SPIRION

Spirion Extensions

Spirion Automation: Python Scripting Overview

Table of Contents

Spirion Automation: Python Scripting Overview

Introduction

Accurate data discovery is “Step One” for proper data loss prevention (DLP), however not all solutions with DLP capabilities are equally equipped for the task. This creates uneven footing as organizations attempt to implement privacy and security strategies across environments plagued by inconsistent visibility into fluctuating sensitive data footprints.

Spirion should instead serve as the single source of truth for software that lacks Privacy-Grade data discovery. Rather than relying on a patchwork of search utilities built into individual platforms, information governance policies can be standardized around the precision of Spirion scan results – while still incorporating control mechanisms from external DLP tools that can be orchestrated either by persistent metadata tags or through API (or CLI) automation from Spirion’s scripting engine.

This extension describes how common support for Python across APIs, SDKs, and Spirion scripts make it ideal for prototyping how to integrate match results from a Spirion discovery scan with meaningful functionality from third-party applications. The following information serves as reference for automating logic via a locally executed Python script called from a SDM workflow or an SDP playbook.

Requirements

Before working on the steps outlined in this document, please confirm the following:

- The Spirion console (SDM or SDP) is up-to-date and accessible.
- A Spirion endpoint agent hosted on a Windows 10 system is online, polling to its console.
- Python is installed on the Windows host running the Spirion agent.
    - **NOTE:** LocalSystem must be able to run the Python script called by Spirion.

Important Notes

Software Versions

SDP console version 22.Q3.1.226.1, SDM console version 11.8.2, Spirion agent version 12.5, and Python 3.11.1 were used to perform the steps described in this document.

Sample Script

The configuration explained below references a generic Python script that writes the match location reported by Spirion into an example text file.

A growing list of application-specific script examples is also included at the end of this document.

Process

Spirion can automate the execution of shell scripts with variables derived from scan results returned by Spirion agents. In SDM, this is handled from the Action tab of any given Workflow, and from Playbooks in SDP.

To execute a Python script, Spirion is configured to run a batch script that calls Python while passing in the Spirion match location (typically a file path) as a command-line argument using the "%Location%" variable.

Creating a Python Script

The Python script must reside on the same system hosting the Spirion agent. This script will be called by the Spirion agent for each match location upon completion of a scan, running as the LocalSystem account. A sample Python script is available on the Spirion website in file sample-pythonscript.txt.

1. From the Spirion agent host system, select a directory in which the Python script should run.
    a. Scripts included with this document reference `C:\temp`.
2. Using the text editor of your choice, create the following Python script:

```
from sys import argv


def write_match(file_path, match_location):

f = open(file_path, "a+")

f.write(f"MATCH FOUND: { match_location }\n")

f.close()
```

```
if __name__ == "__main__":

# join input to handle spaces in file path

match_location = " ".join(argv[1:])

file_path = "C:\\temp\\example.txt"

write_match(file_path, match_location)
```

3. Save this file in the directory specified in the first step.
    a. For example, `C:\temp\script.py`

The example script above takes a command-line argument and writes it to a text file. Spirion can pass the following information into custom scripts as variables that return data based on scan results:

- **%AnyfindCount%**: This expands to "<count> <identity type name>(s)", e.g. "18 Password(s)", "39 Social Security Number(s)" for the current location.
- **%AnyFindTotalCount%**: This expands to "<count> <data type name>(s)", e.g. "10 Password(s)", "34 Social Security Number(s)", etc.
- **%DataTypes%**: This returns a comma-separated list of all data types found for the current location. e.g. "Bank Account Number, Dictionary, Telephone Number."
- **%EndPointName%**: This returns the display name of the endpoint on which the result was found.
- **%Location%**: This return the full path or other location in which the match was found. This variable displays enough information to be able to get back to the source of the result from the machine on which it was found. For example, the file path is relative to the agent that ran the search, an email location contains message folder names, time stamps, and subjects, a database location includes table and column information and a website location includes the full URL.
- **%LocationType%**: This returns the type(s) of location that was found, e.g. "Text Document, XLSX File."
- **%Matches%**: This returns a numerical value representing the number of matches for the current location. e.g. if there were 15 SSN and 25 CCN found in the current location, the value returned would be 40.

- **%SearchUserName%**: This returns the name of the user who performed the search on the endpoint, e.g. "Administrator".
- **%TaskId%**: This returns the task id under which the search was run. The %TaskId% uniquely identifies a specific task stored in the database. A TaskId can be viewed in a report using the *Searches-Task Identifier* column.
- **%TotalDataTypes%**: This returns a numerical value representing the total number of data types found in the search. e.g. if SSN's and CCN's were found during the search, the value returned would be 2.
- **%TotalMatches%**: This returns a numerical value representing the total number of matches found during the search. e.g. if there were 200 SSN and 125 CCN found during the search, the value returned would be 325.
- **%TotalUniqueMatches%**: This returns a numerical value representing the total number of unique matches found in the search.
- **%Owner%**: This returns the file system owner for locations that are files. For data from Windows agents, the NTFS owner of the file. For data from Mac agents, the file system owner of the file.
- **%UniqueMatches%**: This returns a numerical value representing the number of unique matches for the current location.

Configuring the Spirion Agent

The location where the Python script is saved will be referenced in the batch file uploaded to the Spirion console per the instructions that follow.

NOTE: Be sure to also install any Python module dependencies so that they are accessible to the LocalSystem account.

Configuring Sensitive Data Platform (SDP)

Uploading a Script

1. From the main menu, select ***Settings***.
2. Navigate to ***Script Repository***.
3. Click the ***Actions*** button toward the top-right of SDP and select “Add Script.”
4. Specify a ***name*** and ***description*** for the script.
5. Click the icon next to ***Upload Script*** and upload the following command as a batch file:

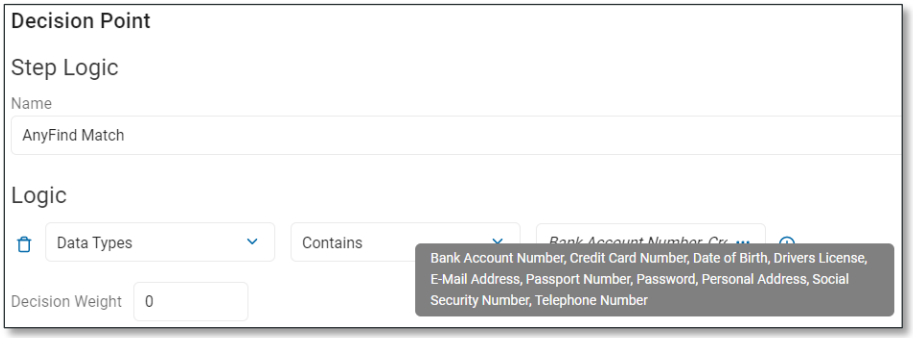
```
python c:\temp\script.py %Location%
```

6. ***Save*** the uploaded script.

Creating a Playbook

The steps described below cover the relevant steps for automated script execution. For additional guidance, please refer to the Spirion KB Article: How to Write a Playbook.

1. From the main menu, select ***Scans***.
2. Navigate to ***Scan Playbooks***.
3. Click ***Add Playbook*** using the button from the top-right.
4. Enter a ***Name*** and ***Description*** as desired.
5. Under the ***Logic*** section of a ***Decision Point***, select the criteria that should trigger the rule.
    a. For example, specifying "Data Types" and selecting any (or all) of the built-in AnyFind search parameters would trigger this script for every location (e.g. file/document) with 1 or more of the selected data types.



6. From the ***Select Action*** pulldown menu, select "Execute Script" for the action card that corresponds to the intended decision path (yes/no) per the logic specified in the previous step.
7. Under ***Select Script***, choose the script uploaded in the previous section.
8. Define additional Playbook logic as necessary before clicking the ***Save*** icon to finalize changes.

Searching for Data

After creating the Playbook it needs to be referenced in a scan policy. This policy controls what target(s) are scanned by the Spirion agent(s) staged with the Python script. Playbook logic evaluates results from this scan and triggers the Python script when applicable.

Please refer to the following documentation for additional information about scan policies: Getting Started with Scans.

Configuring Sensitive Data Manager (SDM)

Creating a Workflow

The steps described below cover the relevant steps for automated script execution. For additional guidance, please refer to the Spirion KB Article: How to Write a Workflow Rule.

1. From the ***Workflows*** tab, select "Add" from the ***Rule*** pulldown menu.
2. Enter a ***Name*** and ***Description*** as desired.
3. On the ***Definition*** tab, select the criteria that should trigger the rule.
    a. For example, specifying "Total Matches Greater Than or Equals 1" would trigger this workflow for every scan with 1 or more match results.

4. On the ***Endpoints*** tab, select the agent(s) that will execute the Python script.
5. On the ***Actions*** tab, enable ***Perform the following remediation action*** and select "Execute script."
6. Click the ellipsis button ("**...**") and upload the following command as a batch file:

```
python c:\temp\script.py %Location%
```

7. Finalize the creation of this Workflow by clicking ***Finish***.

Searching for Data

Once the Workflow has been synced to the Spirion agent(s), its logic will be evaluated upon the completion of any scan including by the selected endpoint(s). The most common way to initiate a scan in SDM is through a scheduled task.

Please reference the following documentation for additional information: Spirion KB Article: Getting Started with Scheduled Tasks

Outcomes

The accuracy and flexibility of Spirion search logic combined with the dynamic capabilities of its scripting engine readily incorporates external functionality into the selective automation handled by SDM Workflows and SDP Playbooks.

In the example script, Python is used to extract scan details into a user-defined log location. Other platform-specific automation examples are explained in separate documents on the Spirion website:

- Applying Box Shield Classifications
- Applying Google Drive Classification & Auditing Share Permissions

Discovery is the crucial precursor to governing sensitive data in any substantial capacity. Spirion centralizes this process by targeting both unstructured and structured data across the sprawl of on-prem and cloud-hosted repositories. By feeding scan details into automated scripts, third-party capabilities may benefit from:

1. Scanning for sensitive data types not included with platform-specific discovery tools while still incorporating control mechanisms from those platforms (via APIs).
2. Creating compound search terms with contextual awareness using Spirion's Sensitive Data Definitions.
3. Using the Spirion Search API to define algorithmically validated custom data types.
4. Standardizing the discovery process across various target repositories by using Spirion to uniformly scan unstructured, structured, or cloud data targets.
5. Consolidating scan results to an enterprise console that reports on matches across the entire sensitive data footprint.

Troubleshooting

Custom script execution can be verified by auditing Spirion scan logs. In addition to confirming a successful connection to the repository environment targeted in a scan, logs will also indicate that the workflow script has been executed for any given match location:

```
"USER ACTION" "Successfully executed script (via workflow) on the file...
```

However, this only indicates that the batch file ran. If the expected outcome of the Python script is not observed – in this example, for instance, if result locations were not written to the specified text file – testing the Python script manually is encouraged.

Manual Python Verification

Console-initiated scans control the Spirion (Windows) agent using the LocalSystem account by default. Since Spirion search logs only indicate if the batch file ran properly, any Python errors will not be reported.

To verify that the Python script executes properly prior to being invoked during a scan, launch a command prompt as the LocalSystem account and call the Python script from that terminal. PsExec provides a convenient means of launching cmd.exe with the appropriate context.

Errors preventing the Python script from completing will be displayed in this terminal. Common causes of concern would be: 1) verifying that the LocalSystem account has access to Python, and 2) ensuring that any module dependencies are also accessible to LocalSystem.