NCDT_Rule_Manager_v2

This script is intended to simplify and automate the process of editing and updating rules on the CDT. We created this script with the primary purpose to to simplify the process of rule updates and to tune analytical efforts to better suit our version of CANES and threat model. Please see the below guide on how to utilize this script, and for a technical breakdown of the functions and employed methods (i.e. AWK, regex, loops and variables, etc.) jump here to read the technical details.

ⓘ For support please email

CWT3 Moffett: robert.o.moffett03.mil@us.navy.mil
CWT2 Lowry: benjamin.d.lowry2.mil@us.navy.mil

Setup

1. Open WinSCP and connect to SecO. (commonly 192.168.100.20)
2. Drag and drop script file into /home/cdtadmin/

> **Note:**
>
> The script can also be run from the /tmp/ directory if desired.

3. ssh into SecO using PowerShell

```
ssh cdtadmin@192.168.100.20
```

4. Run the following command:

```
chmod +x NCDT_Rule_Manager
```

5. The script is ready to be run.

Usage

1. Open WinSCP and connect to SecO.
2. Drag and drop the desired rules files (ETPRO or Anomali) to the /tmp/ directory on SecO
3. ssh into SecO using PowerShell

```
ssh cdtadmin@192.168.100.20
```

4. Run the following commands to verify the rules files transferred and the script is in the proper location.

```
ls
```

```
ls /tmp
```

5. The script **MUST** be run with root privileges. To do this run the following command:

```
sudo su
```

6. To launch the script run

```
./NCDT_Rule_Manager
```

> **You may also use:**
>
> ```
> sudo ./NCDT_Rule_Manager
> ```

7. From the script menu select the option you would like to run by typing in the corresponding number.

    1: Update ETPRO Rules.

    2: Update Anomali Rules.

    3: Update Both Rule Sets.

    4: Remove Unnecessary Rules.

    5: Editing Tool.

    6: Quit.

8. Enter required input / wait for script to complete.

9. Select option '6' at the main menu to exit.

Options Breakdown

Option 1: Update ETPRO Rules

1. Download ETPRO rules from SAILOR (go to MNP quick-links).

2. Transfer rules from NIPR to CDT Laptop.

3. Open WinSCP and transfer rules file to /tmp/ directory on SecO.

4. Ensure ETPRO rules file is moved to /tmp/ directory

> ⓘ **The ETPRO file is usually named etpro.rules.tar.gz**

5. Run the script according to the above procedure.

6. Select option '1' at the main menu and the script will automatically update ETPRO rules.

Option 2: Update Anomali Rules

1. Download Anomali rules from DoD SAFE.

2. Transfer rules from NIPR to CDT Laptop.

3. Open WinSCP and transfer the rules file to the /tmp/ directory on SecO.

4. Ensure the ETPRO rules file is moved to /tmp/ directory

> ⓘ **The anomali file is usually named DoD SAFE<random characters>.zip**

5. Run the script according to the above procedure.

6. Select option '2' at the main menu and the script will automatically update Anomali rules.

Option 3: Update Both Rule Sets

1. Download ETPRO rules from SAILOR.
2. Download Anomali rules from DoD SAFE.
3. Transfer rules from NIPR to CDT Laptop
4. Open WinSCP and transfer rules file to /tmp/ directory on SecO.
5. Ensure ETPRO rules and Anomali rules files are moved to /tmp/ directory

> ⓘ **The ETPRO file is usually named etpro.rules.tar.gz**

> ⓘ **The anomali file is usually named DoD SAFE<random characters>.zip**

6. Run the script according to the above procedure.
7. Select option '3' at the main menu and the script will automatically update both rule sets.

Option 4: Removing Unnecessary Rules

> ⚠ **!! WARNING: !!**
>
> This option is used to remove rules from the all.rules file. Ensure you are 100% sure you would like to remove the rule before selecting this option.

1. Run the script according to the above procedure.
2. Select option '4' at the main menu (This will bring you to a secondary menu).

Select option '1' to remove a rule by SID.

1. When asked enter the SID/s for the rule/s you would like to remove.

> 📋 **You may enter up to 10 SIDs at one time.**
>
> To remove multiple rules separate the SIDs by “,” (i.e. 1234567,7654321).

2. The script will automatically remove the desired rules from the all.rules file.

Select option '2' at the menu to exit to the main menu.

> 🖉 **Note:**
>
> At the beginning of running this section of the script, a copy of the current all.rules file is saved to /tmp/backup and dated. To undo changes run the following command:

```
mv /tmp/backup/<desired file> /opt/so/rules/nids/all.rules
```

Option 5: Editor Tool

This option is designed as a way to easily combat alerts that generate thousands of alerts daily. There are two methods for this. The first method pulls the rule from the all.rules file and allows for manual edits. The second method uses AWK to find the correct location within the rule and automatically adds thresholding to it.

1. Run the script according to the above procedure.
2. Select option '5' at the main menu (this will bring you to the editor menu).

Method 1: Manual Edit

> **Options 1, 2, and 3 in the Editing Tool menu can be used for manual rule edits.**

Select option '1' at the editor menu to save a rule to a text file for manual editing.

1. When asked enter the SID/s for the rule/s you would like to pull.
2. The script will automatically produce a text file with your desired rule.
3. The rule may then be opened in a text editor of your choice and edited to reflect your desired changes.

> **The saved rule can be found at: /tmp/saverule.txt**

Select option '2' at the editor menu to re-implement a rule to the all.rules file.

1. The script will automatically upload saverule.txt back into the all.rules file.

Select option '3' at the editor menu to generate a threshold of your choice.

1. Follow the prompts to create your custom threshold.
2. The script will automatically produce a text file with your threshold.
3. The threshold string can then be copied and pasted to your desired rule. (Instructions are included in the text file).

> **The custom threshold can be found at: /tmp/threshold.txt**

Method 2: Auto-Threshold

> **Option 4 in the Editing Tool menu can be used for automatically adding thresholds.**

Select option '4' at the editor menu to auto-threshold a rule.

1. When asked enter the SID/s for the rule/s you would like to threshold.
2. Follow the prompts to create your threshold.
3. The script will automatically add your threshold to the desired rule/s.

Select option '5' at the editor menu to exit to the main menu.

Option 6: Quit

Select option '6' at the main menu to exit the script.

Testing Mode

To modify the script for testing or training, make the following modifications:

1. Open the script file in your text editor of choice. (Notepad, Notepad++, vim, etc.)
2. Add a '#' before line 420 (save_files function call).
3. Add a '#' before line 109 (update_anomali function).
4. Add "/tmp/AllRules.txt" to "$files" array on line 460 (clear_files function).
5. Remove "/tmp/etpro.rules.tar.gz" from "$files" array on line 480 (save_files function).

6. Save the most recent versions of ETPRO and Anomali rules to /tmp/ directory.
7. The script can now be run continuously without generating excess files.

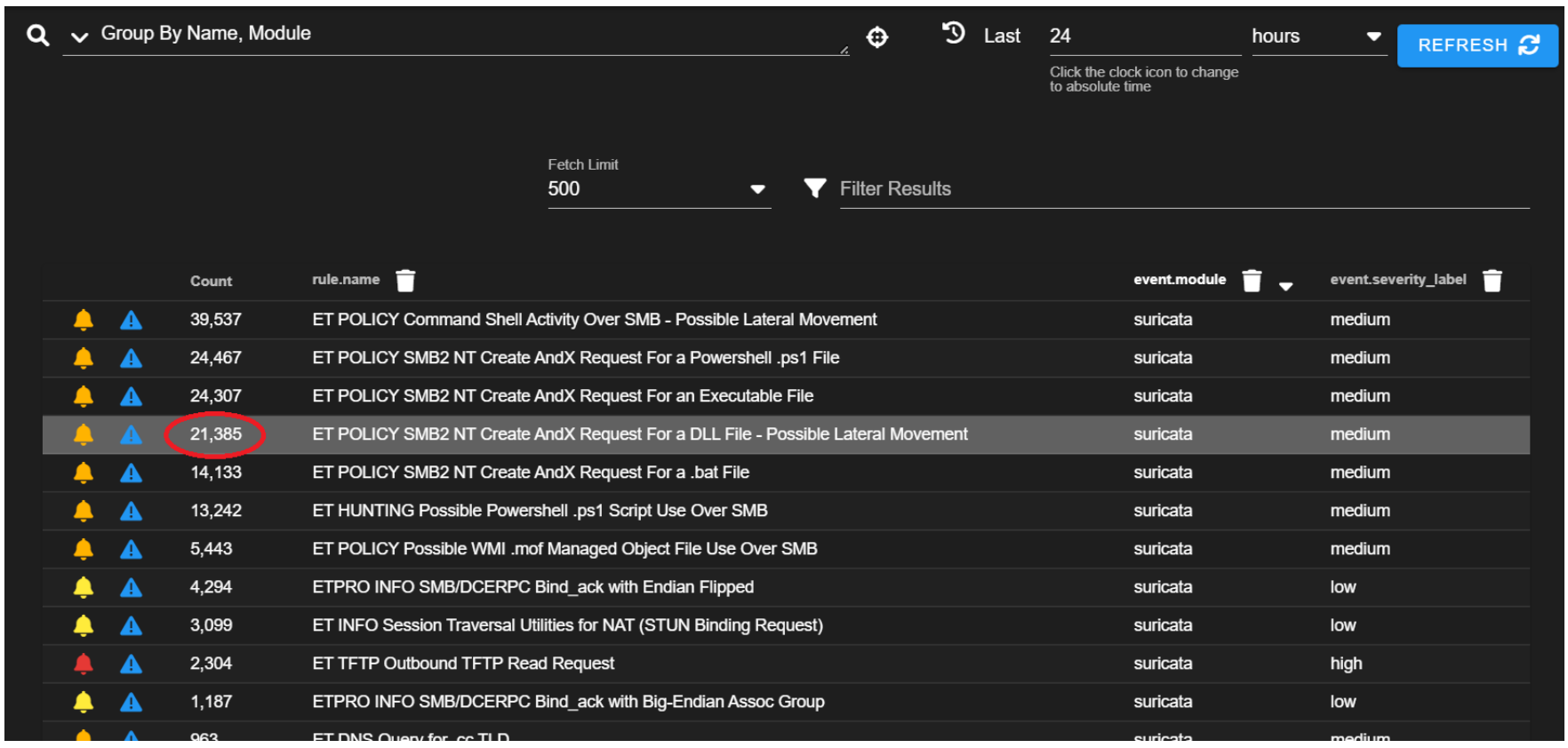
> **Note: To undo reverse these steps.**

Walkthrough

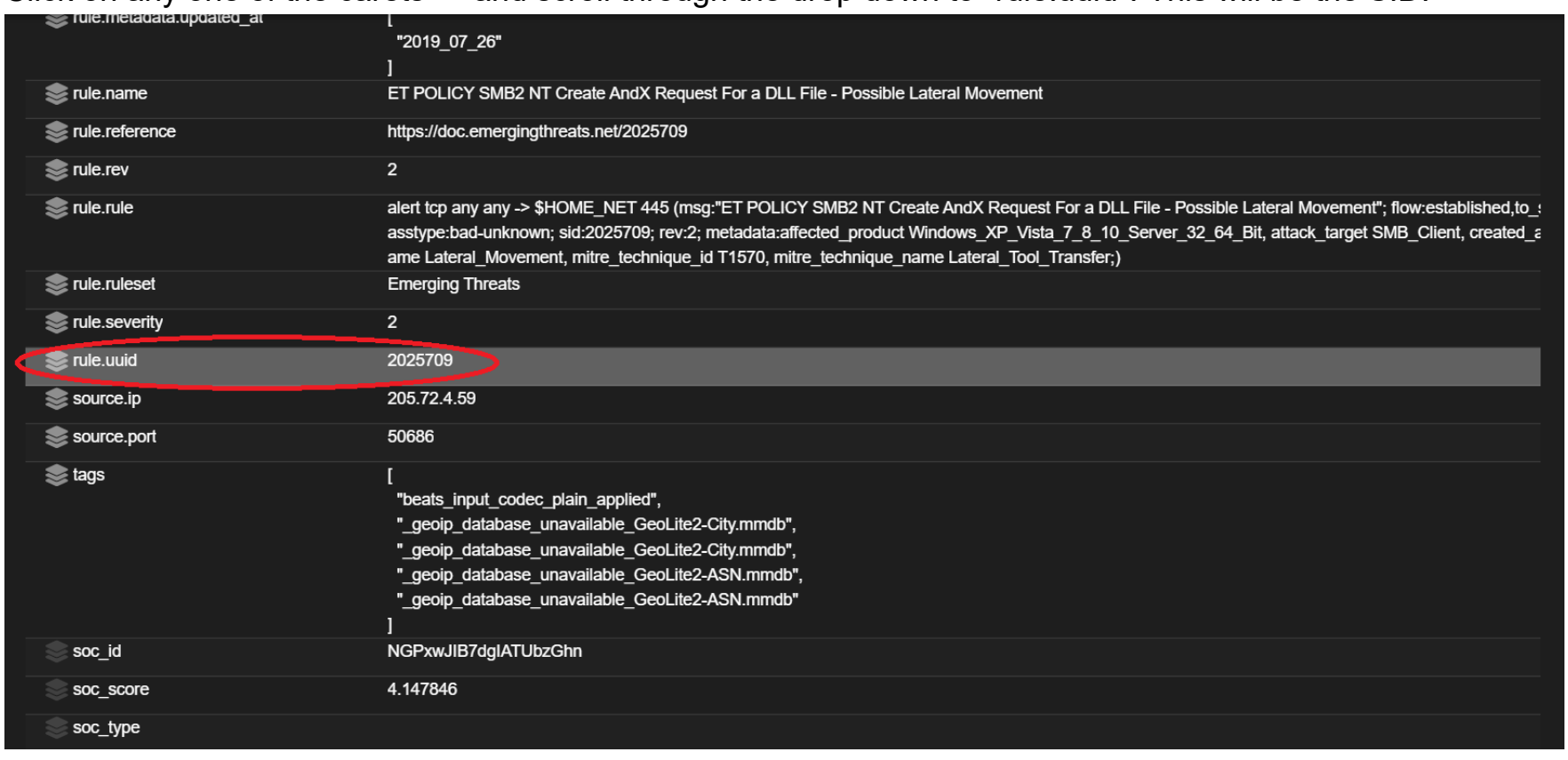
Auto-Threshold

The following procedures are intended to explain the use of the "Auto-Threshold" feature of the "Editing Tool". For a technical breakdown see Here.

1. Identify the rule to be removed. For demonstration purposes we will use: "ET POLICY SMB2 NT Create AndX Request For a DLL File - Possible Lateral Movement"
2. To find the rule SID click on "Quick Drilldown" in the alerts tab of the Security Onion homepage.



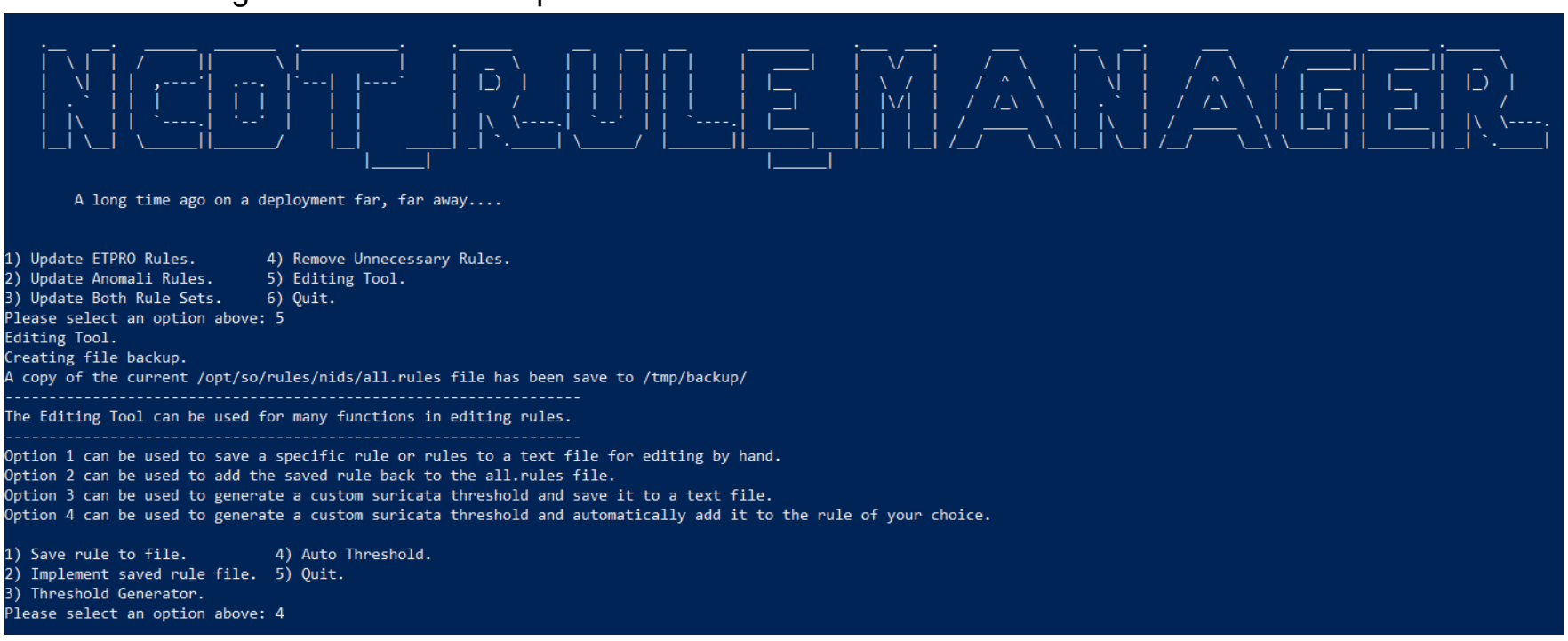
3. Click on any one of the carets '>' and scroll through the drop down to "rule.uuid". This will be the SID.



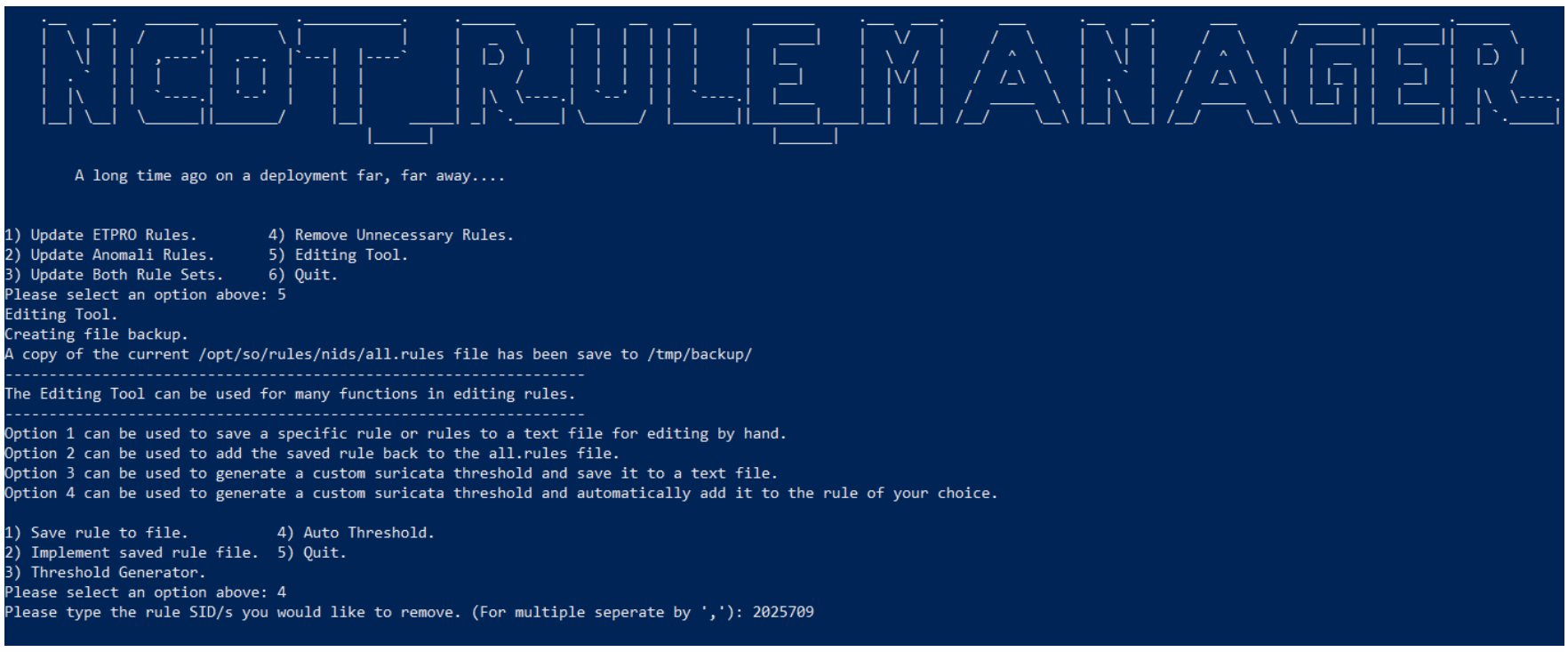
4. Start the script and wait for the main menu to appear.

5. Select option '5' at the menu. (Editing Tool Rules)

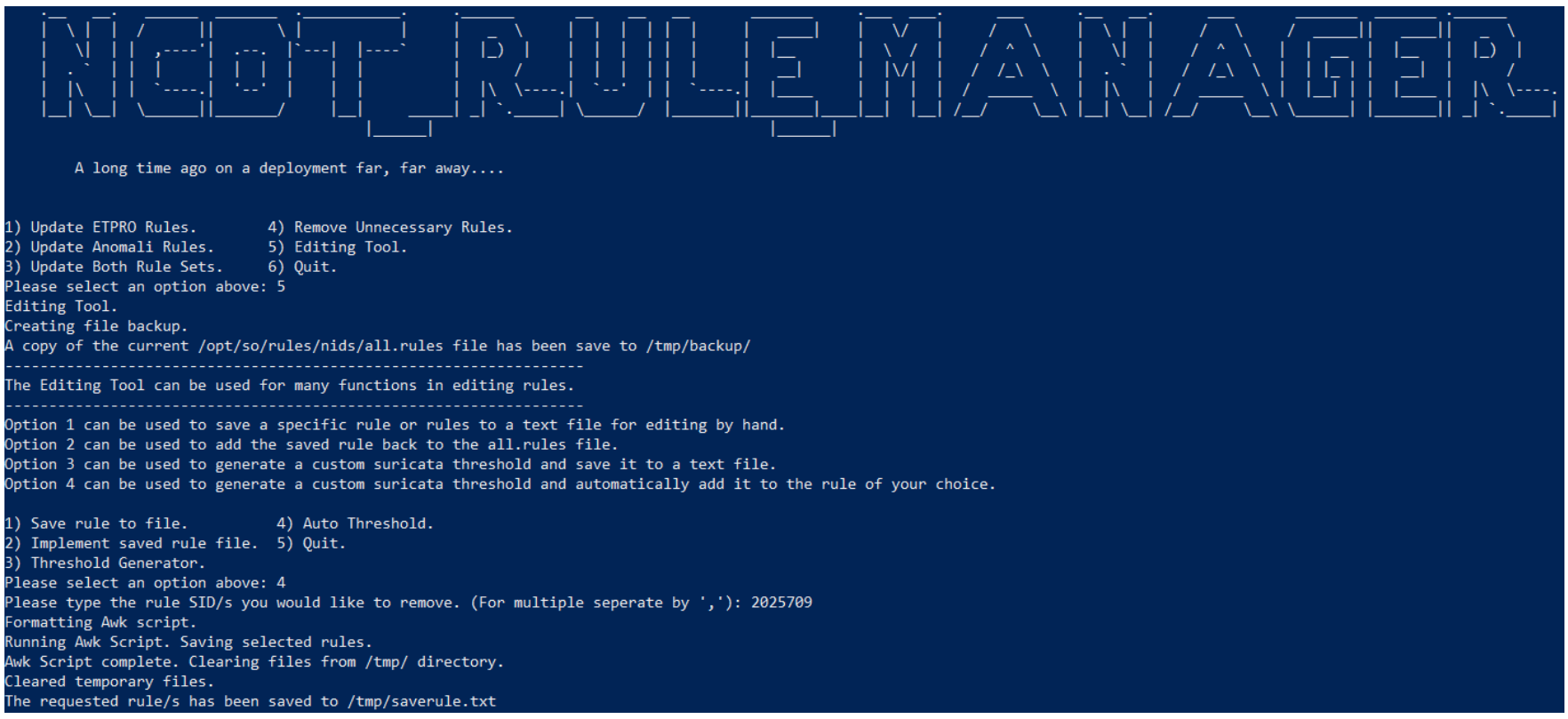
6. A file backup of /opt/so/rules/nids/all.rules will be taken at this point. If one has been taken already, you will be prompted if you would like to make another.
7. From the "Editing Tool" menu select option '4' Auto-Threshold.



8. When prompted enter the desired rule SID/s.



9. The script will automatically format and run a short AWK script to save the desired rule/s for editing.



10. The script will then prompt you to create your custom threshold.

11. The first option is your threshold "type". For this demonstration we will use a "backoff" type threshold.

```
The type of threshold determines the action to be taken by suricata handling the alert. The 4 types are listed below:
--------------------------------------------------------------------------------
Threshold - Sets a minimum number of alerts before one will be displayed.
Limit - Sets a maximum amount of alerts that will be displayed.
Both - Sets both a minimum and maximum number of alerts, combines Limit and Threshold.
Backoff - Uses a backoff algorithm to reduce alerts displayed.
--------------------------------------------------------------------------------
More Information can be found in the Suricata documentation. https://docs.suricata.io/en/latest/rules/thresholding.html
--------------------------------------------------------------------------------\n
1) Threshold.
2) Limit.
3) Both.
4) Backoff.
Please select the type of threshold you would like: 4
```

12. The second option for the threshold is "track". For this demonstration we will track "by_src".

```
The track option determines how suricata will track the alerts for thresholding. The 5 types are listed below
--------------------------------------------------------------------------------
by_src - Thresholds alerts based on source.
by_dst - Thresholds alerts based on destination.
by_rule - Thresholds alerts based on rule.
by_both - Thresholds alerts based on source and destination.
by_flow - Thresholds alerts based on flow.
--------------------------------------------------------------------------------
More Information can be found in the Suricata documentation. https://docs.suricata.io/en/latest/rules/thresholding.html
--------------------------------------------------------------------------------

1) By Source.       3) By Rule.         5) By Flow.
2) By Destination.  4) By Both.
Please select the tracking you would like to use: 1
```

13. The third option to set is "count". For this demonstration we will set a count of "1".

```
Count sets the number of alerts to be used for the threshold. (min, max, starting point for backoff).
--------------------------------------------------------------------------------
More Information can be found in the Suricata documentation. https://docs.suricata.io/en/latest/rules/thresholding.html
--------------------------------------------------------------------------------

Please enter a value for count: 1
```

14. The last option to set is "multiplier". For this demonstration we will set a multiplier of "10".

```
Multiplier sets the multiple for backoff alerts to display. (i.e. count 1, multiplier 10, -> 1,10,100,1000,etc.)
--------------------------------------------------------------------------------
More Information can be found in the Suricata documentation. https://docs.suricata.io/en/latest/rules/thresholding.html
--------------------------------------------------------------------------------

Please enter a value for multiplier: 10
```

> **Note:**
>
> This means the threshold will use a backoff algorithm to threshold alerts. We will see alerts from source IPs only at multiples of 10 starting with 1. (i.e. 1st, 10th, 100th, 1000th, etc...)

15. The script will automatically format and run a short AWK script to threshold the desired rule/s.

```
Multiplier sets the multiple for backoff alerts to display. (i.e. count 1, multiplier 10, -> 1,10,100,1000,etc.)
--------------------------------------------------------------------------------
More Information can be found in the Suricata documentation. https://docs.suricata.io/en/latest/rules/thresholding.html
--------------------------------------------------------------------------------

Please enter a value for multiplier: 10
Adding Auto-Threshold.
Calling ThresholdAwk.awk script.
Complete. Re-implementing rule.
Complete. exiting to menu.
1) Save rule to file.          4) Auto Threshold.
2) Implement saved rule file.  5) Quit.
3) Threshold Generator.
Please select an option above:
```

16. When complete, you will be returned to the "Editing Tool" menu. Select option '2' to exit to the main menu.

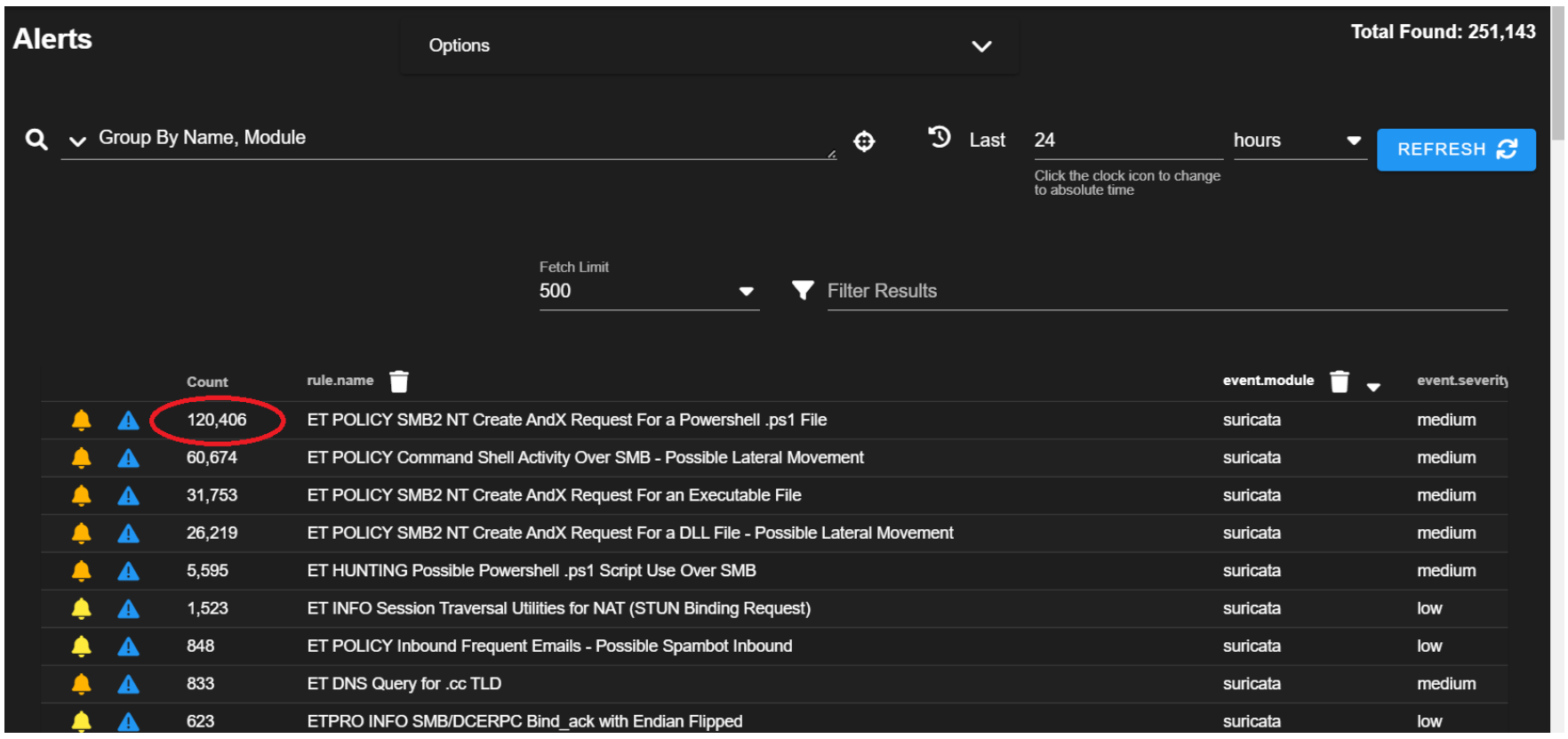
Removing Unnecessary Rules

The following procedures are intended to explain the use of the Remove Unnecessary Rules function. For a technical breakdown see Here.

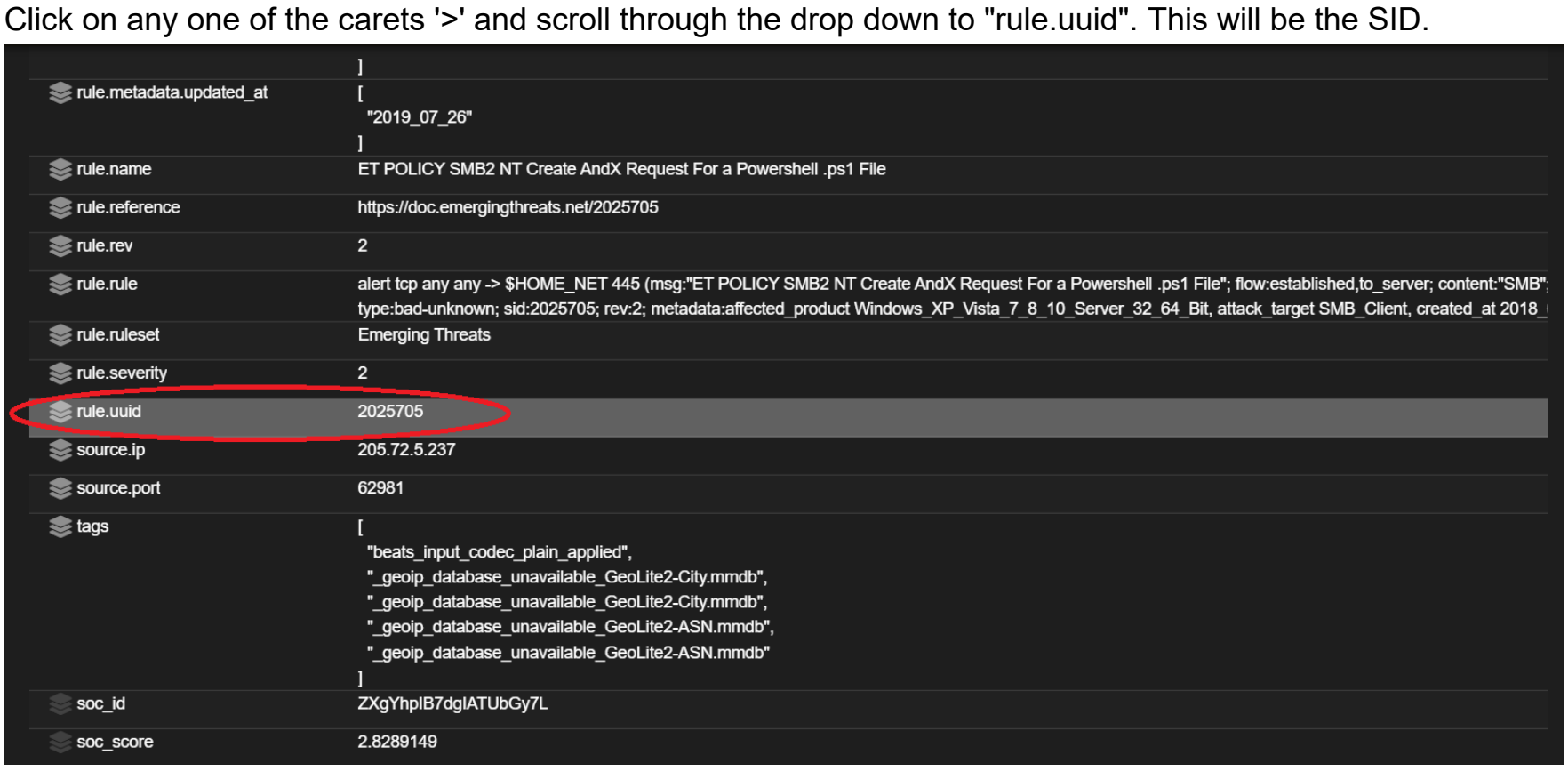
> **⚠ Warning:**

It is recommended to use this option as a last resort or for rules that add no value to analysis.

1. Identify the rule to be removed. For demonstration purposes we will use: "ET POLICY SMB2 NT Create AndX Request For a Powershell .ps1 File"
2. To find the rule SID click on "Quick Drilldown" in the alerts tab of the Security Onion homepage.



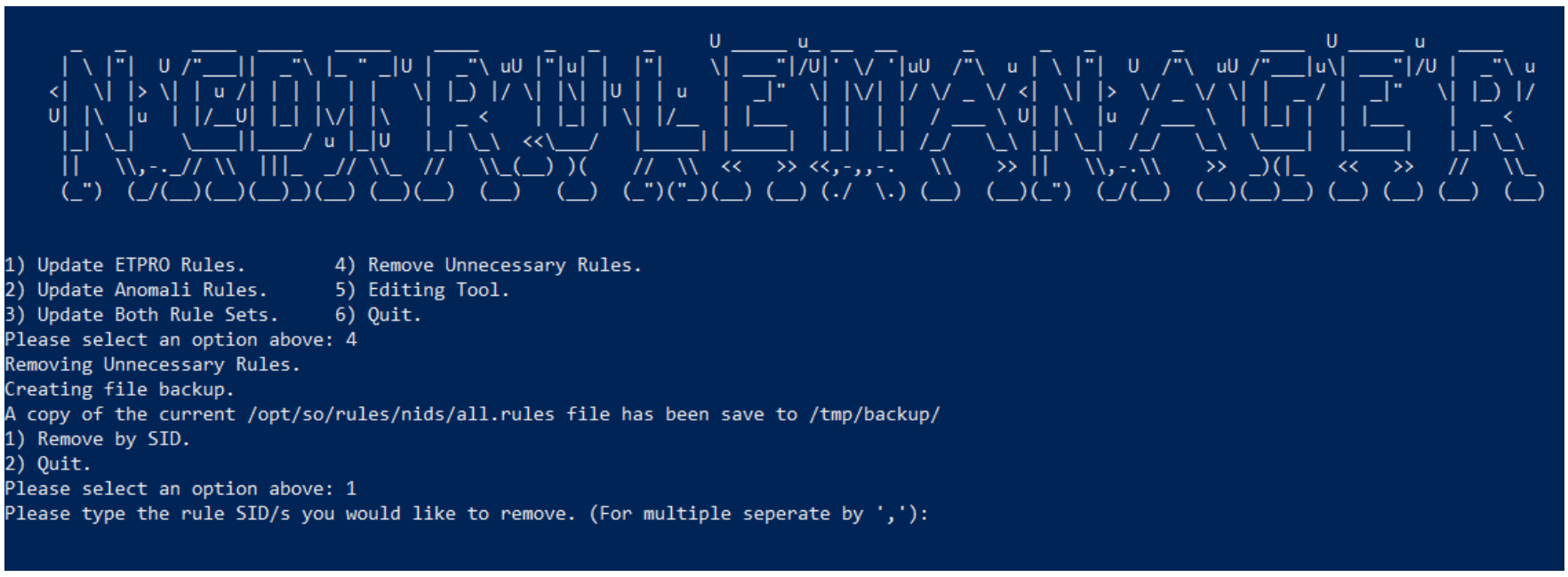
3. Click on any one of the carets '>' and scroll through the drop down to "rule.uuid". This will be the SID.



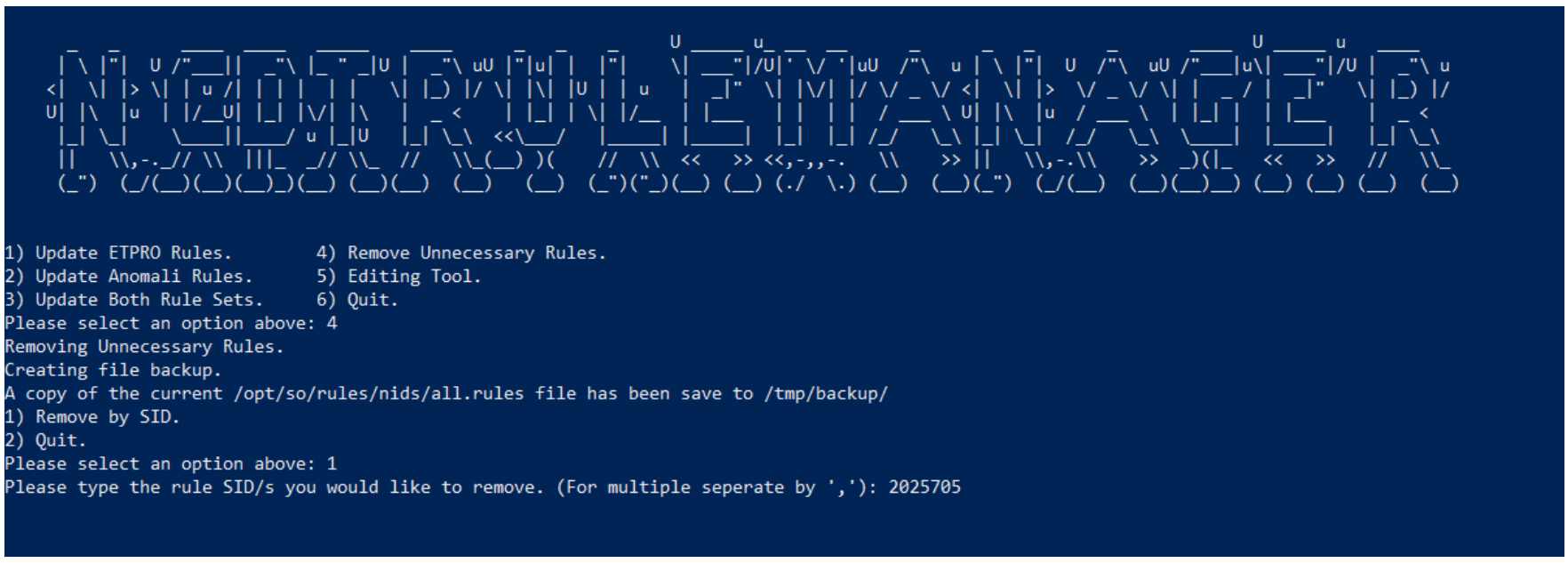
4. Start the script and wait for the main menu to appear.
5. Select option '4' at the menu. (Remove Unnecessary Rules)

6. A file backup of /opt/so/rules/nids/all.rules will be taken at this point. If one has been taken already, you will be prompted if you would like to make another.

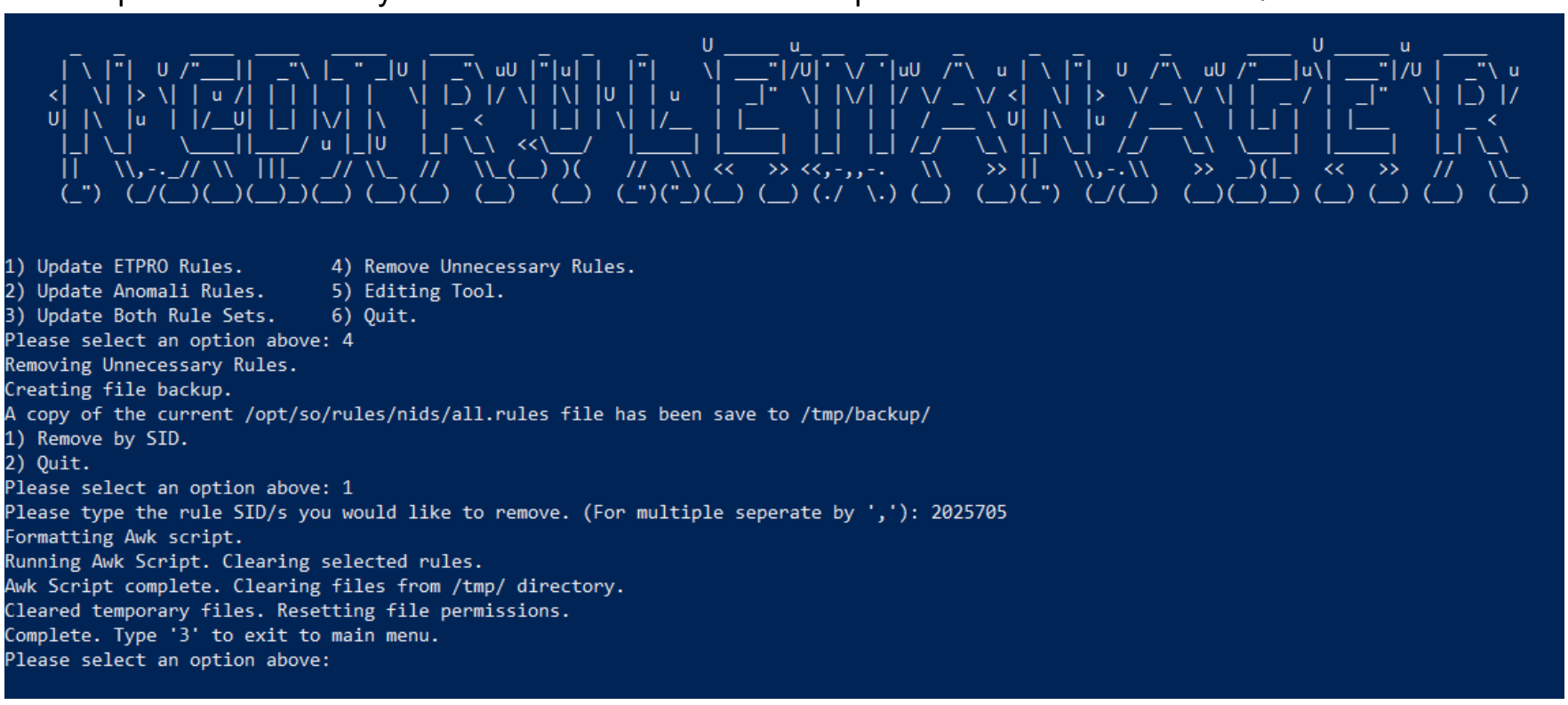
7. From the internal menu select option '1'. "Remove by SID".



8. When prompted enter the desired rule SID/s.



9. The script will automatically format and run a short AWK script to remove the desired rule/s.



10. When complete, you will be returned to the internal menu. Select option '2' to exit to the main menu.

Technical Breakdown

This section covers in depth breakdowns of the code used in each section and its function both individually and in the script as a whole. Included are the breakdowns of small sections and commands, as well as each main code block for review. Additionally the full code is attached at the bottom to review in its entirety.

Global Variables

The "global variables" are a set of variables declared outside of any specific function, allowing them to be called or used as needed in any section of the code.

Below is the list of global variables used for review:

```
# global variables
LOGFILE="log.log"
i=0
date=$(date +%Y-%m-%d-%R)
type=""
track=""
count=""
seconds=""
multiplier=""
num="[0-9]+"
```

Logging

Logging for the script is achieved through use of the "exec" and "tee" commands.

```
exec > >(tee -a "$LOGFILE") 2>&1
echo "Logging started on $(date)"
```

The exec command is used to replace the current shell with a command. To create a log file we redirect "exec" to the "tee" command. "tee" is used to write to STDOUT and one or more files simultaneously. For the script we write to "$LOGFILE" (one of our global variables) and to STDOUT. To ensure we capture both successful commands and errors, we redirect STDERR to STDOUT.

Logfile (log.log) can be found in the directory the script is run.

Main Menu

The main section of the script comes from the "menu" function. This function is used to generate the main menu and make several different function calls based on the users input.

The menu is built using three main commands. A while loop, the "select" command, and the "case" command. The while loop is used to refresh the menu every time it is called. The "select" and "case" commands act as the functional elements of the menu. "select" is used to read the users input, and "case" is used to match that input to a set of actions.

The "select" command is used to create a simple menu. This is done by displaying numbered items from a list and prompting the user for input of a corresponding number, the selected item is then stored in a variable.

```
select variable in list
```

The script features a variation on this by instead of using a list directly in the command, we pass the "select" command an array of strings. This array is called "$options" and contains the names of our menu options. When an option is selected, that value is passed to the variable "$opt".

```
options=("Update ETPRO Rules." "Update Anomali Rules." "Update Both Rule Sets." "Remove Unnecessary
Rules." "Editing Tool." "Quit.")
select opt in "${options[@]}"
do

done
```

The prompt for the select command can be set using the built-in variable "PS3". This is done customize a descriptive prompt for the user at menus.

```
PS3='Please select an option above: '
```

The "case" command is used for pattern matching. It is used to compare a variable against different patterns and execute different commands based on the match.

```
case variable in
        pattern 1)
                # commands
                ;;
        *)
                # commands
                ;;
esac
```

This Script uses this to match the output of the "select" command to our desired code blocks to be executed. "case" is supplied the variable "$opt" and matches its value against our listed patterns. These patterns match the values in the "options" array ('*' is used as the default case, if no other match is found.).

```
case $opt in
        "Update ETPRO Rules.")
                update_etpro
                so_update
                sleep 2
                break
                ;;
        "etc...")
                # commands
                ;;
        * )
                echo "Please select a valid option."
                ;;
esac
```

Once the "case" command matches the pattern string that section of code is then run. Here we make several function calls based on the specific tasks to be accomplished by each menu option. (The main functions can be found in the following sections. More can be found in Supporting Functions).

Below is the menu code snippet for review:

```
menu () {
    # Main Menu
    while true ;
    do
        PS3='Please select an option above: '
        options=("Update ETPRO Rules." "Update Anomali Rules." "Update Both Rule Sets." "Remove
Unnecessary Rules." "Editing Tool." "Quit.")
        select opt in "${options[@]}"
        do
            case $opt in
                "Update ETPRO Rules.")
                    update_etpro
                    so_update
                    sleep 2
                    break
                    ;;
                "Update Anomali Rules.")
                    update_anomali
                    so_update
                    sleep 2
                    break
                    ;;
                "Update Both Rule Sets.")
                    update_etpro
                    update_anomali
                    so_update
                    sleep 2
                    break
```

```
                    ;;
                "Remove Unnecessary Rules.")
                    remove_rule
                    break
                    ;;
                "Editing Tool.")
                    edit_tool
                    break
                    ;;
                "Quit.")
                    echo "Exiting..."
                    sleep 1
                    break 2
                    ;;
                * )
                    echo "Please select a valid option."
                    ;;
            esac
        done
    done
}
```

Option 1: Update ETPRO Rules

The update_etpro function is used to update ETPRO Rules. This section is used to find and process files for ETPRO rule updates.

The main body of this function consists of a single "if" statement. This is used to verify that the "etpro.rules.tar.gz" file used to update the rules is present in the /tmp/ directory, and if so run the commands necessary to update. Otherwise the script will pass an error message and return to the main menu.

```
filepath="/tmp/etpro.rules.tar.gz"
if [ -f "$filepath" ]; then
        # run update
else
        # error
fi
```

The variable "$filepath" is created here and assigned the value of "/tmp/etpro.rules.tar.gz". This variable is used by the if statement to evaluate if the file exists.

The "so-update" function is then called to continue the rules update process. Explained in detail Here.

Below is the full code snippet for review:

```
update_etpro () {
    # Update ETPRO Rules.
    echo "Updating ETPRO Rules."
    filepath="/tmp/etpro.rules.tar.gz"
    if [ -f "$filepath" ]; then
        echo "File found. Starting updates."
        tar -xf /tmp/etpro.rules.tar.gz -C /tmp
        cat /tmp/rules/*.rules > /nsm/repo/rules/emerging-all.rules
    else
        echo "File does not exist. (ETPRO Rules). Exiting to menu."
        sleep 2
        break 2
    fi
}
```

Option 2: Update Anomali Rules

The update_anomali function is used to update Anomali Rules. This section is used to find and process files for Anomali rule updates.

> ⓘ **Note:**
>
> The script uses the raw file downloaded from DoD SAFE to execute Anomali Rule updates.

The main body of this script sits between two nested "if" statements. The first "if" statement is used to verify if the required file exists or not. The second "if" statement is used to verify the file was processed correctly before beginning updates.

```
filepath=$(find /tmp -name "*DoD SAFE*.zip" -type f -print -quit 2>/dev/null)
    if [ ! -f $filepath ]; then
        # error 1
    else
        # process file
        filepath2="/tmp/AllRules.txt"
        if [ -f "$filepath2" ]; then
            # update
        else
            # error 2
        fi
    fi
```

Here the "$filepath" variable is assigned the value of a "find" command. This "find" command is used instead of a direct path because of the whitespace character in the file name, which causes problems when passed to a command.

> 🐞 **Bug:**
>
> Occasionally an error is passed for "binary operator expected". This is due to the whitespace in the name and the way Linux does filenames. This error may be ignored.

The first "if" statement varies by checking if the variable "$filepath" does not exist. If it does not, an error is passed and we are returned to the menu. Otherwise the zip file is processed into a useable file for updates. The "unzip" and "find" commands are used to unpack the file and search it for text files to combine.

```
unzip "$filepath" -d /tmp/temp
find /tmp/temp -name "*.txt" -exec cat {} + > /tmp/AllRules.txt
```

"unzip" unpacks the file and creates the /tmp/temp directory to store it. "find" is then used to recursively search for text files within the directory and combine them into /tmp/AllRules.txt. This is the file that will be used for Anomali rule updates. The second "if" statement then checks to verify this file was created, or an error is passed and we are returned to the main menu.

AllRules.txt is then appended to the local.rules file, and the variable "$filepath" is removed as it is no longer needed.

```
cat /tmp/AllRules.txt >> /opt/so/rules/nids/local.rules
rm -rf "$filepath"
```

The "so-update" function is then called to continue the rules update process. Explained in detail Here.

> ⁝☰ **Below is the full code snippet for review:**

```
update_anomali () {
    # Update Anomali Rules.
    echo "Updating Anomali Rules"
```

```
    filepath=$(find /tmp -name "*DoD SAFE*.zip" -type f -print -quit 2>/dev/null)
    if [ ! -f $filepath ]; then
        echo "File not found (DoD SAFE zip). Exiting to menu."
        sleep 2
    else
        echo "File found (DoD SAFE). Starting updates."
        echo "Unpacking DoD SAFE zip file."
        unzip "$filepath" -d /tmp/temp
        echo "creating AllRules.txt file."
        find /tmp/temp -name "*.txt" -exec cat {} + > /tmp/AllRules.txt
        filepath2="/tmp/AllRules.txt"
        if [ -f "$filepath2" ]; then
            echo "Complete. Preparing for Rule updates."
            cat /tmp/AllRules.txt >> /opt/so/rules/nids/local.rules
            rm -rf "$filepath"
        else
            echo "File does not exist. (AllRules.txt). Exiting to menu."
            sleep 2
            break 2
        fi
    fi
```

Option 3: Update Both Rule Sets

This option calls the "update_etpro" and "update_anomali" functions together to update both rule sets simultaneously.

The "so-update" function is then called to continue the rules update process. Explained in detail Here.

Below is the full code snippet for review:

```
# pulled from 'menu' funtion
"Update Both Rule Sets.")
        update_etpro
        update_anomali
        so_update
        sleep 2
        break
        ;;
```

Option 4: Remove Unnecessary Rules

The remove_rule function is used to remove unnecessary rules. (A walkthrough for this option can be found here)

This function is constructed similarly to the "menu" function. They both use the structure of the while loop, "select" and "case" commands to generate a menu. This is done to add space for more functionality to be added later. The main section of the function follows the "Remove By SID" pattern in the case command.

The "read" command is used to take the users inputted SID/s and assign them to the variable "$sid". A new variable is then created called "$regex" and is assigned a regex pattern. This pattern is used to verify the user input is a valid SID, not just a random string that could cause script errors.

```
regex="(^[0-9]{6}[0-9]+)(,[0-9]{6}[0-9]+){0,10}"
```

Regex Breakdown:

(^[0-9]{6}[0-9]+)(,[0-9]{6}[0-9]+){0,10}
This checks for a 7 or more digit number, followed by a comma and another number, between 0 and 10 times.

An "if" statement is used to compare the user input to the regex string. If a match is not found, an error message is displayed and we are returned to the menu.

```
read -p "Please type the rule SID/s you would like to remove. (For multiple seperate by ','): " sid
regex="(^[0-9]{6}[0-9]+)(,[0-9]{6}[0-9]+){0,10}"
if [[ "$sid" =~ $regex ]]; then
        # remove rule
else
        # error
fi
```

Inside the "if" statement we create a new variable called "$format". This variable is used to format the inputted SIDs into an AWK command, which is then used to remove the desired rule. This is done in conjunction with an "echo" command to modify the syntax to what is required for the command. The AWK command is then saved to a text file "/tmp/sid.txt".

```
format="awk '!/sid:($sid)/' /opt/so/rules/nids/all.rules > tempfile.txt && mv tempfile.txt
/opt/so/rules/nids/all.rules"
echo ${format//,/|} > /tmp/sid.txt
```

Note:

Commas "," are used in the input to delimit rule SIDs, however for command syntax a pipe "|" is needed to create an "or" statement.

Once the text file is created, the "bash" command is used to run the AWK command. The bash command is used to send commands from a file to the command line. Once the command has been run, the temporary script file is cleared and the "chmod" command is used to reset file permissions on the all.rules file. This ensures the so-rule-update command runs properly.

```
bash /tmp/sid.txt
rm -rf /tmp/sid.txt
chmod 777 /opt/so/rules/nids/all.rules
```

AWK Script Breakdown

```
awk '!/sid:($sid)/' /opt/so/rules/nids/all.rules > tempfile.txt && mv tempfile.txt
/opt/so/rules/nids/all.rules
```

This one line command is used to remove rules from the all.rules file. AWK is called to find lines in /opt/so/rules/nids/all.rules that do not contain the pattern "sid:($sid)". The variable "$sid" is the user input from the beginning of the function.

```
awk '!/sid:($sid)/' /opt/so/rules/nids/all.rules
```

Every line that does not contain the desired SID is then sent to a temporary text file "tempfile.txt" AND the "mv" command is called to move the contents of "tempfile.txt" back to "/opt/so/rules/nids/all.rules" overwriting its previous content, removing the rule.

```
> tempfile.txt && mv tempfile.txt /opt/so/rules/nids/all.rules
```

Below is the full code snippet for review:

```
remove_rule () {
    # Remove Unnecessary rules.
    echo "Removing Unnecessary Rules."
    backup
```

```
    while true ;
    do
        options=("Remove by SID." "Quit.")
        select opt in "${options[@]}"
        do
            case $opt in
                "Remove by SID.")
                    read -p "Please type the rule SID/s you would like to remove. (For multiple
seperate by ','): " sid
                    regex="(^[0-9]{6}[0-9]+)(,[0-9]{6}[0-9]+){0,10}"
                    if [[ "$sid" =~ $regex ]]; then
                        echo "Formatting Awk script."
                        format="awk '!/sid:($sid)/' /opt/so/rules/nids/all.rules > tempfile.txt && mv
tempfile.txt /opt/so/rules/nids/all.rules"
                        echo ${format//,/|} > /tmp/sid.txt
                        echo "Running Awk Script. Clearing selected rules."
                        bash /tmp/sid.txt
                        echo "Awk Script complete. Clearing files from /tmp/ directory."
                        rm -rf /tmp/sid.txt
                        echo "Cleared temporary files. Resetting file permissions."
                        chmod 777 /opt/so/rules/nids/all.rules
                        echo "Complete. Type '3' to exit to main menu."
                    else
                        echo "Invalid entry. Input must be a rule SID."
                        continue
                    fi
                    ;;
                "Quit.")
                    echo "Exiting to menu."
                    sleep 2
                    break 2
                    ;;
                * )
                    echo "Please enter a valid response."
                    ;;
            esac
        done
    done
}
```

Option 5: Editing Tool

The edit_tool function is used to perform many functions for editing and thresholding rules.

The function is built using the same method as the "menu" function. The main difference being that the "backup" function is called before the while loop is opened. This menu is used to select options for manual or automatic rule edits.

The first option calls the "save_rule" function to save a rule to a text file (/tmp/saverule.txt) for manual edits.

```
"Save rule to file.")
        save_rule
        ;;
```

The second option is used to add the "saverule.txt" back into the all.rules file after edits have been made. This is the only option that does not use function calls.

```
"Implement saved rule file.")
        cat /tmp/saverule.txt >> /opt/so/rules/nids/all.rules
        ;;
```

The third option is used to generate a custom threshold and save it to a text file so that it may easily be copied and pasted to a rule of your choice. A set of instructions is also included to help. This is done by calling the "threshold"

and "instructions" functions.

```
"Threshold Generator.")
        threshold
        instructions
        ;;
```

The fourth option is used to automatically add a threshold to a rule or rules of your choice. This is done by calling the "save_rule", "threshold", and "threshold_awk" functions.

```
"Auto Threshold.")
        save_rule
        threshold
        threshold_awk
        ;;
```

This combines the functionality of the previous 3 options into one streamlined process. For breakdowns of functions included in this section see Supporting Functions.

Below is the full code snippet for review:

```
edit_tool () {
    # editing tool
    echo "Editing Tool."
    backup
    while true ;
    do
        options=("Save rule to file." "Implement saved rule file." "Threshold Generator." "Auto
Threshold." "Quit.")
        select opt in "${options[@]}"
        do
            case $opt in
                "Save rule to file.")
                    save_rule
                    echo "Complete. Type '5' to exit to main menu."
                    ;;
                "Implement saved rule file.")
                    echo "Re-Implementing /tmp/saverule.txt"
                    cat /tmp/saverule.txt >> /opt/so/rules/nids/all.rules
                    echo "Complete. Type '5' to exit to main menu."
                    ;;
                "Threshold Generator.")
                    threshold
                    instructions
                    echo "Complete. Type '5' to exit to main menu."
                    ;;
                "Auto Threshold.")
                    save_rule
                    threshold
                    threshold_awk
                    echo "Complete. Type '5' to exit to main menu."
                    ;;
                "Quit.")
                    echo "Exiting to menu."
                    sleep 2
                    break 2
                    ;;
                * )
                    echo "Please enter a valid response."
                    ;;
            esac
```

```
        done
    done
```

Supporting Functions

threshold

The "threshold" function is used by the threshold generator to create a menu for the user to pick the threshold type they would like.

This function is also used to call other functions required to create the threshold including: "track", "count_sec", and "backoff".

Below is the full code snippet for review:

```
threshold () {
    # generate menu to select type of threshold (threshold,limit,both,backoff)
    PS3="Please select the type of threshold you would like: "
    options=("Threshold." "Limit." "Both." "Backoff.")
    select opt in "${options[@]}"
    do
        case $opt in
            "Threshold.")
                type="threshold"
                track
                count_sec
                return
                ;;
            "Limit.")
                type="limit"
                track
                count_sec
                return
                ;;
            "Both.")
                type="both"
                track
                count_sec
                return
                ;;
            "Backoff.")
                type="backoff"
                track
                backoff
                return
                ;;
            * )
                echo "Please select a valid option."
                return
                ;;
        esac
    done
}
```

track

The "track" function is used by the threshold generator to create a menu for the user to pick the track option they would like. You are then returned to the "threshold" function.

Below is the full code snippet for review:

```
track () {
    # menu for track by (by_src,by_dst,by_rule,by_both,by_flow)
    PS3="Please select the tracking you would like to use: "
    options=("By Source." "By Destination." "By Rule." "By Both." "By Flow." )
    select opt in "${options[@]}"
        do
            case $opt in
                "By Source.")
                    track="by_src"
                    return
                    ;;
                "By Destination.")
                    track="by_dst"
                    return
                    ;;
                "By Rule.")
                    track="by_rule"
                    return
                    ;;
                "By Both.")
                    track="by_both"
                    return
                    ;;
                "By Flow.")
                    track="by_flow"
                    return
                    ;;
                * )
                    echo "Please select a valid option."
                    return
                    ;;
            esac
        done
}
```

count_sec

The "count_sec" function is used by the threshold generator to take user input for the count and seconds options required for threshold, limit, and both type thresholds. The global variable "$num" is called in the "if" statements to verify the input for the "$count" and "$seconds" variables are numbers.

The variables "$type", "$track", "$count", and "$seconds" are then formatted into the proper sequence for a threshold and echoed into the text file "/tmp/threshold.txt".

Below is the full code snippet for review:

```
count_sec () {

    # input for seconds and count (threshold,limit,both)
    read -p "Please enter a value for count: " count
    if [[ "$count" =~ $num ]]; then
        read -p "Please enter a value for seconds: " seconds
        if [[ "$seconds" =~ $num ]]; then
            echo -e "\nthreshold: type $type, track $track, count $count, seconds $seconds;" >
/tmp/threshold.txt
            return
        else
            echo "Input must be a numerical value. Please use valid input."
            sleep 2
            break 2
        fi
    else
        echo "Input must be a numerical value. Please use valid input."
```

```
        sleep 2
        break 2
    fi
}
```

backoff

The "backoff" function is used by the threshold generator to take user input for the count and multiplier options required for a backoff type threshold. We are then returned to the "threshold" function. The global variable "$num" is called in the "if" statements to verify the input for the "$count" and "$seconds" variables are numbers.

The variables "$type", "$track", "$count", and "$multiplier" are then formatted into the proper sequence for a threshold and echoed into the text file "/tmp/threshold.txt".

Below is the full code snippet for review:

```
backoff () {
    # input for count and multiplier (backoff)
    read -p "Please enter a value for count: " count
    if [[ "$count" =~ $num ]]; then
        read -p "Please enter a value for multiplier: " multiplier
        if [[ "$multiplier" =~ $num ]]; then
            echo -e "\nthreshold: type $type, track $track, count $count, multiplier $multiplier;" >
/tmp/threshold.txt
            return
        else
            echo "Input must be a numerical value. Please use valid input."
            sleep 2
            break 2
        fi
    else
        echo "Input must be a numerical value. Please use valid input."
        sleep 2
        break 2
    fi
}
```

Instructions

The "instructions" function is used by the Editing Tool to append instructions on how to apply a threshold to a saved rule to the bottom of the text file created by the threshold generator.

Below is the full code snippet for review:

```
instructions () {
    echo -e "\n\n\nInstructions:\n-------------\nTo add the threshold to the desired rule, copy from
this file and paste after the last "content" field in the rule.\nYou may use the Editing Tool to save
your desired rule by SID to a text file for editing.\n" >> /tmp/threshold.txt
    echo -e "To copy and paste the string from the command line use the following commands:\ncat
/tmp/threshold.txt - highlight and ctrl+c to copy\nvim /tmp/saverule.txt - type 'i' to enter insert
mode, move cursor to the desired location, right click to paste" >> /tmp/threshold.txt
    echo "The threshold string has been saved to /tmp/threshold.txt"
    echo "Instructions have been included to help you properly install your threshold."
    sleep 2
    break
}
```

save_rule

The "save_rule" function is used by the Editing Tool to copy a rule from all.rules and save it to a text file. The script then removes the rule from "all.rules".

This function is a derivative of the code used in the "remove_rule" function. Instead of utilizing a single AWK command to remove the desired rule, it uses two. The first is a modified version that copies the desired rule to a text file "/tmp/saverule.txt". The second is the same AWK script used to remove the rule from the "all.rules" file.

Below is the full code snippet for review:

```
save_rule () {
    # save rule form all.rules to /tmp/saverule.txt file
    read -p "Please type the rule SID/s you would like to remove. (For multiple seperate by ','): "
sid
    regex="(^[0-9]{6}[0-9]+)(,[0-9]{6}[0-9]+){0,10}"
    if [[ "$sid" =~ $regex ]]; then
        echo "Formatting Awk script."
        format="awk '/sid:($sid)/' /opt/so/rules/nids/all.rules > /tmp/saverule.txt"
        echo ${format//,/|} > /tmp/awk.txt
        echo "Running Awk Script. Saving selected rules."
        bash /tmp/awk.txt
        script="awk '!/sid:($sid)/' /opt/so/rules/nids/all.rules > tempfile.txt && mv tempfile.txt
/opt/so/rules/nids/all.rules"
        echo ${script//,/|} > /tmp/sid.txt
        bash /tmp/sid.txt
        chmod 777 /opt/so/rules/nids/all.rules
        echo "Awk Script complete. Clearing files from /tmp/ directory."
        rm -rf /tmp/awk.txt
        rm -rf /tmp/sid.txt
        echo "Cleared temporary files."
        echo "The requested rule/s has been saved to /tmp/saverule.txt"
    else
        echo "Invalid entry. Input must be a rule SID."
        continue
    fi
}
```

auto_check

The "auto_check" function is used to check if a rule already has a prior implemented threshold before running the "threshold_awk" command. If so execution is stopped.

Below is the full code snippet for review:

```
auto_check () {
    # check for previously implemented threshold
    if grep -q "threshold:" "/tmp/saverule.txt"; then
        echo "Previously implemented threshold found. Please use manual edit."
        echo "Rule saved to /tmp/saverule.txt"
        echo "Exiting..."
        break
    fi
}
```

threshold_awk

The "threshold_awk" function is used by the Editing Tool to automatically run a short AWK script to add a threshold to a chosen rule.

The function starts by creating variable "$threshold_string" and assigning it the value "< /tmp/threshold.txt". This is used to pass the content of the file to the variable.

```
threshold_string=$(< /tmp/threshold.txt)
```

A short "if" statement is then used to check if "/tmp/ThresholdAwk.awk" exists (this is the script used to append thresholds to rules). If not, the script is created using "touch" to create the empty file and "echo" to append the script to the file.

```
    if [ -f "/tmp/ThresholdAwk.awk" ]; then
        # run script
    else
        # create script
        chmod +x /tmp/ThresholdAwk.awk
    fi
```

Once the script is found/created, a one line AWK command is called to execute the script.

```
awk -v threshold="$threshold_string" -f ThresholdAwk.awk /tmp/saverule.txt > /tmp/output.txt
```

The "-v" is used to import a variable, in this case "$threshold_string" which will become AWK variable "threshold". The -f is used to pull from a source file (our script), "ThresholdAwk.awk". The script is then run on the file "/tmp/saverule.txt" outputting to "/tmp/output.txt" our saved rule now with our threshold.

Below is the content of the AWK script:

```
#!/usr/bin/awk -v threshold="$threshold_string" -f
{
  if ($0 ~ /^alert/) {
          sub(/\)$/, " " threshold ")")
  }' >> /tmp/ThresholdAwk.awk
  print
}
```

Put simply, the script looks for a Suricata rule by the starting keyword "alert", and then appends the threshold string before the closing parentheses.

Below is the full code snippet for review:

```
threshold_awk () {
    # call awk script to automatically insert threshold string
    threshold_string=$(< /tmp/threshold.txt)
    echo "Adding Auto-Threshold."
    if [ -f "/tmp/ThresholdAwk.awk" ]; then
        echo "Calling ThresholdAwk.awk script."
    else
        echo "Creating ThresholdAwk.awk script."
        touch /tmp/ThresholdAwk.awk
        echo '#!/usr/bin/awk -v threshold="$threshold_string" -f' > /tmp/ThresholdAwk.awk
        echo '{' >> /tmp/ThresholdAwk.awk
        echo '  if ($0 ~ /^alert/) {' >> /tmp/ThresholdAwk.awk
        echo '      sub(/\)$/, " " threshold ")")' >> /tmp/ThresholdAwk.awk
        echo '  }' >> /tmp/ThresholdAwk.awk
        echo '  print' >> /tmp/ThresholdAwk.awk
        echo '}' >> /tmp/ThresholdAwk.awk
        echo "Complete. Updating file permissions."
        chmod +x /tmp/ThresholdAwk.awk
        echo "Complete. Calling ThresholdAwk.awk script."
        sleep 2
    fi
    awk -v threshold="$threshold_string" -f ThresholdAwk.awk /tmp/saverule.txt > /tmp/output.txt
    echo "Complete. Re-implementing rule."
    auto_add
}
```

auto_add

The "auto_add" function is called by the "threshold_awk" function to add modified rules from "/tmp/output.txt" back to all.rules.

Below is the full code snippet for review:

```
auto_add () {
    # reimplement rule from txt file to all.rules
    cat /tmp/output.txt >> /opt/so/rules/nids/all.rules
    echo "Complete. exiting to menu."
    rm -rf /tmp/output.txt
    sleep 2
    break
}
```

so_update

The "so_update" function is used to run the so-rule-update command and call the "save_files" and "clear_files" functions. This is done separately to reduce redundant code features.

Below is the full code snippet for review:

```
so_update () {
    # Run so-rule-update
    echo "Running Rule Update..."
    so-rule-update
    echo "Update Complete. Saving rule file backup"
    save_files
    clear_files
}
```

backup

The "backup" function is used to generate a timestamped backup of the all.rules file before each option or function that would make changes.

Note:

The backups can be found at "/tmp/backup"
The backup is dated using the format: "all.rules_YYYY_MM_DD_HH:mm"

The function uses a variable "$i" as a counter. When the script is started "$i" is initialized to 0. Every time "backup" is run, "$i" is incremented by 1. If "$i" is greater than or equal to '1', the user is asked if they would like to make another backup.

Below is the full code snippet for review:

```
backup () {
    # create file backup of /opt/so/rules/nids/all.rules
    if [ $i -ge 1 ]; then
        echo "A backup has already been made during this session."
        read -p "Would you like to make another? Y/N : " answer
        case "$answer" in
            [Yy] )
                # create backup
                ;;
            [Nn] )
                echo "Skipping backup."
                return
                ;;
```

```
            * )
                echo "Please enter a valid response."
                return
                ;;
        esac
    fi
    if [ ! -d "/tmp/backup" ]; then
        mkdir /tmp/backup
        echo "Created /tmp/backup/ directory. Creating file backup."
    else
        echo "Creating file backup."
    fi
    cat /opt/so/rules/nids/all.rules > /tmp/backup/all.rules
    rules="/tmp/backup/all.rules"
    newfilename="${rules}_${date}"
    mv $rules $newfilename
    echo "A copy of the current /opt/so/rules/nids/all.rules file has been save to /tmp/backup/"
    ((i++))
}
```

clear_files

The "clear_files" function is used to clear unnecessary or temporary files generated in the process of running rule updates.

Below is the full code snippet for review:

```
clear_files () {
    # clear temporary files from /tmp/
    echo "Complete. Clearing temporary files from /tmp/ directory."
    files=("/tmp/rules" "/tmp/temp")
    for file in "${files[@]}";
    do
        if [ -e $file ]; then
            rm -rf "$file"
            echo "Cleared $file"
        fi
    done
    echo "Complete. Returning to main menu."
}
```

save_files

The "save_files" function is used to save timestamped copies of the raw alerts files for record keeping purposes.

Below is the full code snippet for review:

```
save_files () {
    # save raw alert files to /tmp/updates/ for records
    if [ ! -d "/tmp/updates" ]; then
        mkdir /tmp/updates
        echo "Created /tmp/updates/ directory. Creating file backup."
    else
        echo "Creating file backup."
    fi
    files=("/tmp/etpro.rules.tar.gz" "/tmp/AllRules.txt" "$filepath")
    for file in "${files[@]}";
    do
        if [ -e $file ]; then
            newfilename="${file}_${date}"
            mv $file $newfilename
            mv "$newfilename" "/tmp/updates/"
```

```
            echo "Saved $file"
        fi
    done
    echo "Used rules files can be found in the /tmp/updates directory for reference."
}
```

splash_screen

The "splash_screen" function is used to display randomly selected ascii art when the script is first run. It serves no functional purpose.

Below is the full code snippet for review:

```
splash_screen () {
    # display randomly selected ascii art when the script is run
    clear
    ascii_art=(
    "
        # ascii art goes here
    "
    "
        # ascii art goes here
    "
    )
    num_art=${#ascii_art[@]}
    random_index=$(shuf -i 0-$(($num_art - 1)) -n 1)
    echo "${ascii_art[$random_index]}"
}
```

Sudo Check

The following code block is used at the beginning of the script to verify it is run with root privileges, otherwise it will exit and pass an error message to the user.

```
# sudo check
if [ "$EUID" -ne 0 ]; then
    echo "This script must be run with root privileges."
    exit
else
    echo "It worked."
fi
```