, layouts, scripts, and other configurations that are packaged together for specific security or IT operations use cases. They are designed to simplify deployment and configuration of commonly used workflows and automation within the XSOAR platform. Content packs can be created by Palo Alto Networks (the company behind XSOAR), third-party developers, or by organizations themselves to meet their specific needs.

💡 Playbooks and Automation can be individually exported as a `yml` file and imported using the upload button.



4. Playbook Development

4.1. Tasks Types

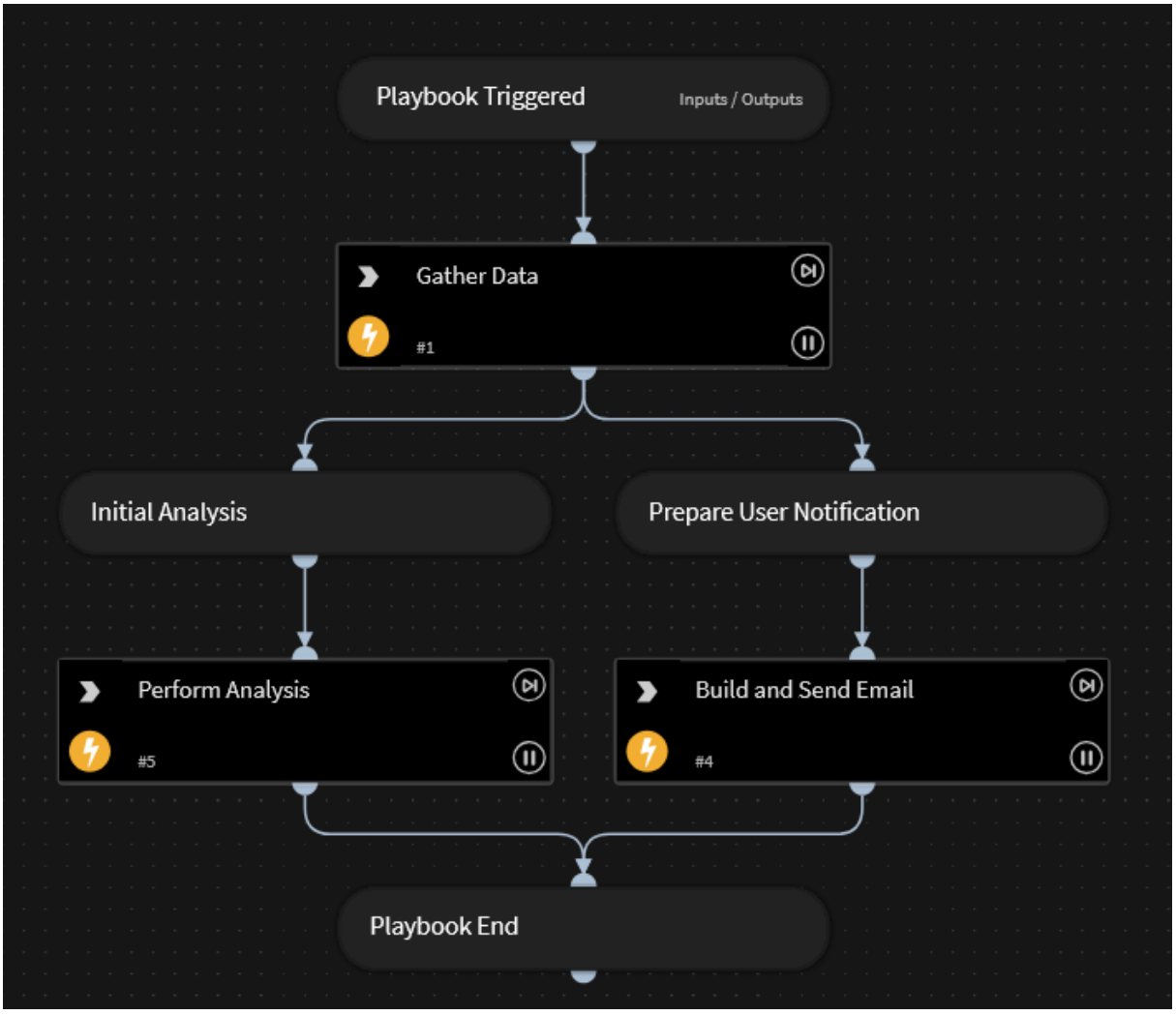
Key Points Summary

- Concepts:
  - Section Headers
  - Manual Tasks
  - Automated Tasks
  - Conditional Tasks
  - Data Collection Tasks
- DEMO: Create and test different task types.

Section Headers

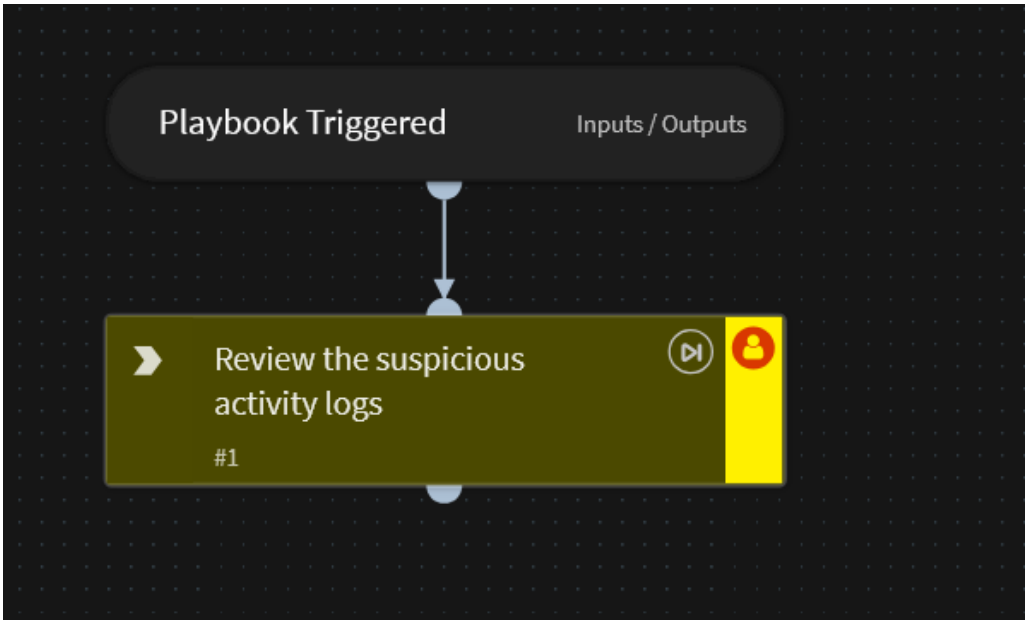
- Section Headers are used to organize and visually separate different parts of a playbook. They act as markers or dividers to group related tasks together.
- These headers do not perform any actions but are useful for making playbooks more readable and easier to manage, especially in complex workflows.

- Example: You might have a Section Header titled "Initial Analysis" followed by tasks related to data collection and analysis, and another titled "Remediation" for tasks related to mitigating threats.



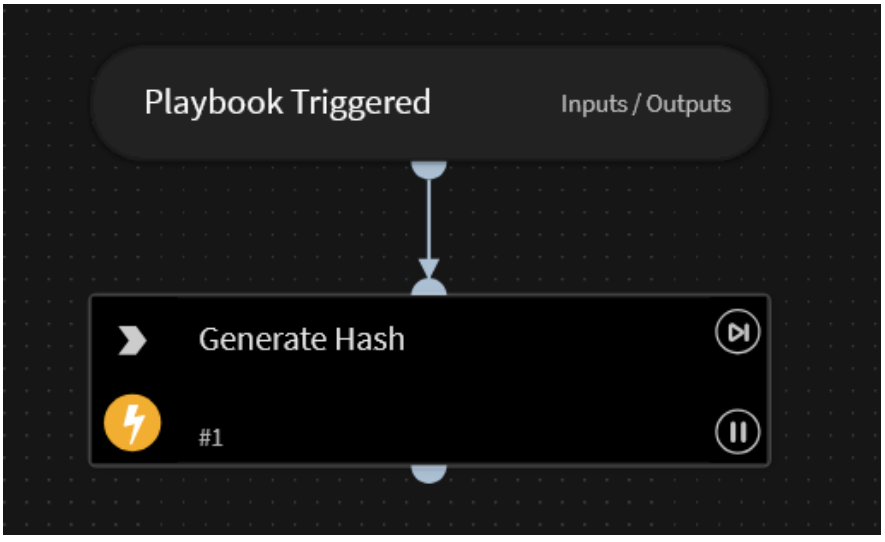
Manual Tasks

- Manual Tasks require human intervention to complete. These are often used when a task needs a decision or input from an analyst.
- They typically include instructions for the analyst on what needs to be done, and the playbook will pause execution until the task is marked complete.
- Example: A task might ask an analyst to review suspicious activity logs and decide whether to escalate the incident.



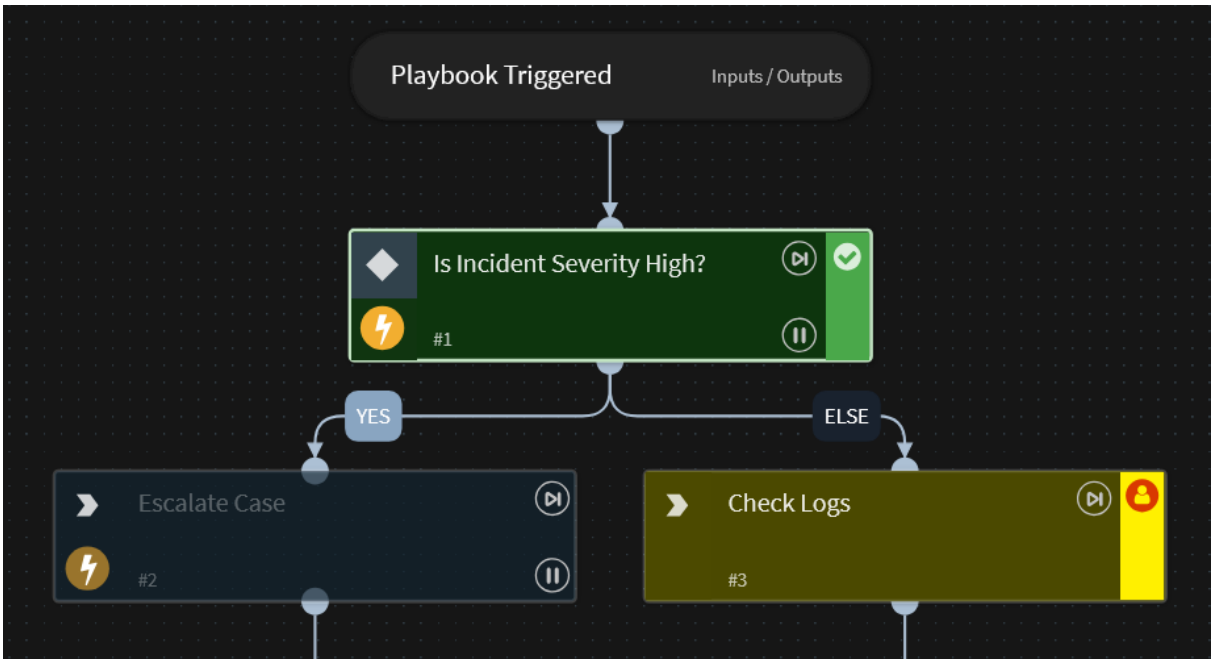
Automated Tasks

- Automated Tasks are actions that the playbook can perform without human intervention. These tasks execute predefined commands, scripts, or integrations with other systems.
- They are essential for automating repetitive and time-consuming tasks, enabling faster response times and reducing the burden on analysts.
- Example: Automatically querying an endpoint for its status, updating a ticket in a ticketing system, or running a script to block an IP address.



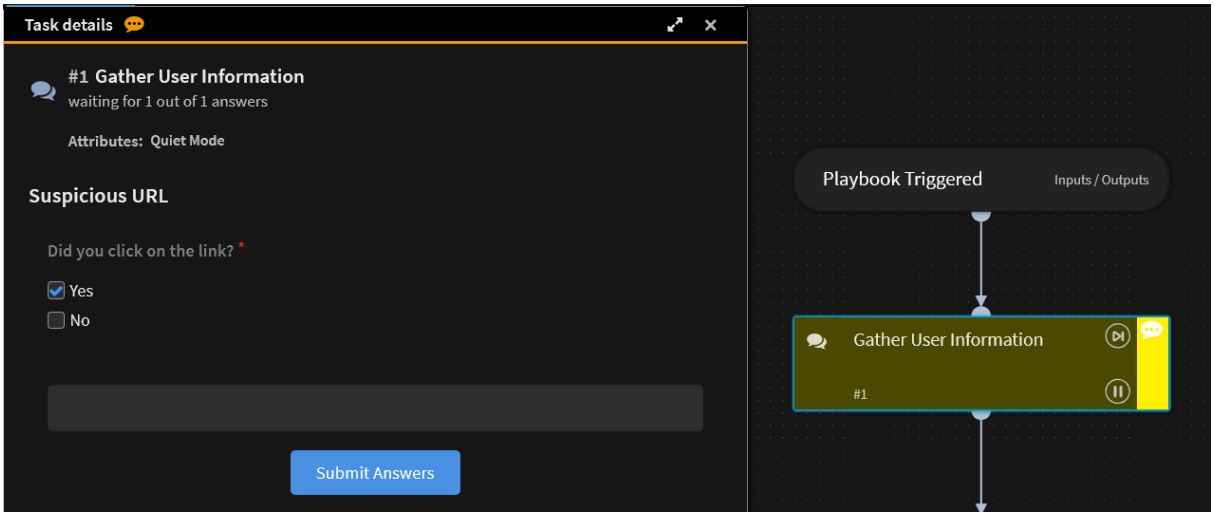
Conditional Tasks

- Conditional Tasks introduce branching logic into the playbook. They allow the workflow to take different paths based on the evaluation of certain conditions or criteria.
- This type of task uses "if-else" logic to decide the next steps in the playbook based on variables, outputs of previous tasks, or other criteria.
- Example: If the severity of an incident is high, the playbook might proceed to an immediate containment task; if it is low, it might move to a monitoring phase.



Data Collection Tasks

- Data Collection Tasks are designed to gather information from various sources, which can then be used in subsequent tasks within the playbook.
- These tasks can include querying databases, retrieving logs, or collecting input from users.
- Example: Collecting information about a potentially compromised endpoint, such as running processes, network connections, and system configurations.



4.2. Sub-Playbooks

Key Points Summary

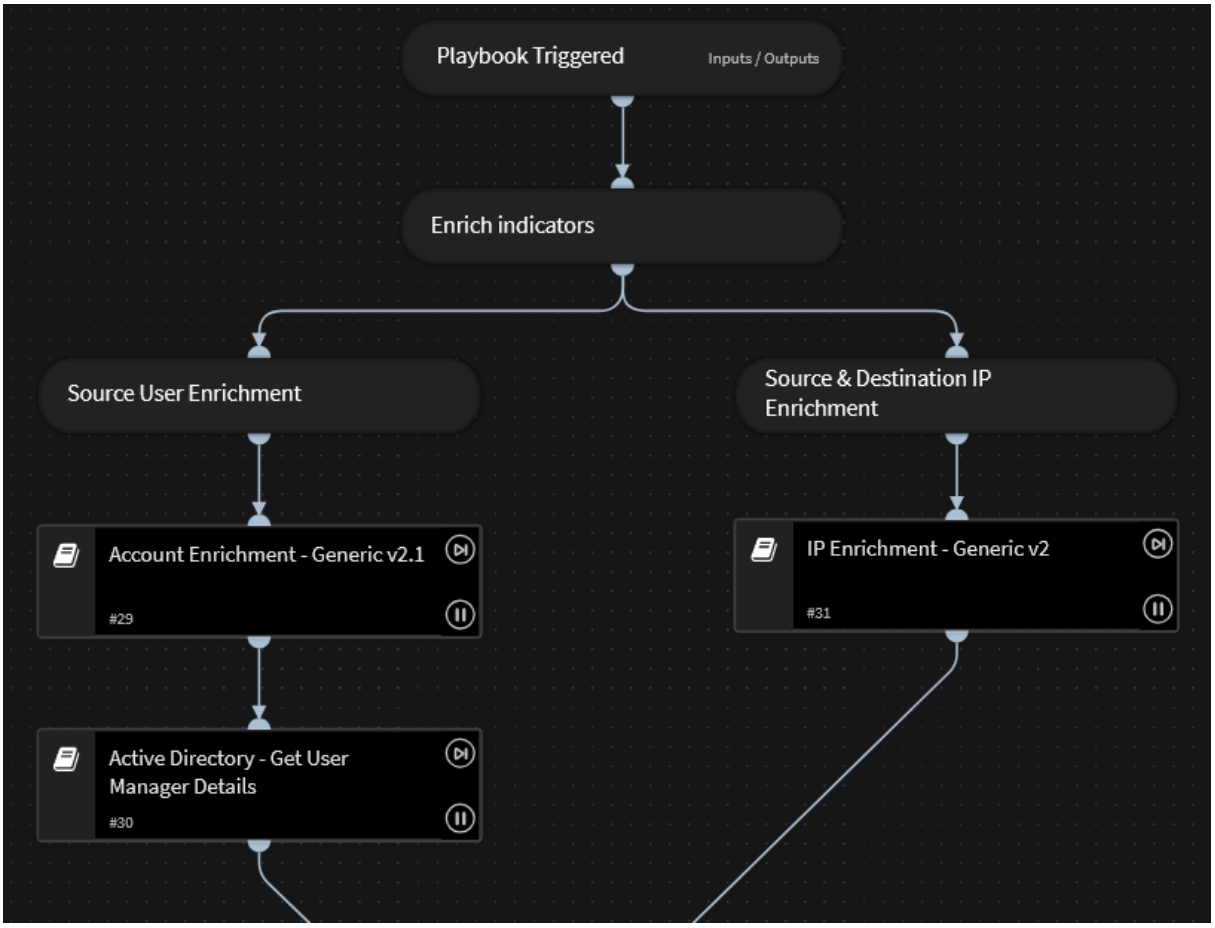
- Concepts: Sub-Playbooks, Parent Playbooks, Playbook Inputs and Playbook outputs.
- DEMO: Create a Sub-Playbook for checking the reputation of an IP Address and use it in a parent Playbook.

- **Sub-Playbooks:** These are smaller, modular playbooks that are designed to execute a specific set of tasks within a larger playbook. They help in organizing and managing complex workflows by breaking them down into more manageable parts. Sub-Playbooks can be reused across different Parent Playbooks, promoting modularity and reducing redundancy.

- **Parent Playbooks:** These are the main playbooks that orchestrate the entire workflow, often calling multiple Sub-Playbooks to perform various tasks. They are the top-level playbooks that define the overall logic and sequence of actions to be performed in response to an event or incident.

- **Playbook Inputs and Outputs:** Playbooks, similar to automations and integration commands, can receive inputs. These inputs are utilized within the sub-playbook and its tasks to carry out specific functions. Sub-playbooks produce outputs, which are then stored in the context data of the parent playbook.

- You can access playbook inputs values with `${inputs.<input_name>}`.



💡 It is recommended to have the context private to the sub-playbook so that only the outputs that we define will be returned to the parent playbook.

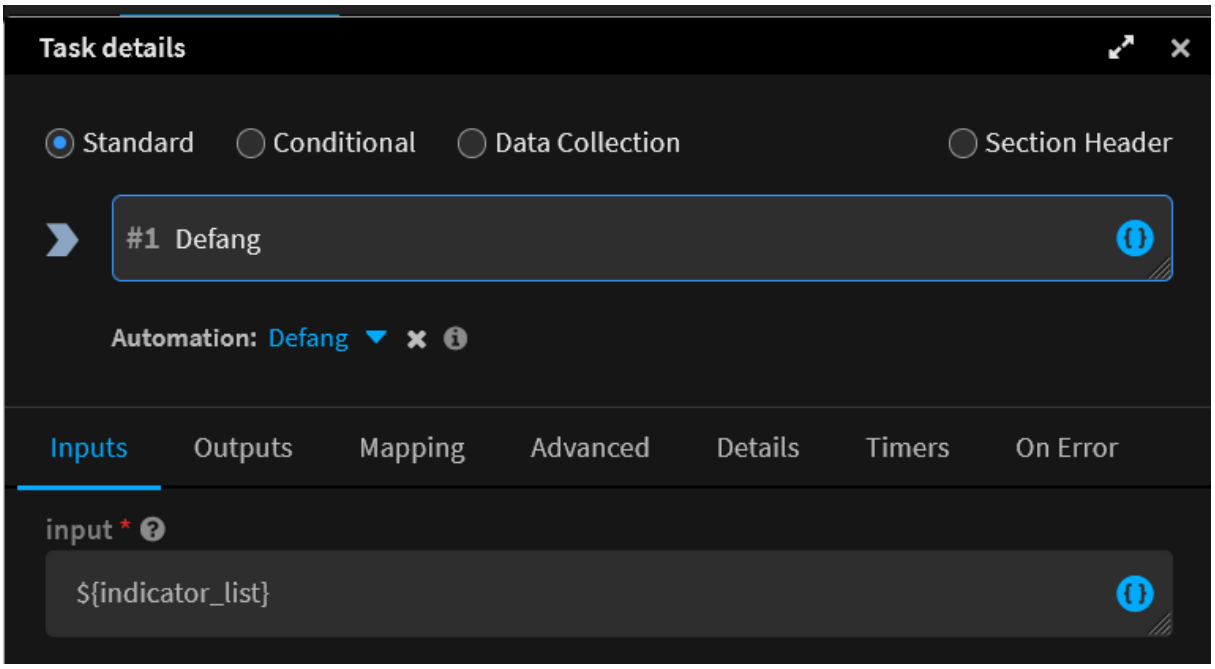
4.3. Loops

Key Points Summary

- Concepts: Loops in Playbook Tasks and Looping with Sub-Playbooks.
- DEMO: Defang multiple indicators using a task loop.
- DEMO: Check if hosts are UP or DOWN using a sub-playbook loop.

Loops in Playbook Tasks

- When an array of items is passed into an automation, XSOAR will automatically loop over the array and execute the task for each item in the array.

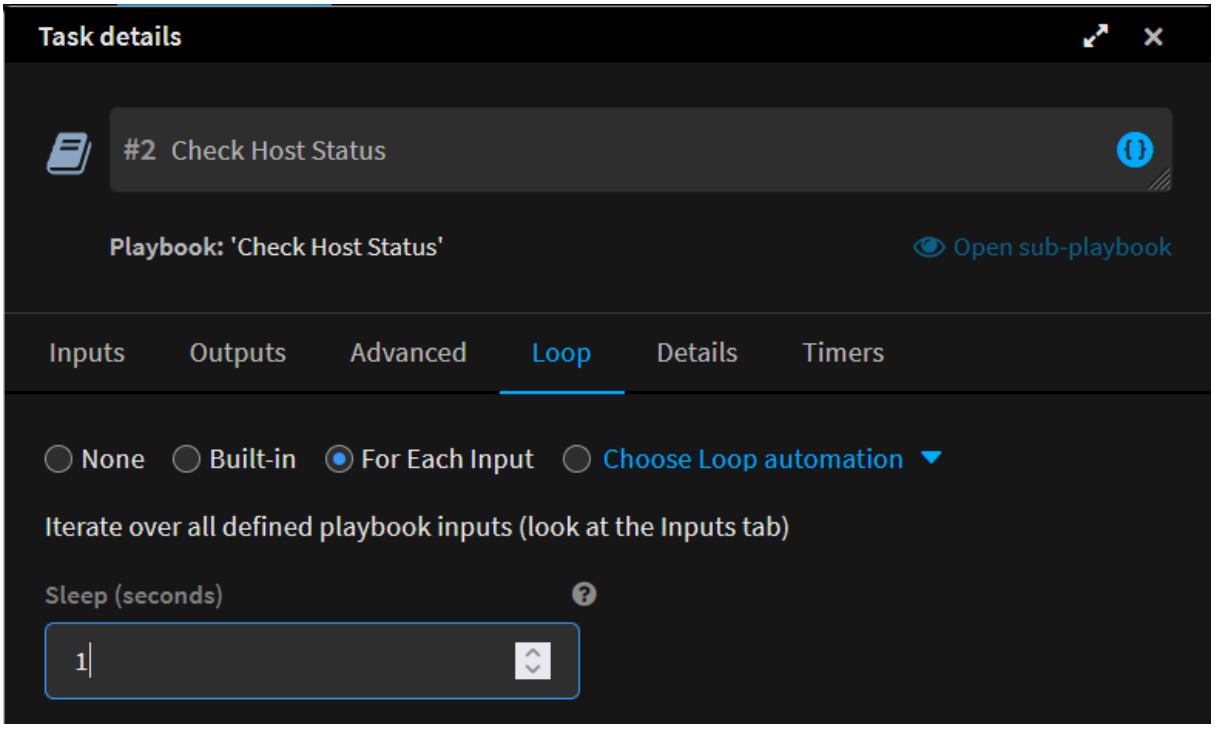


Looping with Sub-Playbooks

- **For Each Input:** The loop exits automatically, when the last item in the input is executed.
  - If there are multiple input lists with the same number of items, the sub-playbook runs once for each set of inputs.
  - If there are multiple input lists with different numbers of items, the sub-playbook runs the first set of inputs, followed by the second, third, etc.

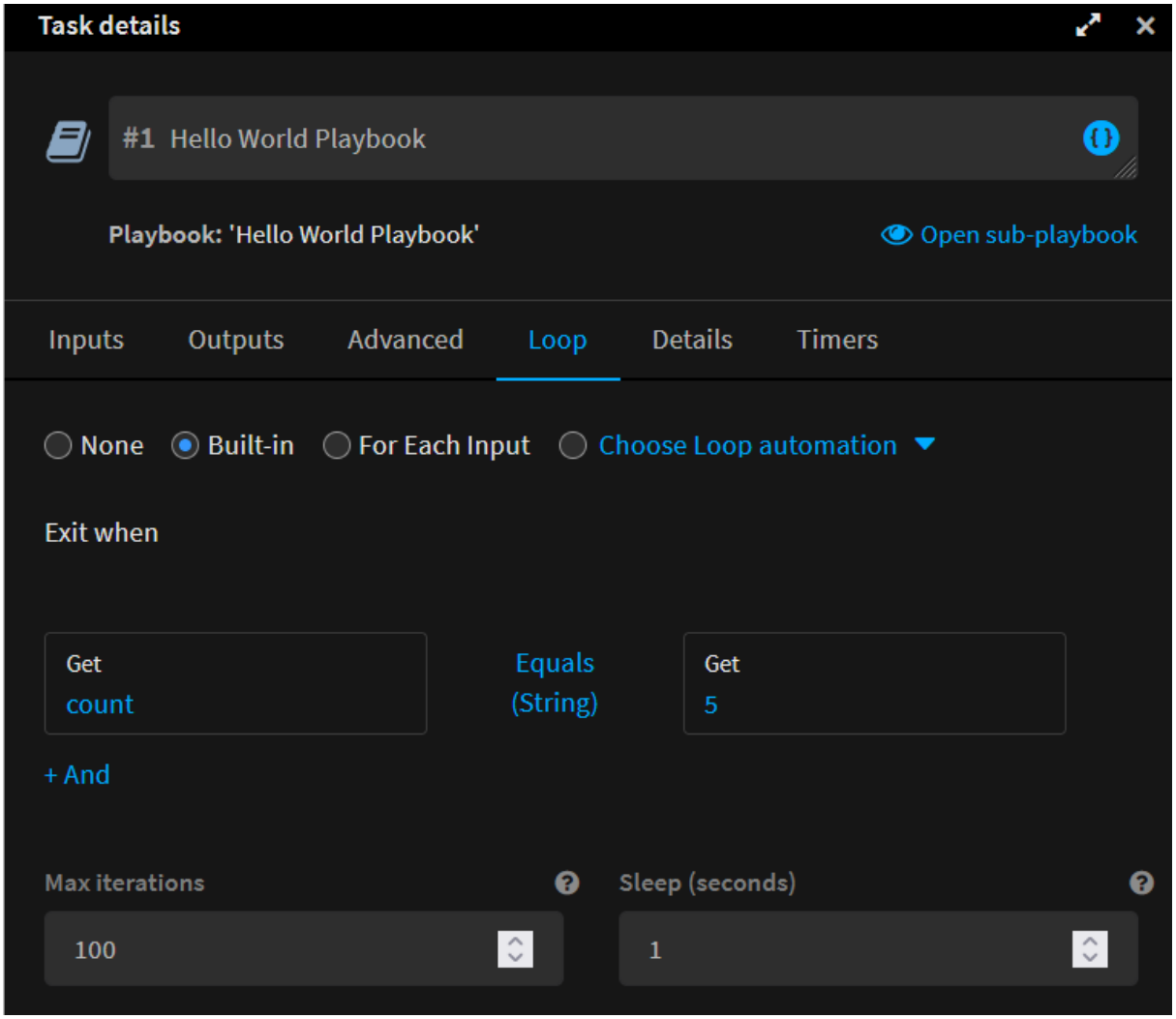
If you have two lists: `list1 = [1, 2]` and `list2 = [A, B]`, the sub-playbook will run twice: once with inputs `1` and `A`, and once with `2` and `B`.

If you have three lists: `list1 = [1, 2, 3]`, `list2 = [A, B]`, and `list3 = [X]`, the sub-playbook will first run with `1`, `A`, and `X`; then with `2` and `B`, and finally with `3`.



- **Built-in or Choose Loop Automation:** The loop exits based on a condition. The playbook does not loop through the inputs but takes the inputs as a whole.

> ! The default number of loops is 100. A high number may affect performance.



4.4. Filters and Transformers

Key Points Summary

- Concepts: Filters and Transformers.
- DEMO: Create a file extension filter.
- DEMO: Manipulating data with different transformers.

The data can be manipulated by using filters and transformers. You can add filters and transformers in a playbook task or when mapping an instance.

Filters

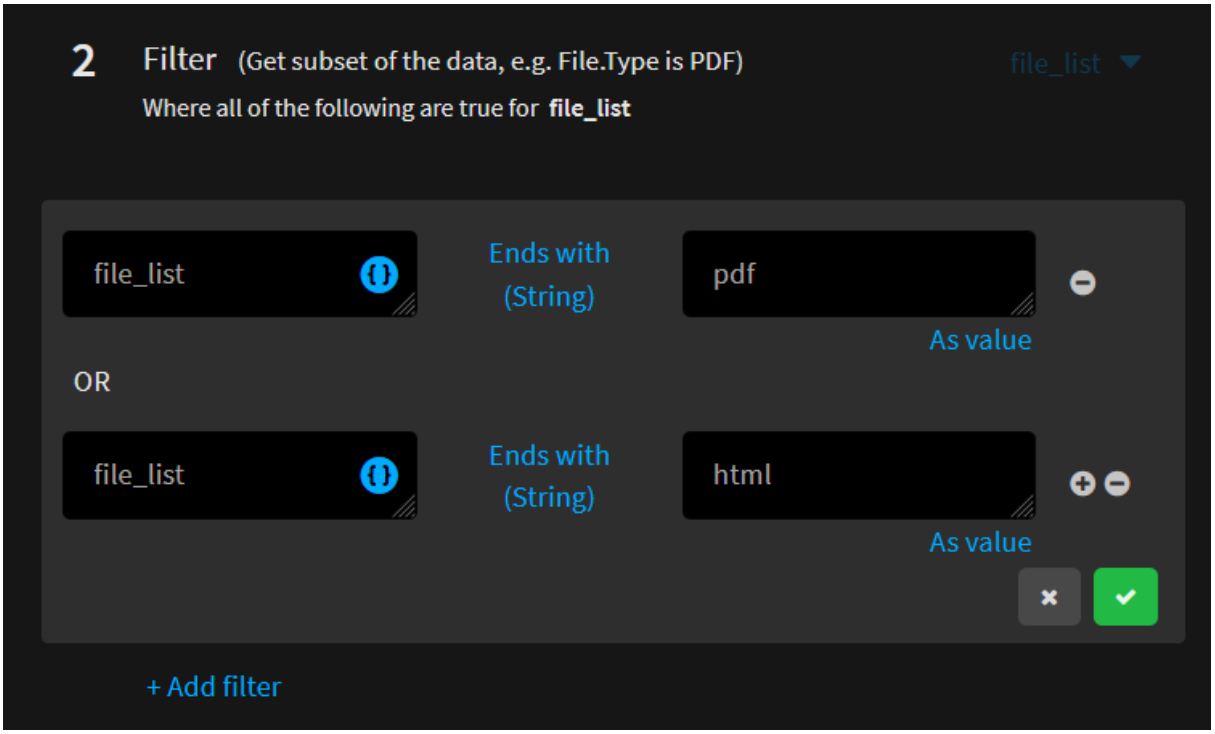
Filters are used to refine and limit the data based on specified conditions. They act as a gatekeeper, allowing only the data that meets the defined criteria to pass through. This is useful when you need to work with a subset of data or when you're looking to trigger actions based on specific data conditions.

Common Use Cases for Filters:

- Isolating incidents based on severity.
- Filtering out false positives.
- Extracting specific data points from a larger dataset.

Example Filter Conditions:

- **Field Equals**: `incident.severity == "High"`
- **Field Contains**: `incident.name contains "phishing"`
- **Field Greater Than**: `incident.time_to_resolve > 30`



Transformers

Transformers are used to manipulate and change data into a desired format or structure. They are applied to data to prepare it for further processing, presentation, or integration with other systems. Transformers modify the data content or structure without changing the original dataset.

Common Use Cases for Transformers:

- Converting timestamps to a readable format.
- Extracting specific information from a text field.
- Reformatting data for API calls or integrations.

Example Transformers:

- **Timestamp to Date**: Converts a Unix timestamp to a human-readable date format.
- **Extract Regex**: Extracts specific data from a string using a regular expression.
- **Lowercase**: Converts all characters in a string to lowercase.

🤖 **Some Useful Transformers**

- **GetField:** Retrieves a field from an object using the dot annotation.
- **Split & trim:** Splits a string into an array of strings and removes white spaces from both ends of the string.
- **ParseJSON:** Parse a given JSON string "value" to a representative object
- **SetIfEmpty:** Checks an object for an empty value and returns a pre-set value.
- **RegexExtractAll:** Extraction of all matches from a specified regular expression pattern from a provided string. Returns an array of results.
- **RegexReplace:** Format patterns matched with regex. If the regex does not match any pattern, the original value is returned.
- **RandomElementFromList:** Randomly select elements from a list in Python.
- **Count:** Counts number of elements.

4.5. Extend Context, Using argument and Quiet Mode

Key Points Summary

- Concepts: Extend Context, Using argument, Quiet Mode.
- DEMO: How to use Extend Context and the Using parameter to retrieve specific data from commands.
- DEMO: How to enable/disable quiet mode at task and playbook level.

Extend Context

Extend Context is a feature that allows you to control what data and how much data is returned to context. Extend Context can also be used when the same command is run multiple times in the same playbook, but the outputs need to be saved to different context keys.

When **Ignore outputs** is selected, the remaining outputs will not be stored in the context. Only the selected keys will remain. By reducing the context data size, the performance is also increased.

You can extend context either in a playbook task, or directly from the command line.

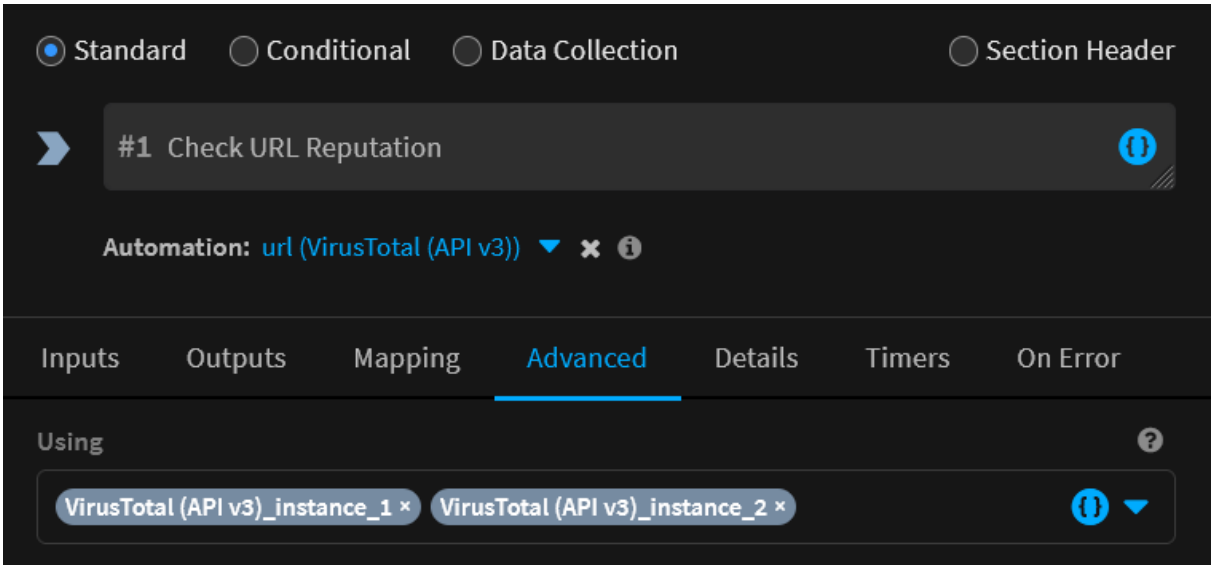
```
!<commandName> <argumentName> <value> extend-context=contextKey=JsonOutputPath ignore-outputs=true
```

> 💡 By default, when you run a command, either from the command line or as part of a script or playbook, a subset of JSON fields are returned. To display the full JSON response, run the command using the raw-response=true flag.
> This will help you identify the information that you want to add to your extended data.

Using argument

The `using` argument specifies which integration instance to use when running an automation or command.

In XSOAR, an integration can have multiple instances configured, each potentially pointing to different servers or using different credentials. The `using` argument allows the user to specify which instance should be used for the given operation, ensuring the right connection is used.




```
!url url=https://google.com raw-response=true using="VirusTotal (API v3)
```

If no instance is specified, it will use all instances available.

Quiet Mode

Quiet Mode refers to an operational mode where the script or automation runs without producing verbose output.

This means that minimal information is printed to the War Room or console. Quiet Mode is beneficial when you want to reduce noise in the output logs or when running automated tasks where detailed output is unnecessary.

When you enable Quiet Mode, War Room entries are not created and inputs and outputs are not stored in the Work Plan. Quiet Mode improves performance by increasing playbook speed and saving database size (recommended in production environments). Turn it off when you are debugging.

4.6. Error Handling and Playbook Metadata

Key Points Summary

- DEMO: How to handle errors in playbooks. Configure an error path to send error email notification.
- DEMO: How to get playbook metadata.

Error Handling

Handling playbook errors in XSOAR involves defining how a playbook task should behave when an error occurs during execution.

This can be managed by setting specific options for each task within the playbook.

- **Stop**
  - If a task encounters an error, the playbook will stop executing.
  - This option is useful when you need manual intervention to review and resolve the error before proceeding.
  - Example: If a task requires manual review or approval, the playbook should halt until this step is completed.

- **Continue**
  - The playbook continues to execute even if the task encounters an error.
  - This is useful for non-critical tasks where an error doesn't impact the overall playbook flow.
  - Example: A task that requires data from an optional service like EWS can be skipped without affecting the main playbook.

- **Continue on Error Path**
  - On task error, the playbook follows a predefined error path.
  - This path can be a separate branch designed to handle errors and can include actions like clean-up, retry mechanisms, or logging.
  - Example: Define an error path to notify the engineering team.

To get task errors use the following command:

```
!GetErrorsFromEntry entry_id=${lastCompletedTaskEntries}
```

💡 The `${lastCompletedTaskEntries}` returns the ID of the last war room entry created by the previous playbook task.

Playbook Metadata

You can examine playbook metadata including the inputs and outputs of tasks, the storage space each task's inputs and outputs consume, and the types of tasks.

This information is helpful for troubleshooting a custom playbook if your system experiences slowdowns or high usage of CPU, memory, or disk space.

Once an incident is assigned to a playbook, you can analyse it to review the storage used by its tasks' inputs and outputs. You can also filter the data based on the kilobytes used by each task's inputs and outputs.

```
!getInvPlaybookMetaData incidentId=<incident ID> minSize=<size of the da
```

5. Automation Scripts Development

5.1. Demisto Class and Common Server functions

Key Points Summary

- Concepts: Demisto Class and Common Server.
- DEMO: Use of the most commonly used methods, functions and classes.

In XSOAR, previously known as Demisto, there are a few important components and classes that you need to understand to effectively write and manage automation scripts and integrations.

Two of the key components are the `Demisto` class and the `CommonServer` script.

- **Demisto Class:** All Python integrations and scripts include the `demisto` class object as part of the runtime environment. This object provides a set of API methods used to retrieve and send data to the XSOAR Server.

- **Common Server:** Common functions that will be appended to the code of each integration/script before being executed. Is a standard script provided by XSOAR that includes various helper functions and classes to simplify script writing. It contains reusable code that can be utilized across different scripts and integrations.
  - CommonServer (Javascript)
  - CommonServerPowerShell (Powershell)
  - CommonServerPython (Python)

Most Commonly Used Functions, Methods and Classes

Construct	Description	Return
`demisto.args()`	Retrieves a command / script arguments.	`dict` - Arguments object
`demisto.context()`	Retrieves the context data object of the current incident.	`dict` - Context data object
`demisto.incident()`	Retrieves the current incident and all its fields. The incident custom fields will be populated as a `dict` under the `CustomFields` attribute.	`dict` - dict representing an incident object

`demisto.get(`
obj: dict, field: str,
defaultParam=None
`)`
	Extracts field value from nested object. You can specify the default value to return in case the field doesn't exist in object.	`str` - The value of the extracted field

`demisto.executeCommand(`
command: str,
args: dict
`)`
	Executes given integration command / script and arguments.	Union[dict, list]: Command execution response wrapped in Demisto entry object

`demisto.setContext(`
contextPath: str,
value: str
`)`
	Sets given value in path in the context data.	`dict` - Object contains operation result status

`demisto.executeCommand(`
"setIncident", {}
`)`
	Sets given value in path in the incident data.	Union[dict, list]: Command execution response wrapped in Demisto entry object

`demisto.info(` msg: str,	Prints a message to the server logs in info/error level.	`None` - No data returned

Construct	Description	Return

*args: dict
 	
`)`
`demisto.error(`
 	
msg: str,
*args: dict
 	
`)`
	Server logs: /var/log/demisto/server.log	

`demisto.results(`
results: Union[list, dict]
 	
`)`
	Outputs entries to the war-room. It is recommended to use `CommandResults` and `return_results()` instead of `results()`.	`None` - No data returned

`CommandResults(`
outputs_prefix: str,
outputs: list | dict
 	
`)`
See full list of arguments in documentation reference
	`CommandResults` class - used to return results to war room.	

`return_results(`
results: CommandResults | str | dict
 	
`)`
	This function wraps the `demisto.results()`, supports.	`dict` - Result entry object

`return_warning(`
message: str, exit: bool=False,
warning: str='', outputs: dict | None=None,
ignore_auto_extract: bool=False
 	
`)`
	Returns a warning entry with the specified message, and exits the script.	`dict` - Warning entry object

`return_error(`
message: str,
error: str='', outputs: dict | None=None
 	
`)`
	Returns error entry with given message and exits the script	`dict` - Error entry object

To see the full list, check the documentation refence.

5.2. Develop in your favourite IDE

Key Points Summary

- Concepts: Demistomock.
- DEMO: How to use Demistomock to develop XSOAR scripts in Pycharm, Visual Studio Code or any other IDE.

- **Demistomock (the demisto object):** Is a Python module used for unit testing automation scripts and integrations. This module allows developers to mock XSOAR-specific functions and objects, enabling the simulation of the XSOAR environment during testing.
  - https://github.com/demisto/content/blob/master/Tests/demistomock/demistomock.py

💉 **XSOAR IDE limitations**

- Absence of an interpreter
- No code completion
- Absence of debugger
- No error detection and linting
- Syntax highlighting no customizable

Use Demistomock in your favourite IDE

```
demisto-playgrounds/
│
├── demistomock/
│   └── [Demistomock Files]
│
└── [Your Scripts]
```

5.3. Developing Automation Scripts

Key Points Summary

- DEMO: Develop an Automation Script using the acquired knowledge.

Get sample data for testing/debugging

```
# Export root data
!js script="return ${.}"

# Export context data
!py script="return_results(demisto.context())"
!PrintContext outputformat=json

# Export incident data
!py script="return_results(demisto.incident())"
!Print value=${.incident}
```

Use case - *AddIncidentTasks*

This automation script creates a task list for the analyst with specific instructions based on incident data. Analysts are expected to perform a list of steps, or tasks, in the process of triaging, investigating, or remediating an incident.

5.4. Docker Images

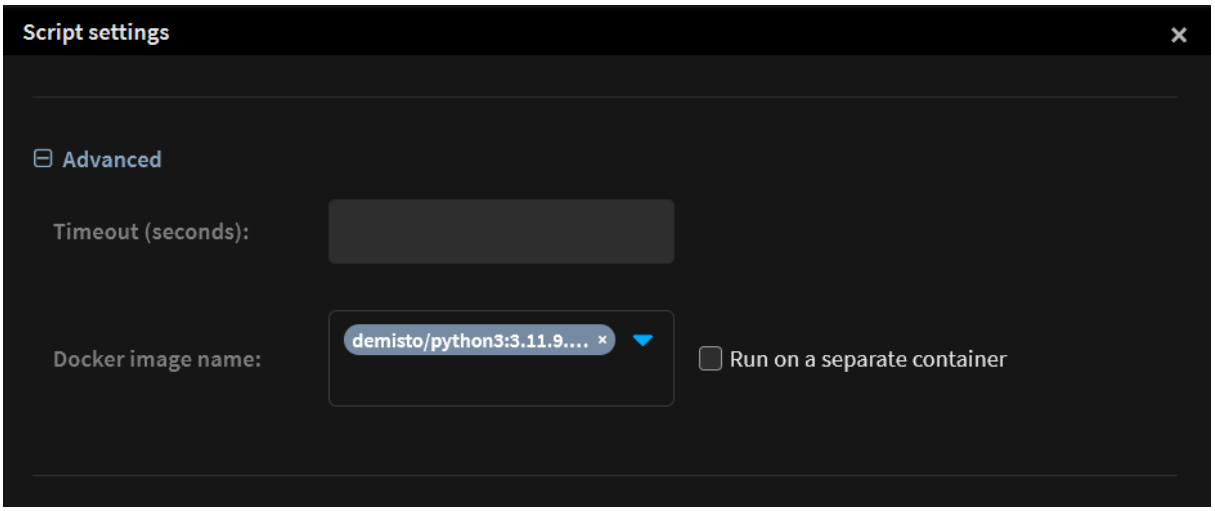
Key Points Summary

- Concepts: Docker in XSOAR.
- DEMO: Create a custom Docker Image to your install dependencies.

- **Docker:** Docker is a platform that allows developers to package applications and their dependencies into containers. These containers are lightweight, portable, and consistent across different environments, making it easier to develop, deploy, and run applications.
  - XSOAR uses Docker primarily for the execution of certain integrations and automations. This helps in isolating execution environments and managing

dependencies.

- Docker is required for engines to run Python/Powershell scripts and integrations in a controlled environment.
- All Docker images are available via Docker hub under the Demisto organization:
  - https://hub.docker.com/u/demisto/
- The Docker image creation process is managed via the open source project demisto/dockerfiles.
- Docker images can be selected inside Automations and Integrations **Settings**.



Commands

List docker images (docker images)

```
/docker_images
```

Update Docker Images (docker pull)

```
/docker_image_update all=<bool> | image=<image_name>
```

Create a new Docker image and install dependencies (docker build)

```
/docker_image_create name="<name>" base="<docker_image_base>" dependen
cies="<dependency1>, <dependency2>" packages="<package1>, <package2>"
```

Example: Create a docker image and install the beatifulsoup4 python library.

`/docker_image_create name="demisto/py3-bs4" base="demisto/python3-deb:3.8.2.6981" dependencies=beautifulsoup4`

Fix Permission Denied Error

> 🚫 Error creating the docker image: ERROR: mkdir /home/demisto: permission denied (2650)

Run the following commands in the XSOAR server.

```
sudo mkdir -p /home/demisto
sudo chown demisto:demisto /home/demisto
```

5.5. XSOAR API

Key Points Summary

- Concepts: XSOAR API and demisto-py.
- DEMO: Core REST API Integration overview.
- DEMO: Running XSOAR commands and creating incidents using demisto-py.

XSOAR API

The XSOAR API is a component of Cortex XSOAR that allows for programmatic interaction with the XSOAR platform, enabling integration with various security tools, customization of workflows, and automation of security operations.

- To get your API KEY: Settings → Integrations → API Keys

Example using `curl`

```
 curl 'https://<host>:443/incidents/search' -H 'content-type: applicat
ion/json' -H 'accept: application/json' -H 'Authorization: <your api k
ey>' --data-binary '{"filter":{"query":"-status:closed -category:jo
b","period":{"by":"day","fromValue":30}}}' --compressed -k
```

Example using Core REST API Integration

```
!core-api-get uri="/health/containers”
```

Demisto-py

Demisto-py is a Python library designed to interact with the XSOAR API. A Demisto Client for Python.

- https://pypi.org/project/demisto-py/
- https://github.com/demisto/demisto-py

Install demisto-py

```
pip install demisto-py
```

Example Script

```
import demisto_client.demisto_api
from demisto_client.demisto_api.rest import ApiException

api_key = 'XXXXXXXXXXXX'
host = 'https://X.X.X.X'
api_instance = demisto_client.configure(base_url=host, api_key=api_ke
y, debug=False, verify_ssl=False)


def create_incident() -> None:
    create_incident_request = demisto_client.demisto_api.CreateInciden
tRequest()
    create_incident_request.name = 'Sample Simulation Incident'
    create_incident_request.type = 'Simulation'
    create_incident_request.owner = 'Admin'

    try:
        api_response = api_instance.create_incident(create_incident_re
quest=create_incident_request)
        print(api_response)
    except ApiException as e:
        print("Exception when calling DefaultApi->create_incident: %s
\n" % e)
```

```
def run_command() -> None:
    update_entry = demisto_client.demisto_api.UpdateEntry(data="!Gener
atePassword min_digits=1",
                                                          investigatio
n_id="529")  # UpdateEntry |  (optional)
    try:
        # Create new entry in existing investigation
        api_response = api_instance.investigation_add_entries_sync(upd
ate_entry=update_entry)
        print(api_response)
    except ApiException as e:
        print("Exception when calling DefaultApi->investigation_add_en
tries_sync: %s\n" % e)


if __name__ == '__main__':
    run_command()
```

6. Integration Development

6.1. Integration Categories and Uses Cases

Key Points Summary

- XSOAR Integration Categories and Use Cases.
- How Integrations work.

Integration Categories and Use Cases

As mentioned in a previous section, an integration in XSOAR is a connector or interface that allows the platform to interact with external systems, applications, or services. Integrations enable automated data exchange, incident management, and execution of actions across different security and IT tools.

Refer to the link below to see typical use cases for various Cortex XSOAR integration categories.

- https://xsoar.pan.dev/docs/concepts/use-cases

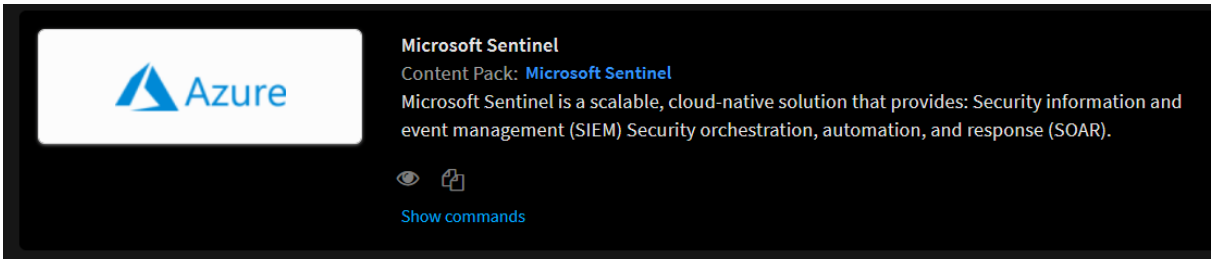
Category	Top Use Cases
Analytics and SIEM	Fetch Incidents with relevant filters, Create, close and delete incidents/events/cases
Authentication	Use credentials from authentication vault in order to configure instances in Cortex XSOAR, Lock/Delete Account – Give option to lock account (credentials), and unlock/undelete
Case Management	Create, get, edit, close a ticket/issue, add + view comments, Assign a ticket/issue to a specified user
Data Enrichment & Threat Intelligence	Enriching information about different IOC types: Upload object for scan and get the scan results, Search for former scan results about an object
Email Gateway	Get message – Download the email itself, retrieve metadata, body, Download attachments for a given message
Endpoint	Fetch Incidents & Events, Get event details (from specified incident)
Forensics and Malware Analysis	Submit a file and get a report (detonation), Submit a URL and get a report (detonation)
Network Security (Firewall)	Create block/accept policies (Source, Destination, Port), for IP addresses and domains, Add addresses and ports (services) to predefined groups, create groups, etc.
Network Security (IDS/IPS)	Get/Fetch alerts, Get PCAP file, packet
Vulnerability Management	Enrich asset – get vulnerability information for an asset (or a group of assets) in the organization, Generate/Trigger a scan on specified assets
IAM (Identity and Access Management)	Create, update and delete users. Block users, Force change of passwords.

And there are still many more categories of integrations; you can consult these in the documentation.

How Integration works

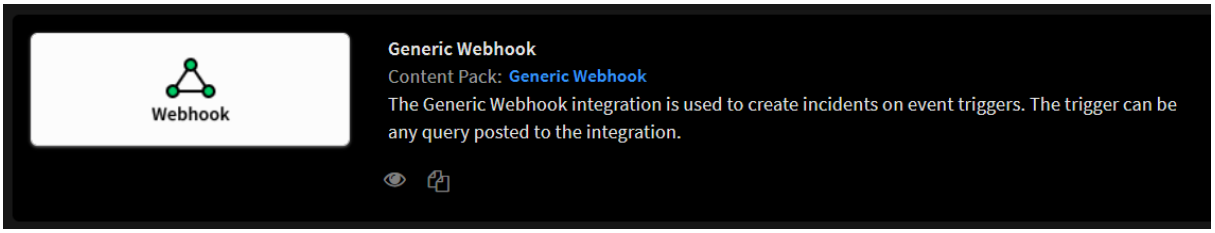
1. API-Based Integrations

API (Application Programming Interface) integrations are one of the most common methods used in XSOAR. These integrations use APIs provided by external systems to fetch data, execute actions, or send data back to those systems.



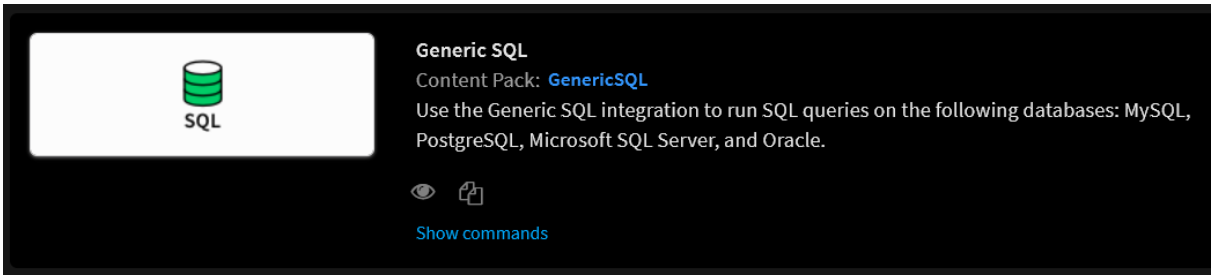
2. Webhook-Based Integrations

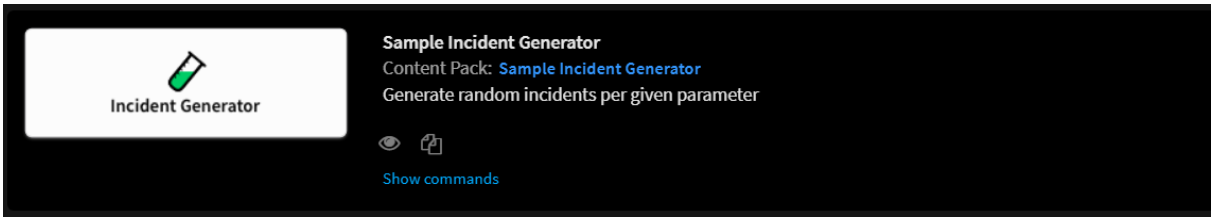
Webhooks provide a way for external systems to push real-time data or execute automation scripts in XSOAR without the need for continuous polling.



3. Other

- Database Integrations
- CLI Integrations
- etc.





6.2. Integration Commands, Methods and Functions

Key Points Summary

- Overview of the most commonly used commands, methods and functions in Integrations.

Most Commonly Used Commands, Methods and Functions in Integrations

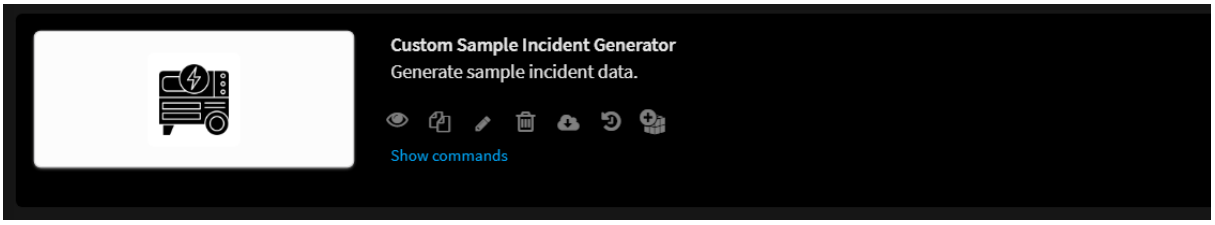
Construct	Description	Return
`demisto.command()`	Retrieves the integration command that is being run.	`str` - Integrations command name
`demisto.getIntegrationContext()`	Retrieves the IntegrationContext object.	`dict` - IntegrationContext object
`demisto.setIntegrationContext(` context: dict `)`	Stores given object in the IntegrationContext object.	`None` - No data returned
`demisto.getLastRun()`	Retrieves the LastRun object. This helps avoid duplicate incidents by fetching only events that occurred since the last time the function was run.	`dict` - LastRun object
`demisto.setLastRun(` obj `)`	When the last events are retrieved, you need to save the new last run time to the integration context. This timestamp will be used the next time the ***fetch-incidents*** function runs.	`None` - No data returned
`demisto.incidents(` incidents: list `)`	In script, retrieves the `Incidents` list from the context. In integration, used to return (create) incidents to the server.	`None` - No data returned
`demisto.integrationInstance()`	Retrieves the integration instance name in which ran in.	`str` - The integration instance name
`demisto.params()`	Retrieves the integration parameters object.	`dict` - Integrations parameters object
`register_module_line(` module_name: str, start_end: str `)`	Register a module in the line mapping for the traceback line correction algorithm.	`None` - No data returned
`fetch-incidents`	Built- in command that XSOAR calls to import new incidents. It is triggered by the *Fetches incidents* parameter	`None` - No data returned

Construct	Description	Return
	in the integration configuration.	
`test-module`	Built-in command that XSOAR calls when the user has clicked the integration test button while setting up or editing an integration instance.	When returning **ok**, the user is shown a green **Success** message. If any value other than **ok** is returned, an error is displayed
`fetch-indicators`	Built- incommand that XSOAR calls to import new indicators. It is triggered by the *Fetches indicators* parameter in the integration configuration.	`None` - No data returned

6.3. Developing Integrations I

Key Points Summary

- DEMO: Develop an Utility Integration using the acquired knowledge.



- Fetches incidents using sample data.
- Load sample incidents on demand.

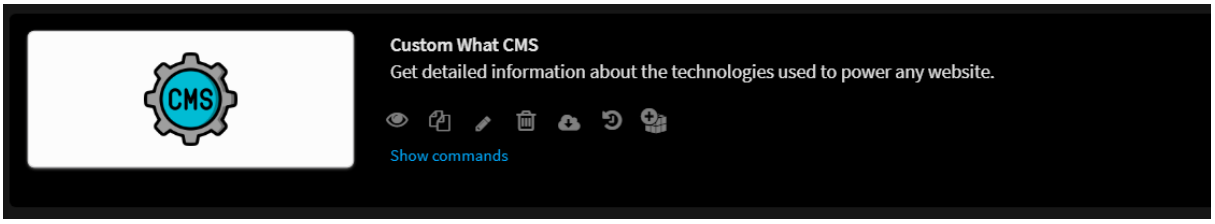
Sample data

```
{"alertName": "Phishing Attack", "type": "phishing", "email": "test@ou
tlook.com"}
```

6.4. Developing Integrations II

Key Points Summary

- DEMO: Develop an API-based Integration using the acquired knowledge.
- EXTRA: Troubleshooting.



- Detect web technologies
- API Development Platform: https://hoppscotch.io/

Integration logs (Log Level On)

```
/var/log/demisto/integration-instance.log
```

```
!command debug-mode=true
```

7. Pre-processing and Post-processing

7.1. Pre-Processing Rules

Key Points Summary

- Concepts: Pre-Process Rules and actions.
- DEMO: Create a Pre-process rule to filter incidents.

- **Pre-processing Rules:** Pre-process rules enable you to perform certain actions on incidents as soon as they are ingested (after classification and mapping) but before the

incident is created XSOAR. These rules help to streamline incident management by performing specific actions such as deduplication, linking, dropping, or closing incidents based on defined criteria.

- Settings → Object Setup → Pre-Process Rules

> 💡 Rules are executed from top to bottom. Once a rule is triggered, the flow stops, and no further rules are triggered.

Actions for Pre-Processing Rules

`Link and Close`	Links the incoming incident to an existing one and closes the incoming incident. If no matching incident is found, a new incident is created.
`Close`	Closes the incoming incident without running the associated playbook.
`Drop`	Discards the incoming incident without creating it, useful for low or no-value incidents.
`Drop and Update`	Discards the incoming incident and updates the Dropped Duplicate Incidents table of an existing incident, also making an entry in the War Room. If no match is found, a new incident is created.
`Link`	Links the incoming incident to an existing one, without closing it.
`Run a Script`	Executes a predefined script on the incoming incident.

7.2. Pre-processing Scripts

Key Points Summary

- Concepts: Pre-processing Scripts and use cases.
- DEMO: Create a Pre-process rule to run a custom script that adds custom tags.

Pre-processing scripts in XSOAR are scripts executed on incoming incidents as part of the pre-processing phase. This phase occurs after incident classification and mapping but before the incident is fully created in the system. These scripts allow for more complex and dynamic handling of incidents based on specific conditions and criteria that may not be covered by standard pre-processing rules.

💡 Pre-processing scripts are identified using the `preProcessing` tag.

Some Use Cases

- Incident Deduplication
- Incident Enrichment
- Add Tasks for Analysts.
- Conditional Incident Routing
- Advanced Incident Filtering and tagging
- Incident Prioritization
- Compliance Check
- Data Transformations
- etc.

7.3. Post-processing Scripts

Key Points Summary

- Concepts: Post-processing scripts, available arguments and use cases.
- DEMO: Use a Post-processing script to ensure incident's required information is in place before closing.

Post-processing scripts in XSOAR are scripts that execute additional actions after an incident has been remediated and is about to be closed. These scripts allow you to perform various tasks such as updating external systems, sending notifications, or adding final notes to the incident. They ensure that all necessary follow-up actions are taken care of automatically, streamlining the incident management workflow.

Post-processing are added to Incident Types.

💡 Post-processing scripts are identified using the `post-processing` tag.
Once the post-processing script completes the incident closes.
If a post-processing script returns an error, the incident does not close.

Available Arguments

`closed`	The time the incident was closed.
`status`	The current status of the incident.
`openDuration`	The duration for which the incident was open.
`closeNotes`	Notes added at the time of closing the incident.
`closingUserId`	The ID of the user who closed the incident, or "DBot" if it was closed by an automated process.
`closeReason`	The reason for closing the incident.
Other fields	Any other field values passed in at closure, whether through the incident close form, the CLI, or a playbook task.

Some Use Cases

- Ticket Closure in External Systems (ServiceNow, Jira, etc.).
- Email or chat message notification.
- Documentation and Logging.
- Metrics and KPIs Tracking.
- Sync with External Systems.
- Incidents Fields Check
- etc.

8. Building Your XSOAR Automated Workflow

8.1. Use Case Definition

Key Points Summary

- Use Case Template overview
- Defining the Use Case and scenario

In this final module, the goal is to consolidate all the knowledge acquired to build your XSOAR Automated Workflow. Typically, everything begins by defining a use case.

It is recommended to use a **Use Case Definition Template** to plan it. This will help us determine what is needed to implement the automation effectively. You can create your own

or use any other provided.

Use Case Template provided by Pal Alto

- https://www.paloaltonetworks.com/content/dam/pan/en_US/assets/pdf/cortex-xsoar/cortex-xsoar-use-case-template.pdf

Normally, when building a automated workflow from scratch, the following steps are followed:

1. Integration installation and configuration.
2. Incident Types and Fields
3. Classifiers and Mappers
4. Pre-processing
5. Playbook development
6. Post-processing
7. Final Tests

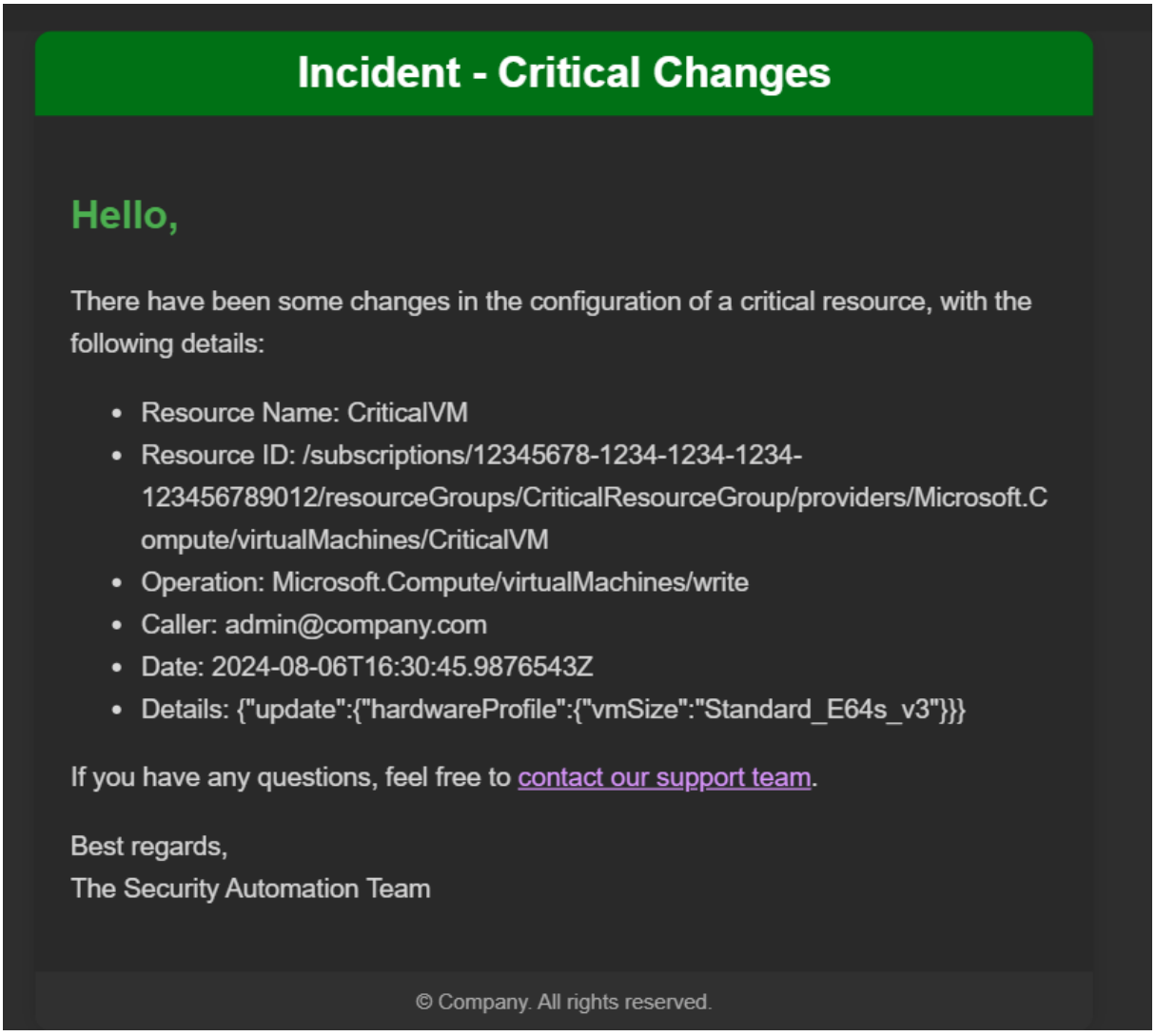
Scenario

As a Security Automation Engineer, you are responsible for ensuring that any critical changes made to your organization environment are promptly identified and communicated to relevant stakeholders.

Your lead has tasked you with building an automated workflow that notifies critical changes on configurations three key areas:

- Virtual Machines (VMs)
- Networking and Security Information
- Event Management (SIEM)

The goal is to send notifications to the respective teams whenever a critical change occurs.



Use Case Definition

Name	Automated notification for critical changes
Description	Playbook that alerts for critical changes in VM configurations, network configurations, and SIEM configurations.
Trigger	Via Webhook (Integration)

Remediation Steps

1. Determine if the change operation is successful (pre-process).
2. Check the type of resource to determine which technical team it belongs to.
    a. Microsoft.Compute → Sysadmin Team

b. Microsoft.Network → Networking Team

c. Microsoft.SecurityInsights → SIEM/SOAR Team

3. Build email body and send notification.

Integrations

Integration Name	Actions Needed
Generic Webhook	Create incidents on event triggers. The trigger can be any query posted to the integration.
Mail Sender	send-mail: Send an email.

Classification

Critical Changes - Classifier

Logic	Incident Type
level == “critical”	Critical Changes

Mapping

Critical Changes - Mapper

Raw Key	XSOAR Field	System
operationName	Operation Name	Yes
status	Status Reason	Yes
caller	Caller	Yes
eventTimestamp	Event Timestamp	No
level	severity	Yes
resourceId	Resource ID	Yes
resourceId (transform)	Resource Name	Yes
properties	Details	Yes
	name (transform)	Yes

Pre-processing

Rule Name	Logic	Action
Drop not successful Critical Changes	type equals Critical Changes AND severity equals Critical AND statusreason doesn't equal Succeeded	Drop

Playbooks

Playbook Name	Inputs	Outputs
Critical Changes - Notification v1	-	-

Post-processing

None

Sample - Virtual Machine

VM Resize to a Much Larger Size (Critical)


```
{
    "time": "2024-08-06T16:30:45.9876543Z",
    "operationName": "Microsoft.Compute/virtualMachines/write",
    "status": "Succeeded",
    "properties": {
        "statusCode": "OK",
        "serviceRequestId": "c4d5e6f7-8901-2345-6789-abcdef012345",
        "errorCode": null,
        "eventCategory": "Administrative",
        "isComplianceCheck": false
    },
    "caller": "admin@company.com",
    "correlationId": "f6g7h8i9-0123-4567-8901-bcdef1234567",
    "resourceId": "/subscriptions/12345678-1234-1234-1234-12345678901
2/resourceGroups/CriticalResourceGroup/providers/Microsoft.Compute/vir
tualMachines/CriticalVM",
    "eventTimestamp": "2024-08-06T16:30:45.9876543Z",
    "submissionTimestamp": "2024-08-06T16:31:15.9876543Z",
    "eventDataId": "e7f8g9h0-1234-5678-9012-abcd34567890",
    "level": "Critical",
    "resourceGroupName": "CriticalResourceGroup",
    "resourceProviderName": {
        "value": "Microsoft.Compute",
        "localizedValue": "Microsoft.Compute"
    },
    "resourceType": {
        "value": "Microsoft.Compute/virtualMachines",
        "localizedValue": "Virtual Machines"
    },
    "operationId": "g8h9i0j1-2345-6789-0123-cdef45678901",
    "operationName": {
```

```
        "value": "Microsoft.Compute/virtualMachines/write",
        "localizedValue": "Update Virtual Machine"
    },
    "properties": {
        "update": {
            "hardwareProfile": {
                "vmSize": "Standard_E64s_v3"
            }
        }
    }
}
```

Sample - Networking

Critical Network Security Group (NSG) Rule Change

```
{
    "time": "2024-08-06T17:15:30.4567890Z",
    "operationName": "Microsoft.Network/networkSecurityGroups/security
Rules/write",
    "status": "Succeeded",
    "properties": {
        "statusCode": "OK",
        "serviceRequestId": "d6e7f8g9-0123-4567-8901-abcdef234567",
        "errorCode": null,
        "eventCategory": "Administrative",
        "isComplianceCheck": false
    },
    "caller": "networkAdmin@company.com",
    "correlationId": "h9i0j1k2-3456-7890-1234-def567890123",
    "resourceId": "/subscriptions/12345678-1234-1234-1234-12345678901
2/resourceGroups/SecurityResourceGroup/providers/Microsoft.Network/net
workSecurityGroups/CriticalNSG",
    "eventTimestamp": "2024-08-06T17:15:30.4567890Z",
    "submissionTimestamp": "2024-08-06T17:16:00.4567890Z",
    "eventDataId": "i0j1k2l3-4567-8901-2345-efgh67890123",
    "level": "Critical",
    "resourceGroupName": "SecurityResourceGroup",
    "resourceProviderName": {
        "value": "Microsoft.Network",
        "localizedValue": "Microsoft.Network"
    },
    "resourceType": {
        "value": "Microsoft.Network/networkSecurityGroups",
```

```
        "localizedValue": "Network Security Groups"
    },
    "operationId": "j1k2l3m4-5678-9012-3456-ghij78901234",
    "operationName": {
        "value": "Microsoft.Network/networkSecurityGroups/securityRule
s/write",
        "localizedValue": "Update Network Security Group Rule"
    },
    "properties": {
        "update": {
            "securityRule": {
                "name": "Allow-SSH",
                "properties": {
                    "priority": 100,
                    "direction": "Inbound",
                    "access": "Allow",
                    "protocol": "Tcp",
                    "sourcePortRange": "*",
                    "destinationPortRange": "22",
                    "sourceAddressPrefix": "*",
                    "destinationAddressPrefix": "*"
                }
            }
        }
    }
}
```


Sample - SIEM

Adding a New Data Connector in Microsoft Sentinel


```
{
    "time": "2024-08-06T18:45:20.6543210Z",
    "operationName": "Microsoft.OperationalInsights/workspaces/provide
rs/Microsoft.SecurityInsights/dataConnectors/write",
    "status": "Succeeded",
    "properties": {
        "statusCode": "OK",
        "serviceRequestId": "e7f8g9h0-1234-5678-9012-abcd345678ef",
        "errorCode": null,
        "eventCategory": "Administrative",
        "isComplianceCheck": false
    },
    "caller": "siemAdmin@company.com",
```

```
    "correlationId": "k2l3m4n5-6789-0123-4567-ijkl89012345",
    "resourceId": "/subscriptions/12345678-1234-1234-1234-12345678901
2/resourceGroups/SentinelResourceGroup/providers/Microsoft.Operational
Insights/workspaces/SentinelWorkspace/providers/Microsoft.SecurityInsi
ghts/dataConnectors/NewDataConnector",
    "eventTimestamp": "2024-08-06T18:45:20.6543210Z",
    "submissionTimestamp": "2024-08-06T18:46:00.6543210Z",
    "eventDataId": "l3m4n5o6-7890-1234-5678-mnop12345678",
    "level": "Critical",
    "resourceGroupName": "SentinelResourceGroup",
    "resourceProviderName": {
        "value": "Microsoft.SecurityInsights",
        "localizedValue": "Microsoft.SecurityInsights"
    },
    "resourceType": {
        "value": "Microsoft.SecurityInsights/dataConnectors",
        "localizedValue": "Data Connectors"
    },
    "operationId": "m4n5o6p7-8901-2345-6789-opqr34567890",
    "operationName": {
        "value": "Microsoft.SecurityInsights/dataConnectors/write",
        "localizedValue": "Create or Update Data Connector"
    },
    "properties": {
        "dataConnector": {
            "name": "NewDataConnector",
            "properties": {
                "kind": "AzureActivity",
                "connectorUiConfig": {
                    "title": "Azure Activity",
                    "description": "Connect Azure Activity logs to Azu
re Sentinel",
                    "instructions": "Follow the instructions to enable
Azure Activity logs data connector."
                },
                "config": {
                    "state": "Enabled",
                    "logAnalyticsWorkspaceResourceId": "/subscription
s/12345678-1234-1234-1234-123456789012/resourceGroups/SentinelResource
Group/providers/Microsoft.OperationalInsights/workspaces/SentinelWorks
pace"
                }
            }
        }
```

```
    }
}
```

8.2. Walkthrough

Key Points Summary

- DEMO: Building your XSOAR Automated Workflow

In this walkthrough, we will cover:

- Integration Installation and Set Up
- Incident Types and Fields
- Classifiers and Mappers
- Pre-processing
- Playbook Development
- Final Tests

Check if server is listening for webhooks

```
netstat -pnltu | grep <port>
```

Example: netstat -pnltu | grep 8080

Send test POST request

```
curl -X POST -d '{"test":"test"}' http://<server>:<port>
```

Example: curl -X POST -d '{"test":"test"}' http://localhost:8080

8.3. Another Use Case and Automated Workflow

Key Points Summary

- Overview of another use case definition and automated workflow.

Use Case Definition

Name	A potentially phishing link was clicked
Description	User clicked on a link that may be part of a phishing attempt. The aim is to implement a playbook to determine if it is truly malicious, discard false positives, notify the necessary teams, block it, etc.
Trigger	Via API (Integration)

Remediation Steps

1. Determine if the URL is malicious.
2. If not malicious close incident as False Positive and add it to a list.
3. If malicious:
    a. Run quick device scan and notify EDR Team.
    b. Reset user password and provide new password.
    c. Block URL and add it to a list.

Integrations

Integration Name	Actions Needed
VirusTotal (API v3)	url: Checks the reputation of a URL.
SplunkPy	fetch-incidents
Microsoft Defender for Endpoint	microsoft-atp-run-antivirus-scan: Initiate a Microsoft Defender Antivirus scan on a machine.
Azure Active Directory Users	msgraph-user-change-password: Changes the user password.
Zscaler Internet Access	zscaler-blacklist-url: Adds the specified URLs to the block list.
Mail Sender	send-mail: Send an email.

Classification

Phishing - Custom - Classifier

Logic	Incident Type
alarm_category == “Phishing”	Phishing - Custom

Mapping

Phishing - Custom - Mapper

Raw Key	XSOAR Field	System
User	User	Yes
URL	Email URL Clicked	Yes
DeviceID	Device Id	Yes
DeviceName	Device Name	Yes
siem_type_id	Vendor Product	Yes
alarm_category	Type	Yes

Pre-processing

Rule Name	Logic	Action
Drop incidents with excluded URLs	If URL in list(ExcludedURLs)	Drop

Playbooks

Playbook Name	Inputs	Outputs
Phishing - Custom v1	-	-
AAD - Reset User Password	user	NEW_PASSWORD

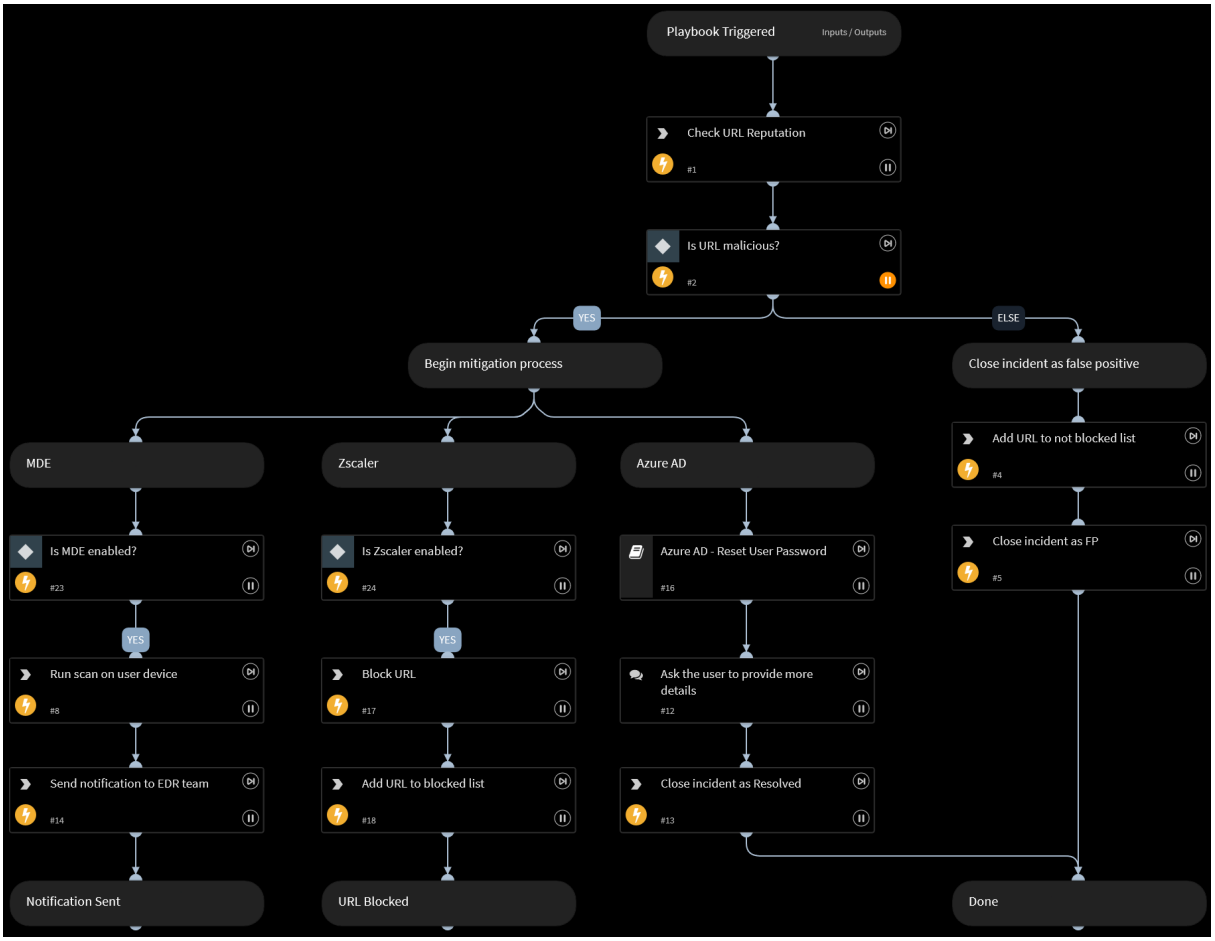
Post-processing

Script Name	Objective
CheckDataPostProcessing	Verify close reason and notes are provided.

Sample

```
{
  "events": [
    {
      "DeviceName": "test-device",
      "DeviceID": "6fa3f44c8f4b4b5b8c12345678abcdef",
      "User": "john.doe@mycompany.com",
      "URL": "https://duckduckgo.com"
    }
```

```
  ],
  "security-alarm": {
    "alarm_category": "Phishing",
    "event_type_id": 172157,
    "name": "A potentially phishing link was clicked",
    "num_events": 1,
    "risk": 2,
    "siem_type_id": "Splunk",
    "timestamp": 1722848280000,
  }
}
```

9. Course Conclusion

Thank you for joining me on this journey!

By now, you should have a solid understanding of how to leverage XSOAR's robust capabilities to integrate security tools and streamline incident response.

Remember, the PDF guide and source code provided will be valuable resources as you continue to refine your skills and apply what you've learned in real-world scenarios. Automation is a powerful tool that can dramatically improve response times and reduce human error, and with the knowledge gained in this course, you're well-equipped to make the most of it.

If you have any questions or need further assistance, don't hesitate to reach out.

Thank you once again for your time and support. I look forward to seeing how you apply your new skills in the field and wish you all the best in your future security automation projects.

Happy automating!

Kalec Blau

10. Reference and Further Reading

Cortex XSOAR Administrator Guide

https://docs-cortex.paloaltonetworks.com/r/Cortex-XSOAR/6.5/Cortex-XSOAR-Administrator-Guide/Overview

XSOAR Content GitHub

https://github.com/demisto/content

XSOAR Developer Docs

https://xsoar.pan.dev/docs/welcome

Cortex XSOAR Python Development Quick Start Guide

https://docs-cortex.paloaltonetworks.com/r/Cortex-XSOAR/6.x/Cortex-XSOAR-Python-Development-Quick-Start-Guide

Cortex XSOAR Tutorials

https://docs-cortex.paloaltonetworks.com/r/Cortex-XSOAR/6.x/Cortex-XSOAR-Tutorials-6.x

Cortex XSOAR Playbook Design Guide

https://docs-cortex.paloaltonetworks.com/r/Cortex-XSOAR/6.x/Cortex-XSOAR-Playbook-Design-Guide

Best Practices

https://docs-cortex.paloaltonetworks.com/r/Cortex-XSOAR/6.x/Cortex-XSOAR-Playbook-Design-Guide/Best-Practices

Feel free to share this material with anyone who needs it for learning XSOAR, as long as the authorship is mentioned :)

If it has been helpful to you, please support me by purchasing my course on Udemy using this link.

https://www.udemy.com/course/xsoar-security-orchestration-and-automation-course/?referralCode=8F469AAD51A79BAA5950

Thank you very much for your support!

Kalec Blau ⚡

Happy Automating!