

NetWitness Script Executor

Run Scripts as Response Actions

Helmut Wahrmann

Table of Content

Contents

Summary2

Installation.....2

Configuration.....3

 Configure the NetWitness Script Executor3

 Configure the Response Actions in NetWitness4

Sample Workflow6

 Starting the Script Executor6

 Executing a Response Action7

Log Events.....11

Summary

NetWitness introduced in 12.4 a way to execute Response Actions on various meta data, like IP addresses or host names.

This can be used by Analysts for running actions, like blocking an ip address on a firewall or adding a file to a blocklist, etc.

The first iteration allows interaction with Threatconnect / NetWitness Orchestrator. It is basically calling a web hook that is executing a playbook.

The NetWitness Script Executor makes use of this concept, but instead of specifying a playbook, the name of a script is given, which will then be executed by the script executor.

Installation

A compiled release for Linux and Windows is available in the Release folder on Github at <https://github.com/hwahrman/nwscriptexecutor/releases>.

Unzip it to an appropriate folder and change the configuration options as described in the next section.

The folder service contains a service file for running the program in systemd. Just copy the file to /lib/systemd/system and adapt the installation path and working folder.

Installing the service on Windows is done using the "sc" command:

```
SC CREATE "NetWitness Script Executor" binpath= "C:\Program Files\nwscriptexecutor\ nwscriptexecutor.exe"
```

The program is written in GO. The source code is available in a Github repository at <https://github.com/hwahrman/nwscriptexecutor.git>.

To compile the code, checkout the repository. Make sure that at least Golang 1.22.3 is installed.

To compile for Linux use:

```
$env:GOOS="linux"
go build -o nwscriptexecutor .\options.go .\nwscriptexecutor.go
```

compiling for Windows use:

```
$env:GOOS="windows"
go build -o nwscriptexecutor.exe .\options.go .\nwscriptexecutor.go
```

Make the necessary changes in the config file and start the program from the command line or as a service.

Configuration

Configure the NetWitness Script Executor

The program is configured using a config file in yaml format. It is named config.yml and should reside in the same folder as the program.

It has several sections. The first section configures the Server itself:

```
#
# Script Server specific settings
#
Server:
  Port: 8443 # Listen Port
  ServerCertificate: certs/nwscriptexecutor.crt
  ServerKey: certs/nwscriptexecutor.key
  LogPath: logs\
```

Port – configures the listening port of the Server

ServerCertificate – points to a location for the certificate

ServerKey – points to a location for the private key of the certificate

LogPath – a relative or absolute definition of the path where the program writes log files

Note: The program comes with a self-signed certificate in the “certs” folder. Custom certificates can be provided. If certificates are omitted then the server is started in http mode.

The next section is script specific:

```
#
# Path of the scripts and various settings for script execution
#
Scripts:
  ScriptsFolder: scripts\
  PythonPath:
c:\Users\helmut.wahrmann\AppData\Local\Programs\Python\Python312\python.exe
  PassParmAsJson: true # if set to false the parameters are passed as comma
separated value pairs
```

ScriptsFolder – a relative or absolute path of the folder containing the scripts

PythonPath – The path where the python interpreter can be found

PassParmAsJson – if the parameters should be passed as Json or CSV

Note: New scripts can be added to the script folder and are picked up dynamically. Currently only scripts in python or sh, on Linux, and cmd, on Windows, are supported.

Some samples are provided to show how the parameter can be extracted.

Not everyone knowing the URL should be able to execute scripts, so a basic Password authentication is provided. Currently the password is provided in clear text, therefor the config file should be only accessible to the root user and the user executing the program. In a later version a method will be introduced to store the password encrypted.

```
#
# Authentication
#
Authentication:
  User: helmut
  Password: Password123!
```

Besides writing into a log file, the program can also send syslog on tcp / udp. For customers having NetWitness Logs, a log parser is available in the parser folder. It should be copied to /etc/netwitness/ng/envision/etc/devices/nwscriptexecutor:

```
#
# Configure Syslog Host
#
Syslog:
  Enabled: true
  Host: 192.168.1.53
  Port: 514
  Protocol: udp
```

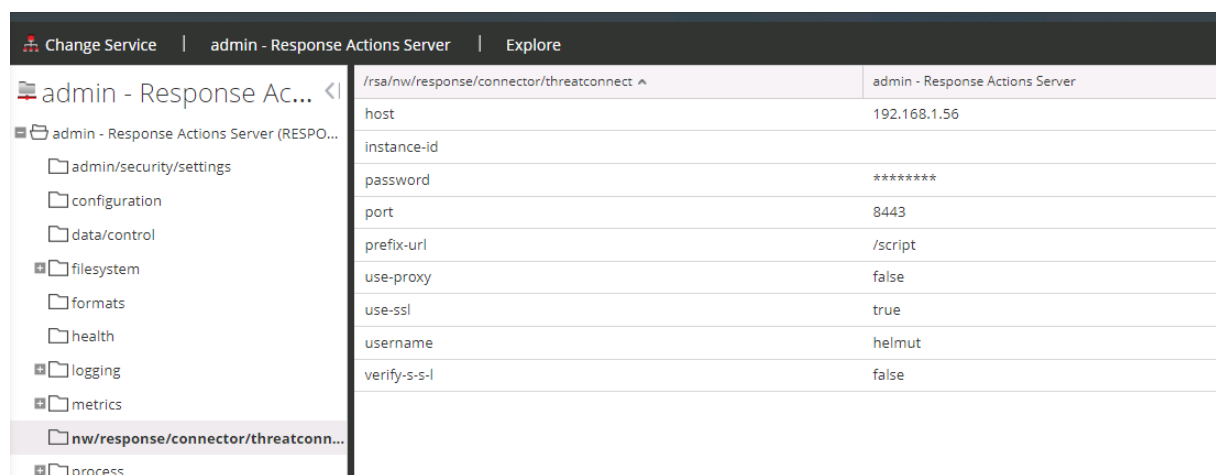
Configure the Response Actions in NetWitness

As mentioned in the Summary, we make use of the ThreatConnect Response Actions Implementation.

Therefor you first need to enable the integration on the Response Actions Server.

In Admin -> Services select the Response Actions Server and navigate to Explore.

From the TreeView select the “nw/response/connector/threatconnect” entry and provide the configuration as described below:



The screenshot shows the NetWitness Admin console interface. The top navigation bar includes 'Change Service', 'admin - Response Actions Server', and 'Explore'. The left sidebar shows a tree view of the configuration hierarchy, with 'nw/response/connector/threatconnect' selected. The main panel displays the configuration for this connector as a table.

/rsa/nw/response/connector/threatconnect		admin - Response Actions Server
host	192.168.1.56	
instance-id		
password	*****	
port	8443	
prefix-url	/script	
use-proxy	false	
use-ssl	true	
username	helmut	
verify-s-s-l	false	

Host – specify the ip address or resolvable fqdn of the server hosting the script executor
instance-id – should be left blank. Not needed in our case
password – specify the same password as in your config.yml
port – the listening port of the server as specified in config.yml
prefix-url – MUST be “/script”
use-proxy – false Proxy support is currently not implemented
use-ssl – true If certificates have been specified in the config.yml
username – the User specified in config.yml
verify-s-s-l – false Default: if custom certs are used, ssl verification can be turned on

When the above definitions are made, then the Response Actions can be defined in Config -> Response Actions.

On “Create” select “ThreatConnect” and instead of a Playbook, specify the name of the script, without an extension.

NETWITNESSPlatform Investigate Respond Users Hosts Files Dashboard Reports

LIVE CONTENT SUBSCRIPTIONS CAPTURE POLICIES POLICIES ESA RULES CUSTOM FEEDS **RESPONSE ACTIONS** MORE

Edit Response Action
Edit a response action of type ThreatConnect by modifying the required fields

ACTION NAME

Block IP Address on Firewall

DESCRIPTION

Block the IP Address on the Firewall

URL PATH

block-ip

PARAMETERS
Add required parameters to forward the data to connector + Add Parameter

PARAMETER KEY	DEFAULT PAR...	PARAMETER T...	PARAMETER L...
ipaddr	Yes		

APPLICABLE META

☒ ip_src ☒ ip_dst ☐ ip.src ☐ ip.dst

In the above example a Response Action with a name of “Block IP Address on Firewall” has been defined.

It is available as a right-click action on Source and/or Destination IP. The script that should be called is named block-ip. This is the sample python script in the scripts folder.

And it should pass a parameter called “ipaddr”, which will be the default parameter and contain the ip address selected.

We now have our first Response Action:

NETWITNESSPlatform Investigate Respond Users Hosts Files Dashboard Reports

LIVE CONTENT SUBSCRIPTIONS CAPTURE POLICIES POLICIES ESA RULES CUSTOM FEEDS **RESPONSE ACTIONS** MORE

Response Actions

+ Create Edit Clone Enable Disable Delete ⚙ View Action History

<input type="checkbox"/>	NAME	DESCRIPTION	CONNECTOR	META KEYS	STATUS	LAST UPDATED	LAST UPDATE
<input type="checkbox"/>	Block IP Address on Firewall	Block the IP Address on the Firewall	ThreatConnect	ip_src, ip_dst, ip.src	Enabled	2024/05/27 15:39:59	helmut

Sample Workflow

Starting the Script Executor

On a Linux server, I have the following config.yml:

```
#
# Script Server specific settings
#
Server:
  Port: 8443      # Listen Port
  ServerCertificate: certs/helmut-local.helmut.lab.crt
  ServerKey: certs/helmut-local.helmut.lab.key
  LogPath: logs/

#
# Path of the scripts and various settings for script execution
#
Scripts:
  ScriptsFolder: scripts/
  PythonPath: /usr/bin/python3
  PassParmAsJson: true # if set to false the parameters are passed as comma separated value pairs

#
# Authentication
#
Authentication:
  User: helmut
  Password: Password123!

#
# Configure Syslog Host
#
Syslog:
  Enabled: true
  Host: 192.168.1.53
  Port: 514
  Protocol: tcp
```

I am running with a custom certificate, write the logs into the logs folder, have basic authentication in place and I am sending syslog via tcp to my Log Decoder on 192.168.1.53.

My scripts folder contains 2 scripts:

```
[root@NWVLC scripts]# ll
total 8
-rw-r--r--. 1 root root 159 May 27 13:33 block-ip-bash.sh
-rw-r--r--. 1 root root 887 Apr 26 13:59 block-ip.py
[root@NWVLC scripts]#
```

We have defined “block-ip” in our Response Actions. So it will execute the Python script.

First we start the service using:

```
systemctl start nwscriptexecutor
```

This will produce following log lines:

```
[root@NWWVLC nwscriptexecutor]# tailf logs/nwscriptexecutor.log
2024/05/27 13:31:20.622635 Welcome to NetWitness Script Executor v.1.0 GPL v3
2024/05/27 13:31:20.622740 Copyright (C) 2024 Helmut Wahrmann.
2024/05/27 13:31:20.622745 Setting up Syslog connectivity
2024/05/27 13:31:20.623438 Building Script Cache
2024/05/27 13:31:20.623481 Found 2 scripts
2024/05/27 13:31:20.623487 Starting up the HTTP(S) Listener
```

It found 2 scripts in the scripts folder, which are cached now, started the HTTPS listener and is now waiting on incoming requests on 8443.

Executing a Response Action

From the Investigation, or alternatively an Incident/Alert in Respond, I am now selecting an Ip Address and select “Quick Actions”(Not sure, why they didn’t name this “Response Actions”):

↔

2024/05/27 14:36

Events Meta

1,376 Events (Desc)

Summary List

Meta Group

↓

⚙️

Enter text to be found.

255.255.255.255 (18)

99.99.99.60 (1)

192.168.1.50 (11)

192.168.255.255 (1)

192.168.1.53 (8)

192.168.1.2 (7)

199.232.210.172 (6)

192.168.1.1 (5)

192.168.1.8 (5)

20.72.205.209 (4)

151.101.22.172 (4)

192.178.50.67 (4)

23.48.246.240 (3)

40.119.6.228 (3)

52.191.219.104 (3)

192.168.1.51 (3)

224.0.0.252 (3)

239.255.255.250 (3)

192.168.1.255 (2)

192.178.50.74 (2)

CONTEXT HIGHLIGHTS

0 INCIDENTS

0 ALERTS

0 LISTS

- CRITICALITY

- ASSET RISK

0 TI

- NETWORK EXP...

External Lookup

Copy Value

Copy Statement

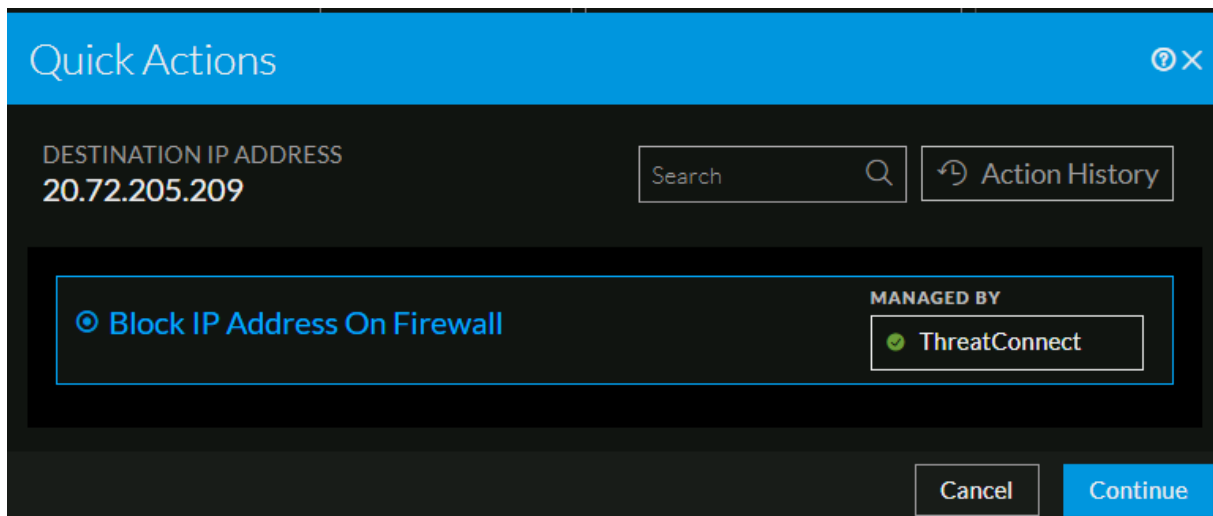
Live Lookup

Context Lookup

Quick Actions

Pivot to Investigate > Hosts/Files

This shows me all my actions, which are defined for the selected ip address:



I am entering a comment and press confirm.

This results in an invocation of the Web Hook and it sends a http POST command to the server defined in the ThreatConnect integration.

In my example it will be:

```
https://192.168.1.56/script/block-ip
```

The parameters submitted in the post where those:

```
{"nw-user":"helmut","nw-comment":"Blocking the suspicious ip address","ipaddr":"20.72.205.209","nw-actionId":"665488af7ed7bf4e3d52956e","nw-actionName":"Block IP Address on Firewall"}
```

Nw-user – The name of the NetWitness User submitting the request.

nw-comment – The comment, which was added in the form when submitting the action

ipaddr – That is the default parameter, which we defined in the action and contains the ip address, that was selected.

nw-actionId – the id of the response action

nw-actionName – the name of the Response Action

The Log has now additional lines, starting from 13:43:31:

```
2024/05/27 13:43:31.756092 Received request to execute script block-ip
2024/05/27 13:43:31.756179 Getting request body
2024/05/27 13:43:31.756234 Script requested: block-ip.py with arguments: {"nw-user":"helmut","nw-comment":"Blocking the suspicious ip address","ipaddr":"20.72.205.209","nw-actionId":"665488af7ed7bf4e3d52956e","nw-actionName":"Block IP Address on Firewall"}
2024/05/27 13:43:36.953311 Script output: Parameters are in JSON Format
2024/05/27 13:43:36.953431 Script output: IP address: 20.72.205.209
2024/05/27 13:43:36.953451 Script output:
```

The server acknowledges that it received a request to execute the script “block-ip”.
it is getting the request body, which contains the parameters, described above.
And the lines starting with “Script output:” have the output of the script itself.

```
import time
import sys
import json
from json.decoder import JSONDecodeError

# Arguments to the script can be passed as JSON (in config PassParmAsJson:
# true) or separated by Comma
# The below logic shows the handling of both methods
#
# The sample assumes that a Response Action has been defined, which sends the
# ip address as "ipaddr"
#
json_error = False
ip = ""
try:
    args = json.loads(sys.argv[1])

    print("Parameters are in JSON Format")

    # Now you can access the parameters as sent by NetWitness Respond Action
    # Server
    # e.g. accessing ipaddr
    ip = args["ipaddr"]
    print("IP address: ", ip )
except JSONDecodeError:
    print("Parameters are in CSV Format")
    json_error = True

if json_error:
    pos = 0
    args = sys.argv[1].split(sep=",")
    for arg in args:
        if arg == "ipaddr":
            ip = args[pos+1]
            print("IP address : ", args[pos+1])
            break
        pos = pos + 1

# Insert your logic here to do whatever is needed with the ip address

# Simulate longer execution time of script
time.sleep(5)
exit(0)
```

The script tests first if the parameters are in JSON format and then prints out the IP Address. If it fails the JSON Test, it assumes that the parameters have been passed as CSV and will extract the IP address from the CSV.

After that the real logic would start, where, depending on the integration a Python SDK or REST API might be used to interface with the third party.



In the sample script, I just put a sleep statement to simulate some execution.

Most of the vendors provide either a Python SDK or samples on how to use their REST API.

Google Gemini or ChatGPT might be your friend:



show me a sample script of using the palo alto api to block an ip address from inside python

Vorschläge anzeigen  



Here's a sample Python script demonstrating how to block an IP address on a Palo Alto firewall using the PAN-OS API:

Python

```
# Import libraries
import requests
from urllib3.exceptions import InsecureRequestWarning

# Set firewall details (replace with your values)
PAN_IP = "192.168.1.1" # Firewall IP address
API_KEY = "YOUR_API_KEY" # Replace with your generated API key
API_VERSION = "9.1" # Update based on your PAN-OS version
IP_TO_BLOCK = "10.0.0.10" # IP address to block
SECURITY_RULE_NAME = "Block_IP_Rule" # Name for the security rule

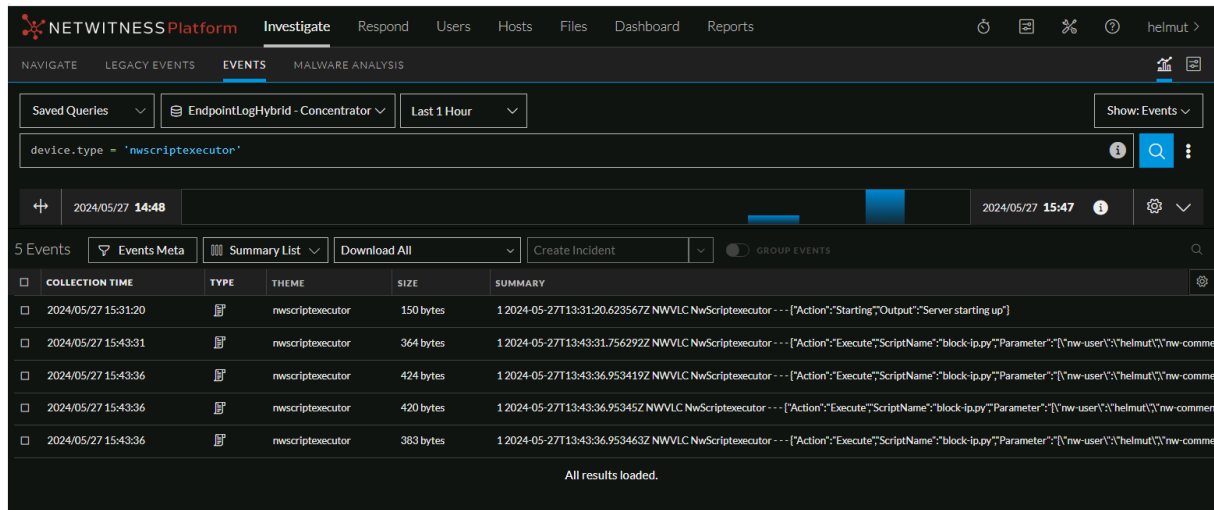
# Disable insecure request warning (if using self-signed certificate)
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

# Build URL for address object creation
url = f"https://{PAN_IP}/restapi/v{API_VERSION}/Objects/Addresses"

# Address object data
address_data = {
    "entry": [
        {
            "@name": "Blocked_IP",
            "ip": IP_TO_BLOCK,
            "description": "IP blocked by script"
        }
    ]
}
```

Log Events

We have enabled sending syslog to our Log Decoder. So we have received 5 events:

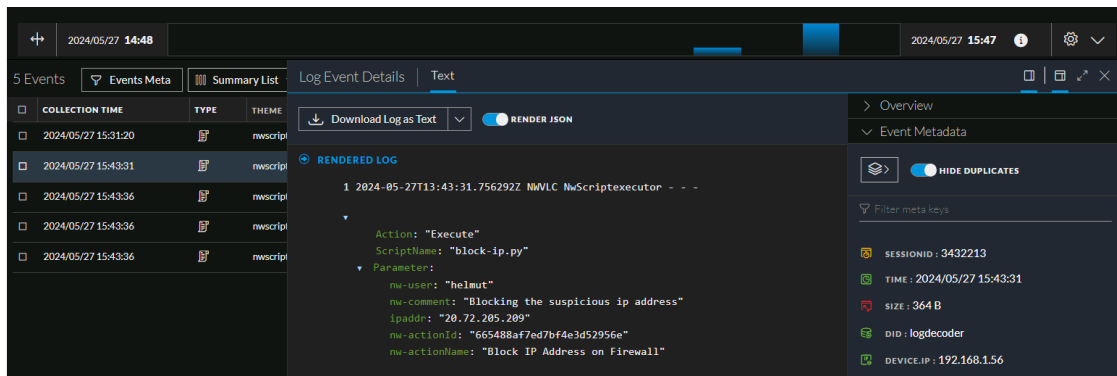


CHECKBOX	COLLECTION TIME	TYPE	THEME	SIZE	SUMMARY
<input type="checkbox"/>	2024/05/27 15:31:20	nwscriptexecutor		150 bytes	1 2024-05-27T13:31:20.623567Z NWVLC NwScriptexecutor --- [Action:"Starting","Output":Server starting up]
<input type="checkbox"/>	2024/05/27 15:43:31	nwscriptexecutor		364 bytes	1 2024-05-27T13:43:31.756292Z NWVLC NwScriptexecutor --- [Action:"Execute","ScriptName":"block-ip.py","Parameter":{"nw-user":"helmut","nw-comment":
<input type="checkbox"/>	2024/05/27 15:43:36	nwscriptexecutor		424 bytes	1 2024-05-27T13:43:36.953419Z NWVLC NwScriptexecutor --- [Action:"Execute","ScriptName":"block-ip.py","Parameter":{"nw-user":"helmut","nw-comment":
<input type="checkbox"/>	2024/05/27 15:43:36	nwscriptexecutor		420 bytes	1 2024-05-27T13:43:36.95345Z NWVLC NwScriptexecutor --- [Action:"Execute","ScriptName":"block-ip.py","Parameter":{"nw-user":"helmut","nw-comment":
<input type="checkbox"/>	2024/05/27 15:43:36	nwscriptexecutor		383 bytes	1 2024-05-27T13:43:36.953463Z NWVLC NwScriptexecutor --- [Action:"Execute","ScriptName":"block-ip.py","Parameter":{"nw-user":"helmut","nw-comment":

All results loaded.

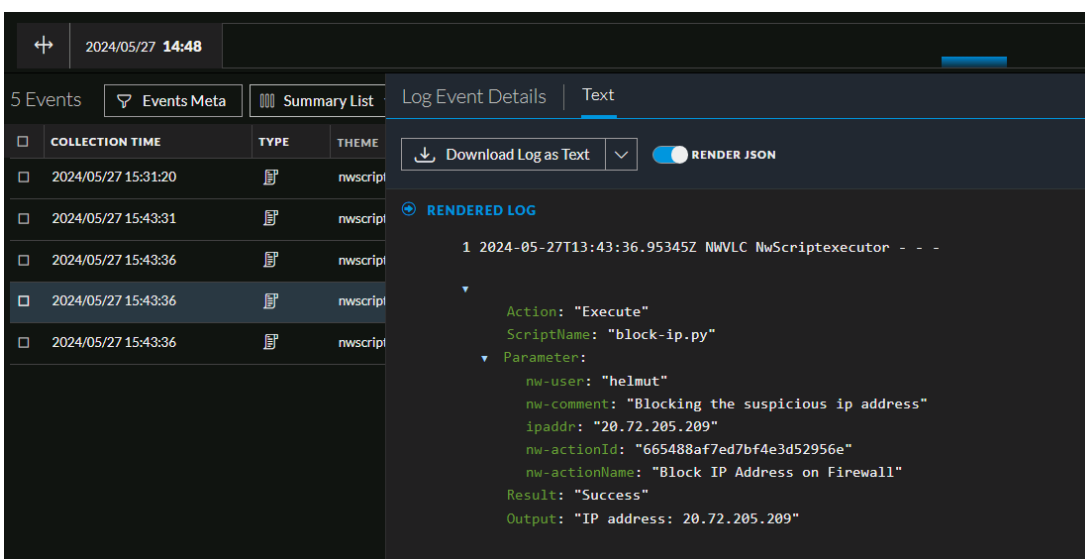
The first event is telling us that the Server was started.

Then we have an entry for the execution of the script with all the parameters:



```
1 2024-05-27T13:43:31.756292Z NWVLC NwScriptexecutor ---  
  
  Action: "Execute"  
  ScriptName: "block-ip.py"  
  Parameter:  
    nw-user: "helmut"  
    nw-comment: "Blocking the suspicious ip address"  
    ipaddr: "20.72.205.209"  
    nw-actionId: "665488af7ed7bf4e3d52956e"  
    nw-actionName: "Block IP Address on Firewall"
```

Followed by an event for every line that the script prints in the "Output"



```
1 2024-05-27T13:43:36.95345Z NWVLC NwScriptexecutor ---  
  
  Action: "Execute"  
  ScriptName: "block-ip.py"  
  Parameter:  
    nw-user: "helmut"  
    nw-comment: "Blocking the suspicious ip address"  
    ipaddr: "20.72.205.209"  
    nw-actionId: "665488af7ed7bf4e3d52956e"  
    nw-actionName: "Block IP Address on Firewall"  
  Result: "Success"  
  Output: "IP address: 20.72.205.209"
```