

By utilizing the Python scripting language and its huge variety of libraries, I was able to write an advanced script that implements CLI commands, in order to simplify and increase the speed of specific actions. The reason why this script was created was because of the inability of Snort's IPS mode to function properly in SDN networks (This will be seen in more detail on chapter 8. Let's analyze how it works, what libraries it utilizes and how one can tweak the code in order to adjust it to his/her needs.

The basic idea behind this script was to utilize Python libraries to implement CLI commands. To achieve this, I used Pexpect, a python module that spawns other applications as childs and allows the control of them through it. It is especially useful if you want to automate CLI procedures, like SSH, FTP etc. and setup scripts in different hosts, in order to execute specific actions. In my case, I combined it with Snort, and used it to read its alerts and insert firewall rules based on the incoming alerts and the administrator's choices.

To be able to use this module on any python script, you have to install it using pip:
`pip install pexpect`

After installing it, its usage is very simple. You only have to import it in your script, as you will see in a while.

To create a child, which will refer to the application you want to start and control, you need to use the `pexpect.spawn()` command and assign it to a variable. After you spawn your child-application, the pexpect has two main procedures you will use the most to control it: **`expect()`** and **`send()/sendline()`**.

The first procedure expects the application to return a specified string, while the second one sends a specified string back to the child-application. After using the `expect()` method, you can use the "before" and "after" properties to either get all the text up to the specified string or only get the text that matched the specified string, respectively. In figure 35, you can see an example where the user partially automates the usage of an FTP server, and then gains control over the application.

When you try to match a specified string, Pexpect will always return a minimal match. If you want to return just one character, you can use the `child.expect('.')` and if you want to match as little as possible, you should use the `child.expect('.*')`.

```
import pexpect
child = pexpect.spawn('ftp ftp.openbsd.org')
child.expect('Name .*: ')
child.sendline('anonymous')
child.expect('Password:')
child.sendline('noah@example.com')
child.expect('ftp> ')
child.sendline('ls /pub/OpenBSD/')
child.expect('ftp> ')
print child.before # Print the result of the ls command.
child.interact()   # Give control of the child to the user.
```

Figure 1 FTP server control through Pexpect [51]

A very important property you should keep in mind is the TIMEOUT property, which allows you to set the time before a timeout exception is generated on a session, according to your needs. For example. Let's say you try to create a child-application and it is not responding or it is taking longer than 30 seconds to start. Then, by default, Pexpect will throw a timeout exception after 30 seconds. In order to change the timeout time in any Pexpect procedure, you can use this attribute like this:

```
child.expect('password:', timeout=120)
```

Finally, there are the EOF exceptions and the EOL identification. For the later, since Pexpect reads one character at a time, when it comes to child output, in order to identify the end of the line of this output we use the DOS-style "\n" or "\r\n". This way, Pexpect either "sees" the end of the line or "sees" the EOL and looks for a new one, for example `child.expect('\r\n')`.

When a child-application has stopped working or it has stopped giving an output, the `expect()` procedure will throw an EOF exception. In order to avoid this, we can use the `expect(pexpect.EOF)` method. We can also use this to catch such exceptions, in a Try...catch scenario as well, like this `except pexpect.EOF:`, in order to take specific actions in case the child-application stops sending output.

Now, let's break down the basic structure of the script, which consists of 2 main parts: The IPS part, that detects Snort alerts and prompts the administrator to take preset actions, and the Firewall control part, which allows the user to insert firewall rules based on specific attributes. These are used based on user choice, through a selection input. If the administrator wishes, he can add similar sections in each part, to cover his/her needs. The first part can also be divided into parts, depending on what kind of rules the administrator wants to implement. In my experiments, I focused on ICMP and TCP attacks for demonstration, therefore I separated my code into 2 parts, dedicated into each one of those. Those "parts" are a couple of "if..." statements, inside 2 while loops and one for... loop. The "while" statements are responsible for running the program and choice-making, while the if... statements, for each part, are responsible for matching with alert output from Pexpect or alerting the user, if specific flags reach a specific number. They can either look for individual alerts, or use a list, which is compared with the alert's SID to look for bulk rules, for example TCP attacks.

```

while True:
    try:
        child.expect('\r\n', timeout=120000)
        print(child.before)

    #Checks output for specific alert code and acts
    for line in str(child.before).split("/r"):
        #Ping attack check
        for word in line.split(" "):
            if word == "[1:10000002:1]":
                pingFlag = pingFlag + 1
                print("PING TEST")
                break
            if pingFlag == 3:
                print("Suspicious ICMP packets - select action [0-nothing, 1-block traffic] ")
                action = input("")
                if action == 0:
                    pingFlag = 0
                    break
                else:
                    reject AllTraffic = "sudo iptables -A INPUT -i '{0}' -j DROP".format(iface)
                    child2 = pexpect.spawn(reject AllTraffic)
                    print("IPS: All Traffic Towards Server has been cut")
                    pingFlag = 0
                    break
            if pingFlag == 0:
                break;

```

Figure 2 A section in the script, discovering 1 specific rule.

For bulk rule SIDs, for which we want to have the same action, we can use the structure seen in Figure 37, which utilizes a list of Snort rule SIDs.

```

#Nmap_detection
for word in line.split(" "):
    for lex in nmapRules:
        if word == lex:
            nmapFlag = nmapFlag + 1
            print("NMAP TEST")

            continue

    if nmapFlag > 3:
        print("Nmap XMAS Scan - select action [0-nothing, 1-block traffic] ")
        action = input("")
        if action == 0:
            nmapFlag = 0
            continue
        else:
            reject AllTraffic = "iptables -A INPUT -i '{0}' -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP".format(iface)
            child2 = pexpect.spawn(reject AllTraffic)
            print("IPS: All Traffic Towards Server has been cut")
            nmapFlag = 0
            continue

```

Figure 3 Bulk rule trigger structure

In general, the way the script triggers any IPS action is by using specific counter flags, which increase by 1 every time an alert with a set ID is recognized. These flags are usually named after the attack they are focused on, like “pingFlag” and they are what I will call a “threshold”, as seen in Figure 38.

```

pingFlag = 0
tcpFlag = 0
nmapFlag = 0
tcpRules = ["[1:10000021:0]", "[1:10000022:0]", "[1:10000023:0]"]
nmapRules = ["[1:10001136:1]", "[1:10001137:1]", "[1:10001138:1]"]

```

Figure 4 Script threshold flags and rule SID lists

In Figure 36 and 37, you see a section focused to look and take specific actions for a specific alert. After splitting the alert, using the “child.before” procedure, it searches for a specific alert ID, set in a variable or list. If this alert ID shows up, the threshold count raises. When it reaches the set threshold, the administrator is alerted that something is wrong and is prompted to take actions. In my example, the action is to cut all traffic towards the selected interface. By inserting a different CLI command, or by adding another option, the administrator can tweak the pre-selected actions.

Otherwise, after he gets alerted, he can insert his/her own in the next main part of the script, the firewall control. That part, which is the second one, consists of 4 options, based on how complicated the firewall rule the admin wants to insert is.

Bulk rule comparison works in a similar fashion, but instead of comparing incoming rule SIDs with a variable, the SIDs get compared with a list of rule SIDs, which will trigger actions after a specific alert threshold.

```
elif inpt=="1":
    while True:
        try:
            iface = input("Insert interface you would like to target:\n")
            selection = input("1-Accept all traffic on {0} 2-Reject all traffic on {0}\n".format(iface))

            if selection=="1":
                comm = "sudo iptables -A INPUT -i '{0}' -j ACCEPT && sudo iptables -D INPUT -i '{0}' -j DROP ".format(iface)
                child2 = pexpect.spawn(comm)
                print(child2.before)
                time.sleep(1)

            else:
                comm = "sudo iptables -A INPUT -i '{0}' -j DROP".format(iface)
                child2 = pexpect.spawn(comm)
                print(child2.before)
                time.sleep(1)

            choice = input("Do you want to permanently save those rules?(y/n)\n")
            if choice == 'y' or choice == 'Y':
                child3 = pexpect.spawn("sudo invoke-rc.d iptables-persistent save")
                print(child3.before)
                break

        except (KeyboardInterrupt, SystemExit):
            break

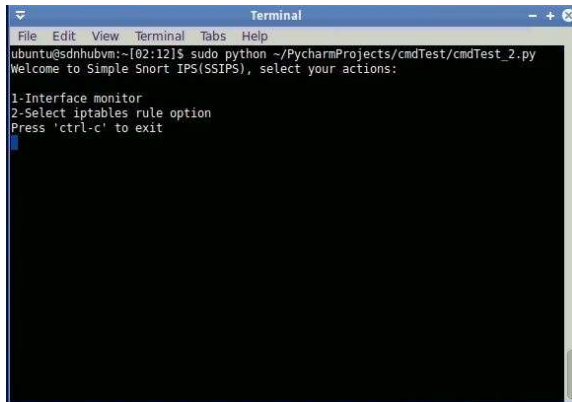
    except pexpect.EOF:
        print("Pexpect error, quitting...\n")
        break

    except SyntaxError:
        print("Error in command... \n")
        break
```

Figure 5 Choice structure of firewall control.

In Figure 39 you can see the 1st option, which allows the administrator to choose a specific interface to Accept or Block traffic. Each option, in the second part, has the same structure: it takes one or more user inputs, asks if the user wants to Accept or Reject traffic with the specified attributes and executes the necessary commands.

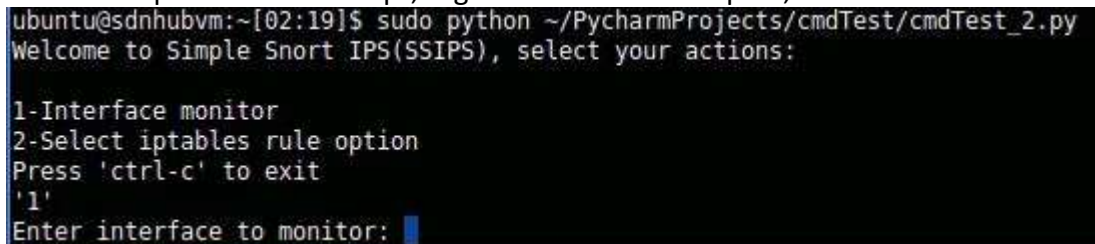
In practice, all these are executed in CLI by typing “sudo python [script path]” and have the following option, seen in Figure 40.



```
Terminal
File Edit View Terminal Tabs Help
ubuntu@sdnhubvm:~[02:12]$ sudo python ~/PycharmProjects/cmdTest/cmdTest_2.py
Welcome to Simple Snort IPS(SSIPS), select your actions:
1-Interface monitor
2-Select iptables rule option
Press 'ctrl-c' to exit
```

Figure 6 SnortIPS starting screen

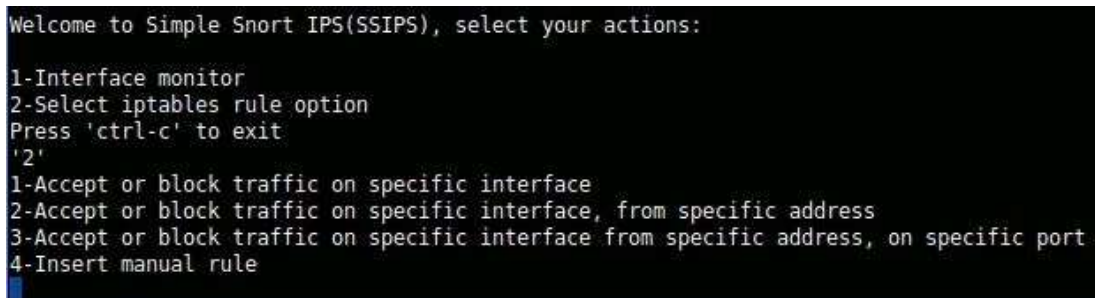
By typing “1” or “2”, the user can either monitor or use iptables rules. If he/she chooses the “1”, he will be prompted to enter the interface he/she wants to monitor, and then Snort will be launched. Together with the Snort alerts, the administrator will receive custom outputs from the script, together with choice inputs, for IPS functions.



```
ubuntu@sdnhubvm:~[02:19]$ sudo python ~/PycharmProjects/cmdTest/cmdTest_2.py
Welcome to Simple Snort IPS(SSIPS), select your actions:
1-Interface monitor
2-Select iptables rule option
Press 'ctrl-c' to exit
'1'
Enter interface to monitor: 
```

Figure 7SnortIPS interface selection

In the “2” case, the administrator will be prompted to select between 4 options, as seen in Figure 41. Each choice will include 1-3 inputs, depending on how many attributes are required.



```
Welcome to Simple Snort IPS(SSIPS), select your actions:
1-Interface monitor
2-Select iptables rule option
Press 'ctrl-c' to exit
'2'
1-Accept or block traffic on specific interface
2-Accept or block traffic on specific interface, from specific address
3-Accept or block traffic on specific interface from specific address, on specific port
4-Insert manual rule
```

Figure 8 SnortIPS firewall options

Any input that does not comply with the available options will usually redirect the user to select an option again.