An Automated Checking and Feedback System for a Console Web Interface

Team 20

[redacted]

University of Adelaide, South Australia

Abstract

This project aims to build an automated checking and feedback system by adding features to an existed web-based virtual console. The final deliverable of this project can sanitize and block malicious input, execute automated scripts that take action based on input/output, and provide feedback to users. With the aid of this project, users can enjoy greater safety and efficiency while operating the virtual console.

1. Introduction

A virtual console (or terminal) is the conceptual combination of keyboard and display for the user interface [1]. The virtual console provides an approach for the kernel to receive text input from the user and to send output to the user, which enables user interactions with the computer.

A web-based virtual console has been provided as the prototype of the project. (Shown in Figure 1). When the user types in the virtual console, the web server sends the input to the device, and sends back the output to the client-side. In this way, the application allows users to remotely interact with the device as if operating on the local host.

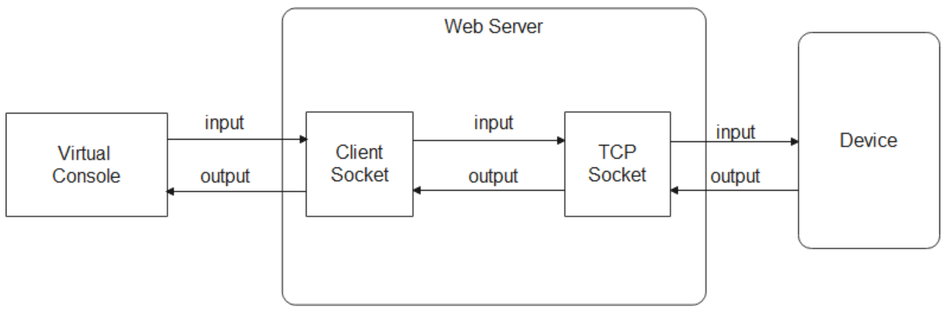


Figure 1. The original web-based virtual console

However, there are several disadvantages for the existed virtual console:

- Terminal commands are difficult to use for users with little background in Computer Science, especially for complex operations such as changing device configurations.
- The application lacks security measures for detecting and blocking malicious commands sent from users, thus the device is vulnerable to attacks such as code injection and data leaks.
- The system administrator is unable to provide feedback to users as a guideline.

Our project aims to solve the problems mentioned above by adding features to the current application. We need to build up a robust, efficient web application that can reduce the complexity for users to operate, prevent the device from malicious commands, and provide users with useful feedback.

In the next section, I will list what is the project aimed to achieve in detail. In section 3, I will describe the architecture and key process of the project, the software and tools we used to build the project, and the main tasks we did to meet these goals. In section 4, I will discuss what the project has achieved and where the goals of this project have been met. Finally, section 5 will conclude with a summary of the project outcomes and future work that would enhance its performance.

2. Project Aims

The project aims to extend the web-based virtual console by adding features as follows:

1. Sanitizing and intercepting the user input:

   To protect the virtual console against malicious commands, the project shall sanitize the user input and check the input against a blocklist. If the input contains any keyword in the blocklist, the input controller shall reject the input and inform the user that the input is illegal.

2. Setting up and running pre-defined scripts:

   - The project shall allow users to upload pre-defined scripts.
   - The user interface shall include a menu that displays the names of scripts in a dropdown list and allows the user to operate (execute/stop/delete) those scripts.
   - When the user chooses to run a pre-defined script, the project shall automatically execute the script within a time limit.
   - The project shall run a persistent script in the background that checks the user input and takes actions accordingly.

3. Providing feedback to users:

   The project shall generate feedback based on the input/output and display the feedback on the webpage.

3. Approach

3.1 Software Architecture

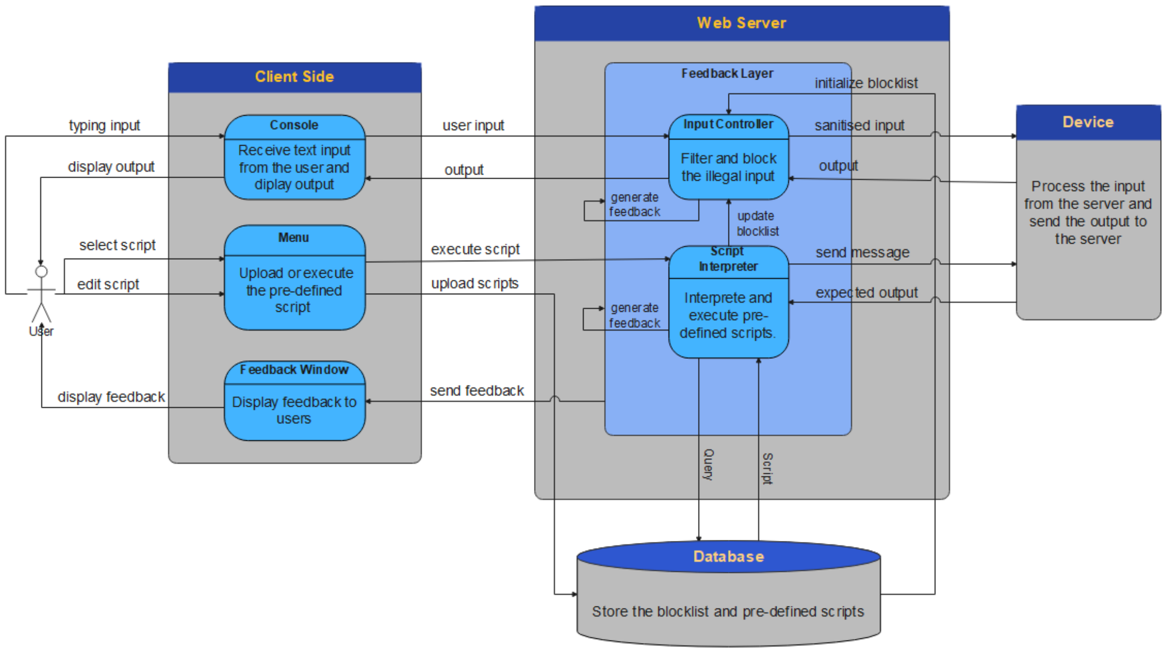


Figure 2. Software architecture of the project

The architecture of the project is shown in Figure 2. The project consists of four components: the client-side, a web server, a device, and a database.

3.1.1 Client-Side

The client-side runs on the browser, which includes three sections:

- A console process that allows text input and displays output sent from the device.
- A user menu that contains options for uploading scripts and a dropdown list that allows users to operate (execute/stop/delete) the script.
- A feedback window separated from the virtual console that displays feedback sent from the server.

3.1.2 Web Server

The web server locates between the client and the device, processes the data sent from both sides, and takes actions based on the input/output. The web server consists of three modules:

- Input controller: A function that sanitizes the user input and blocks illegal input.
- Script interpreter: A function that translates and executes instructions written in pre-defined scripts.
- Feedback layer: A simple function that is integrated with the input controller and the script interpreter. It receives the feedback generated by other modules and sends the

feedback to the user.

3.1.3 Device

Each virtual console connects to a single device. During the development of the project, the devices we connected to were Cisco routers at the Computer Networking lab. The router could process commands sent from the server and send the output to the server.

3.1.4 Database

A MongoDB database is created for storing the blocklist and pre-defined scripts. The server fetches the blocklist from the database for initializing the blocklist. The server can also fetch pre-defined scripts from the database for execution. Diagrams are listed in Appendix A for explaining the structure of the database.

3.2 Project Flow

Figure 3 describes the flow of the project. When the client-side connects to the server, the user can either choose to type input in the virtual console or execute the selected script.

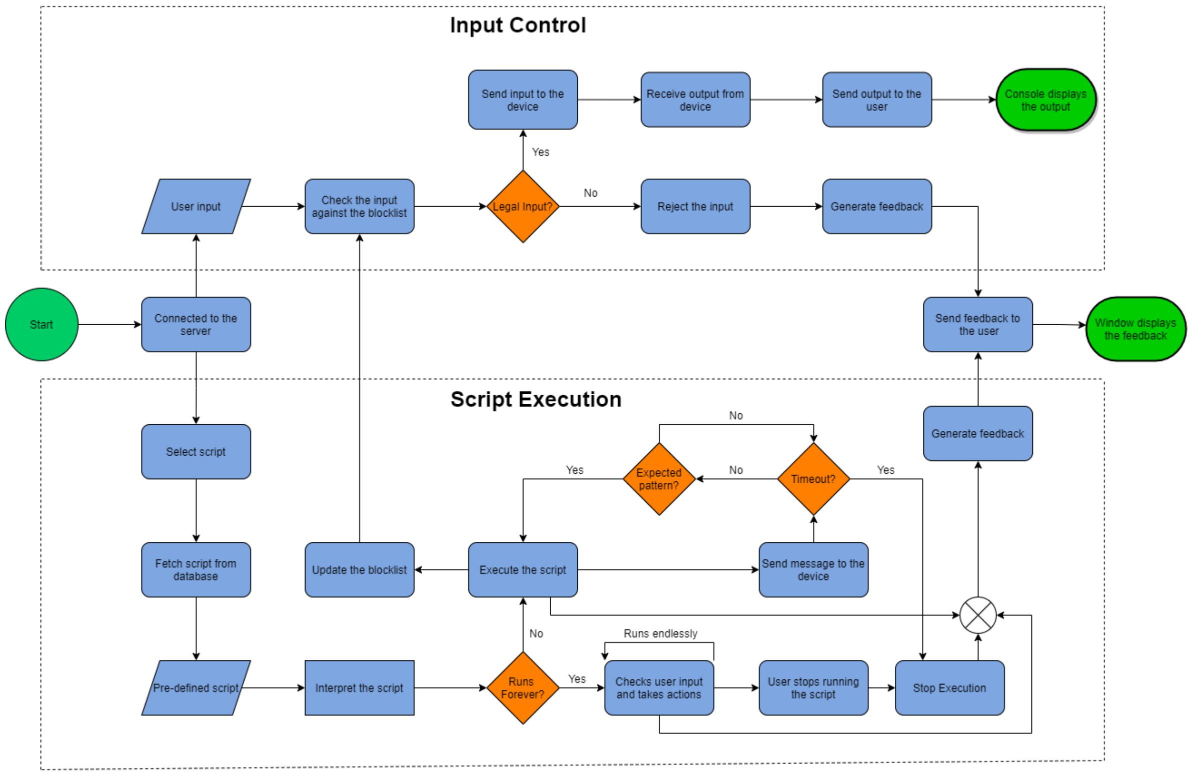


Figure 3. Flowchart for the automated checking and feedback system

3.2.1 Input Control

When the user types in the virtual console, the browser sends the user input to the server, where the input controller tests the input against the blocklist. If there is any illegal keyword in the user input, the server rejects the input and generates an error message as feedback. Otherwise, the server sends the input to the device. After receiving the output from the device, the server sends the output to the user. Users can see the output displayed in the console.

3.2.2 Script Execution

When the user chooses to run a pre-defined script, the server fetches the script from the database and sends the script to the script interpreter for execution. The interpreter first identifies the type of the script. If the script is persistent, it runs on the server-side to check the user input and take actions accordingly. Otherwise, the interpreter executes the script within a time limit. The script can take four types of actions: a) sending a message to the device; b) waiting for an expected pattern from the output. c) generate feedback defined in the script; d) update the blocklist.

If the server manages to receive the expected pattern before timeout, the interpreter continues running the remaining commands. Otherwise, the server stops executing the script and generates feedback containing the details of failure.

3.2.3 Providing Feedback

When the feedback system receives feedback generated from the above processes, it sends the feedback to the client-side, where the feedback window displays the feedback on the webpage.

3.3 Software and Tools

Xterm.js: Xterm.js is a standard console emulator written in TypeScript that allows text input and displays text output. Xterm.js was imported as the interface of the virtual console.

Node.js & Express: Node.js is a platform that uses event-driven, non-blocking models for building scalable network applications. Express is a Node.js web framework that provides features for building up web applications. We use Node.js and Express to build up the web server of the project.

Vue.js, HTML, CSS: The project uses Vue.js, HTML, and CSS to build up the user interface.

Socket.io: To establish communications between the client-side and the server-side, the project uses Socket.io to create the environment that handles events transmitted between the server and the client.

MongoDB: The project needs a data store that allows the user to store/update the data as needed. We implemented a MongoDB database to store the blocklist and pre-defined scripts.

3.4 Main Tasks

3.4.1 Input Limit/Interception

The project limits/intercepts the user input through the following processes:

1) Sanitizing the input:

First, the input controller sanitizes the user input to protect the device from attacks [2]. Each time the user types in an http-special character, the input controller instantly replaces it with a safe symbol (For example, replace "<" with "&lt") before appending it to the input.

2) Storing the input:

After receiving the sanitized character, the controller should correctly store the user input. There are special keys that can fail the input tests against the blocklist. For example, left/right arrows and backspace should change the input index instead of being stored in the input. Thus, I created a variable in the controller to record the index of the input. When those special keys are detected, the controller increases/decreases the index instead of appending them to the input so that it can store the input correctly.

3) Checking & intercepting the input:

Finally, the input controller checks the input against the blocklist and rejects the input that didn't pass the test. When the server starts running, it fetches the blocklist from the database for initialization. The blocklist is a customizable regular expression that forbids any keyword in it. If the input passes the test against the blocklist, the server directly sends the input to the device. Otherwise, the server abandons the current input and generates an error message. The error message gives details about what is not allowed in the input. For example, if the blocklist contains a keyword "sudo" which also appears in the user input, the feedback should be "Illegal input, your input shouldn't include: sudo".

Individual contribution for this part: I designed and implemented the above processes. I collaborated with [redacted] on constructing the blocklist, and I contributed all source code for the rest parts.

3.4.2 Script interpretation

Before executing the script, the first step for the interpreter is to translate the syntax of the pre-defined script into the executable format. I implemented script interpretation through the following procedure:

1) Reading the script: After fetching the pre-defined script from the database, the interpreter reads the script and breaks the script into single lines.
2) Translating the code: The interpreter loops through those lines and parses tokens of the pre-defined script, then translates the syntax into JavaScript. The syntax of the pre-defined script is listed in appendix B.
3) Error processing: When encountering syntax errors, the interpreter should throw an error and generate feedback instead of causing a server breakdown.

Individual contribution for this part: I designed the script interpretation process with [redacted] and I contributed all source code for this part. I defined the grammar of the pre-defined script referring to a script interpreter called runscript [3] and I implemented methods for translating the script. Besides, I implemented a database to store the pre-defined scripts for execution.

3.4.3 Script execution

After the interpreting process, the interpreter executes the script following the rules based on its type. There are two types of pre-defined scripts:

1) Persistent script:

The persistent script runs endlessly in the background. When the user input matches any keyword in the script, the script takes actions that pair with the keyword in response. For example, a keyword "address" is followed by actions {send "ip a"} and {reply "looking up for the IP address"}. When the user input includes "address", the server sends the command "ip a" to the device then provides feedback to the user.

2) Non-persistent script:

The Non-persistent script runs within a time limit. For simple commands such as sending messages to the device and generating feedback, the interpreter executes them sequentially without delay.

The critical command for the script execution is the block called "expect". Inside the block, the interpreter keeps checking if the output from the device contains the expected pattern. Before receiving the expected pattern or timeout reached, it suspends the execution of the following instructions. When the script fails to receive the expected pattern within the time limit, the interpreter terminates the script immediately and generates a failure message. For example, if the script expected a "#" pattern but failed to receive it from the device within 3 seconds, the interpreter would stop executing the script and generate feedback "Failed to receive the expected pattern #, stop executing the script".

Individual contribution: I designed the above process and I contributed all source code for this part. I built the asynchronous "expect" command by learning the runscript interpreter and using JavaScript Promise process [4].

3.4.4 Feedback system

The feedback system is a simple function that receives feedback generated by the input controller or the script interpreter and sends feedback to the client through Socket.io. When the client-side receives the feedback, the browser displays the feedback in the feedback window.

Individual contribution: I designed the feedback system with [redacted] and contributed the source code for this part. I also took part in designing the feedback window on the client-side.

4 Results

4.1 Achievements

The project has completed all activities listed in the milestones. In milestone 1, we implemented an input controller. It has two functions:

- It replaces http-special characters in the user input with safe symbols.
- It checks user input and rejects illegal input that contains any keyword in the blocklist. When the input is rejected, the application displays an error message on the webpage.

Apart from the functions mentioned above, users are allowed to customize the blocklist from the menu on the webpage. The input controller provides the application with effective measures to block malicious commands.

In milestone 2, we designed pre-defined scripts and implemented a script interpreter, which executes pre-defined scripts automatically. The interpreter can execute two types of scripts:

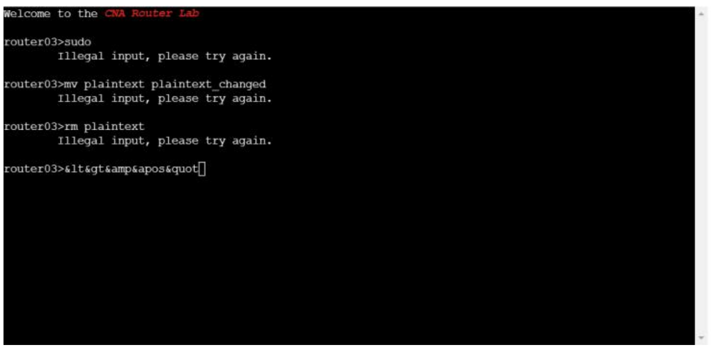
- The time-limited script that executes commands automatically. It can execute simple commands including sending commands to the device and providing arbitrary feedback to the user, and the "expect" command that takes actions based on the output from the device.
- The persistent script that runs continuously on the server. When running on the server, the script checks the user input and takes actions when the keyword defined in the script is detected in the input.

The script interpreter enables users to operate the device without typing complex commands in the console, which reduces the difficulty for users to use the console.

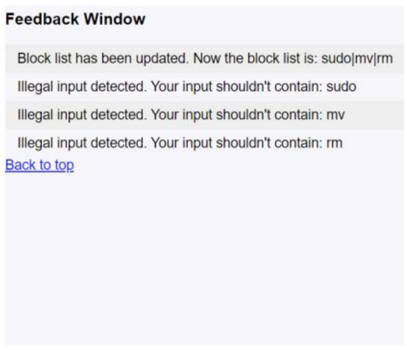
Finally, we have also implemented a feedback system that gathers feedback generated from other modules and provides users with useful feedback.

4.2 Examples to verify the results

Figure 4 and figure 5 are screenshots of test cases that verify part of the results in milestone 1 and milestone 2.



```
Welcome to the CNA Router Lab

router03>sudo
        Illegal input, please try again.

router03>mv plaintext plaintext_changed
        Illegal input, please try again.

router03>rm plaintext
        Illegal input, please try again.

router03>&lt&gt&amp&apos&quot
```



Feedback Window

Block list has been updated. Now the block list is: sudo|mv|rm

Illegal input detected. Your input shouldn't contain: sudo

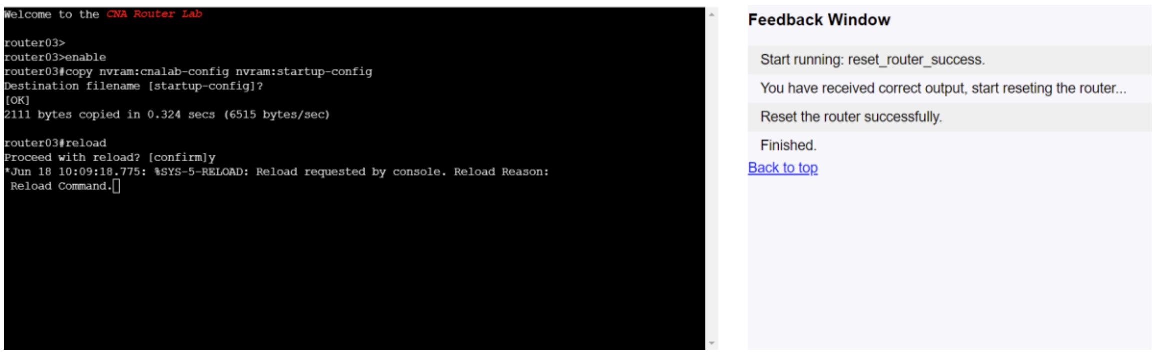
Illegal input detected. Your input shouldn't contain: mv

Illegal input detected. Your input shouldn't contain: rm

Back to top

Figure 4 Test cases for input control

In figure 4, when the user types in "<", ">", "&", ", ', the left and right brackets (both are http-special characters) in the input are replaced by “&lt”, “&gt”, “&amp”, “&quot”, and “&apos”, which proves that the sanitizing process works as expected. When the user input contains a keyword in the blocklist, the input is rejected and the user can see an error message displayed in the feedback window.



```
Welcome to the CNA Router Lab

router03>
router03>enable
router03#copy nvram:cnalab-config nvram:startup-config
Destination filename [startup-config]?
[OK]
2111 bytes copied in 0.324 secs (6515 bytes/sec)

router03#reload
Proceed with reload? [confirm]y
*Jun 18 10:09:18.775: %SYS-5-RELOAD: Reload requested by console. Reload Reason:
 Reload Command.
```

Feedback Window

Start running: reset_router_success.

You have received correct output, start reseting the router...

Reset the router successfully.

Finished.

Back to top

Figure 5 An example for running a pre-defined script

In figure 5, the user selects and executes a pre-defined script that automatically resets the router and provides feedback defined in the script. We can observe from the console that the script sends and receives messages in the correct sequence, and the router is reset successfully. Meanwhile, the user can see the feedback displayed in the feedback window.

Examples for using the user menu, executing persistent scripts, and the test case when the interpreter fails to execute the script are provided in Appendix C.

4.3 Individual Contribution

During the lifecycle of the project, I was responsible for developing the features required in the project. Apart from the user interface, I contributed the vast majority of source code (Mentioned in the descriptions of GitHub commits history) including:

1. Checking and limiting user input: I designed and contributed all source code for the input sanitization and input checking processes; I designed the blocklist with Kaifeng Chen and I contributed all source code for the input limiting process.
2. Setting up and running pre-defined scripts: I designed the syntax of pre-defined scripts and contributed all source code for the script interpretation process. I designed the script execution process with Kaifeng Chen and I contributed all source code for this part.
3. Providing arbitrary feedback to users: I designed and contributed the source code for generating and sending the feedback to the client-side.
4. Database: I implemented a MongoDB database to store the blocklist and pre-defined scripts.
5. Testing: I conducted all defect testing and unit testing during the development process. After the development process, I was in charge of conducting integration testing (Mentioned in the testing plan).

5 Conclusion

5.1 Project Summary

The final version of the project has integrated all features to form a safe, flexible web application. The project can detect and limit user input to protect the device against malicious commands, execute pre-defined scripts automatically to reduce the complexity for operating the console, and provide useful feedback to users where needed.

5.2 Future Directions

1. Differentiate user scenarios:

According to the user requirement, we don't need to distinguish between ordinary users and the system administrator, which means the user can change system configurations on the client-side (For example, customize the blocklist and upload pre-defined scripts). However, we shouldn't allow users to do so in real-life scenarios. In the future, we can establish a login system to distinguish users from users and users from the system admin.

2. Setup remote connections to the server:

The current application only allows users to connect to the server on the localhost. In the future, we will allow users to connect to the server via a website rather than on their own computers.

3. Extensions to the script interpreter:

We may add additional features to the current script interpreter to extend its functions. For example, we can add a feature that allows the system administrator to choose to show/hide output in the console.

References

[1] Wikipedia 2021, Virtual Console, *Wikipedia*, viewed 15 March 2021, <https://en.wikipedia.org/wiki/Virtual_console>.

[2] PHP Manual 2000, htmlspecialchars*, PHP Documentations*, viewed 1 April 2021, <https://www.php.net/manual/en/function.htmlspecialchars.php>.

[3] Miquel, S. and Jukka, L. 2007, runscript(1): script interpreter for minicom, *Linux man page*, viewed 5 April 2021, <https://linux.die.net/man/1/runscript>.

[4] MDN Web Docs 2016, Promise – JavaScript, *MDN,* viewed 21 April 2021, <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise>.

Appendix A - Database Diagrams

Collection: block_list		
Column	**Data Type**	**Description**
id	Id	Primary key for the blocklist.
name	String	Name of the blocklist.
content	Regular Expression	The content of the blocklist, which is a regular expression that checks if the input includes any keywords in it.
words	Array	An array that contains the keywords of the blocklist. It is used for reconstructing the blocklist when the blocklist is updated.

Collection: test_script		
Column	**Data Type**	**Description**
id	Id	Primary key for the pre-defined script.
name	String	Name of the script.
content	String	Contents of the pre-defined script.

Appendix B - Syntax of Pre-defined Scripts

1. timeout value

Appears at the first line of the script. The value is either an integer that represents number of seconds as the time limit for executing the script, or "forever" that indicates the script runs persistently on the server.

2. send "message"

send a message to the device.

3. expect

```
expect {
    ("pattern" statement* '\r')*
    (timeout value statement*)?
}
```

This command keeps reading the output sent from the device until it reads the expected pattern. If it is followed by any statement, it will execute it. If it fails to receive the expected pattern when timeout is reached, the script terminates immediately.

4. reply "feedback"

Generate feedback and send the feedback to the feedback system.

5. setblock "keywords"

Add keywords to the blocklist.

6. reset

Clear the blocklist.

7. block

Block the user input and generates an error message.

8. stop

Terminate the script immediately.

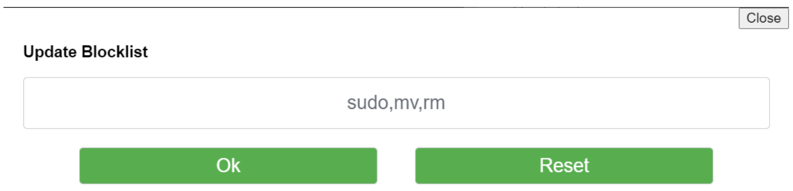
9. ifincludes "keywords" (action)*

Syntax in the persistent script. It detects whether the user input contains the keyword and takes actions accordingly.

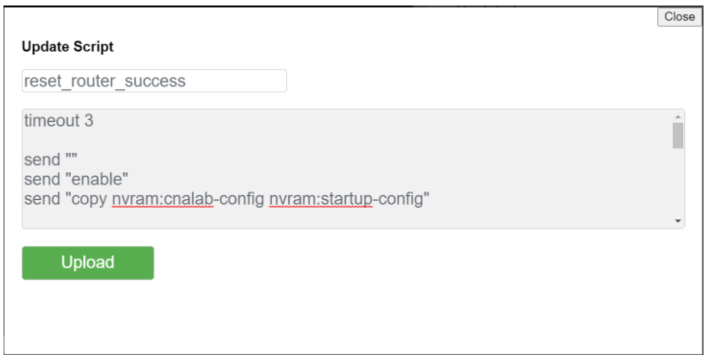
Appendix C - Screenshots for the Results

1. Examples for using the user menu

1) Set the blocklist:

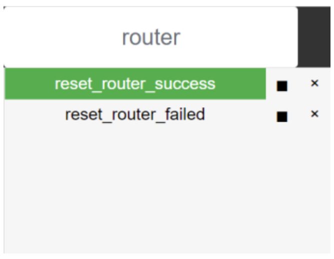


2) Upload a script:



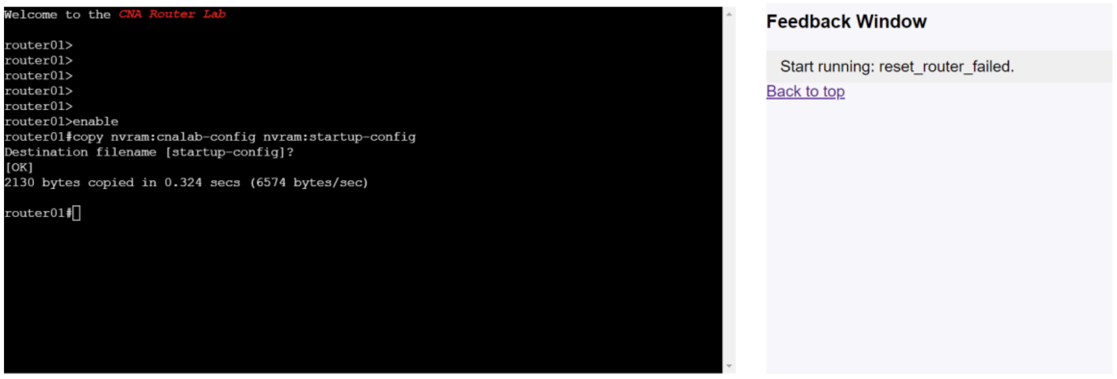
```
timeout 3

send ""
send "enable"
send "copy nvram:cnalab-config nvram:startup-config"
```

3) Search and select the script:



2. An example when the script failed to receive expected pattern. The script waited until timeout, then it provided feedback in the feedback window.

Waiting for expected pattern:



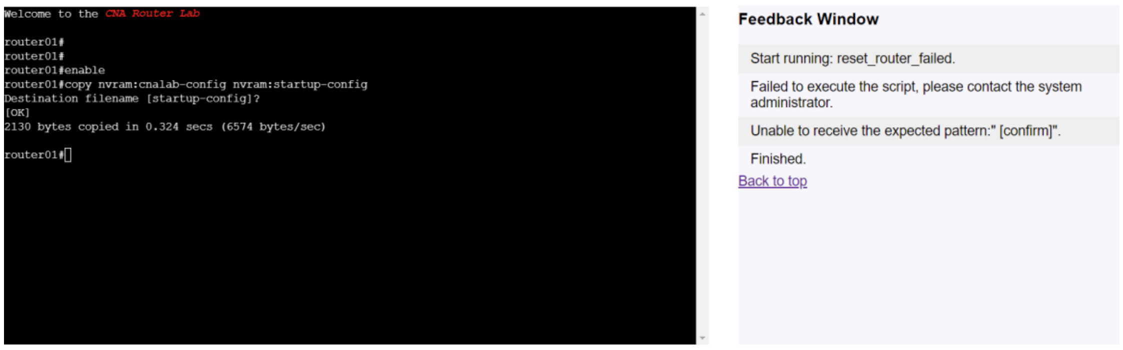
```
Welcome to the CNA Router Lab

router01>
router01>
router01>
router01>
router01>
router01>enable
router01#copy nvram:cnalab-config nvram:startup-config
Destination filename [startup-config]?
[OK]
2130 bytes copied in 0.324 secs (6574 bytes/sec)

router01#
```

Feedback Window

Start running: reset_router_failed.

Back to top

Timeout:



```
Welcome to the CNA Router Lab

router01#
router01#
router01#enable
router01#copy nvram:cnalab-config nvram:startup-config
Destination filename [startup-config]?
[OK]
2130 bytes copied in 0.324 secs (6574 bytes/sec)

router01#
```

Feedback Window

Start running: reset_router_failed.

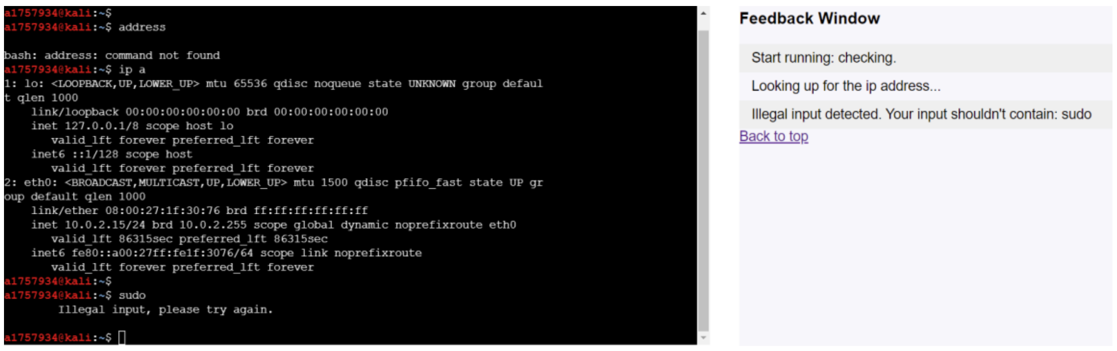
Failed to execute the script, please contact the system administrator.

Unable to receive the expected pattern:" [confirm]".

Finished.

Back to top

3. An example for executing persistent script. It looks up the IP address when the user types “address” and blocks the input that contains “sudo”.



```
a1757934@kali:~$
a1757934@kali:~$ address

bash: address: command not found
a1757934@kali:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP gr
oup default qlen 1000
    link/ether 08:00:27:1f:30:76 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute eth0
       valid_lft 86315sec preferred_lft 86315sec
    inet6 fe80::a00:27ff:fe1f:3076/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
a1757934@kali:~$
a1757934@kali:~$ sudo
        Illegal input, please try again.

a1757934@kali:~$
```

Feedback Window

Start running: checking.

Looking up for the ip address...

Illegal input detected. Your input shouldn't contain: sudo

Back to top