Individual Game Technical Writeup

Table of Contents

- Introduction
- Open Source Intelligence (OSINT)
- Cryptography
- Password Cracking
- Log Analysis
- Network Traffic Analysis
- Forensics
- Scanning & Reconnaissance
- HMAC Integrity Verification
- Conclusion

Introduction

This document provides a technical analysis of my participation in the NCL Individual Game challenges, breaking down the methodology, solutions, and techniques used across multiple categories.

Open Source Intelligence (OSINT)

Challenge 1: Code of Conduct

Verified competition terms and conditions:

- No peer collaboration allowed
- Responsible AI usage required

Challenge 2: Honor (Easy)

Analyzed an image from a data breach using ExifTool:

```
exiftool dog.jpg
```

Key Findings:

- Discovered hex-encoded flag in copyright metadata: `534B592D4C494D492D31333337`
- Decoded to ASCII: `SKY-LIMI-1337`

Challenge 3: Controversial Challenge (Medium)

Identified messaging platform and associated individual:

- Visual analysis revealed Signal platform
- Researched "signalgate" context
- Identified "S M" as Stephen Miller

Challenge 4: Nostalgia (Hard)

Analyzed historical photo location:

- Identified Dutch text "Electr Klompenmakeru" (Electric Clog Maker)
- Located business as Ratterman Wooden Shoes
- Address: Noorddammerlaan 22, 1185, ZA, Amstelveen

Challenge 6: Github in Action (Hard)

Investigated GitHub repositories:

- Found user spmedia's website: http://edmond.ma/jor
- Identified anti-phishing repo: PhishingSecLists

- Located pull request by Ashley Tolbert

Cryptography

Challenge 1: The Bases (Easy)

Decoding various encoded strings:

1. Binary to ASCII:
   - Original Message: `01001011 01100001 01101110 01100111 01100001 01010010 00110000 00110000 00110101 00111000`
   - Decoded Result: `KangaR0058`
2. Octal to ASCII:
   - Original Message: `113 145 160 162 151 127 141 164 151 070 063`
   - Decoded Result: `KepriWati83`
3. Octal to ASCII:
   - Original Message: `052 052 051 053 115 117 110 100 105 097 108`
   - Decoded Result: `4435sundial`

Challenge 2: Super Shift (Easy)

Various shift ciphers to decode:

1. Caesar Cipher (shift 13):
   - Original Message: `84cbagvnpfhaoveq`
   - Decoded Result: `84pontiacsunbird`
2. Caesar Cipher (shift 5):
   - Original Message: `Mfnwiwjxxnsl3`
   - Decoded Result: `Hairdressing3`
3. Morse Code + Caesar Cipher (shift 17):
   - Original Message: `-.. --.. - -.-- ..-. . . ...- --.. .--- ... .-. - -...`
   - Intermediate Result: `Dztyfeevzjsrtb`
   - Final Decoded Result: `Michonneisback`

Challenge 3: Pizza Time (Easy)

Rail fence cipher decoding:

1. Rail Fence (key 3):
   - Original Message: `Usezfiysnestpzasrmtltsilhiioaot`
   - Decoded Result: `Unlesshtepizzaisfromitalytossit`
2. Rail Fence (key 6, offset 1):
   - Original Message: `sdIslrstneoathdnowhgakkoeirtl`
   - Decoded Result: `Indarknesslooktowardsthelight`

Challenge 4: Signed (Medium)

Verified GPG signatures with a script that:

- Loops through signature files
- Verifies each against corresponding document
- Reports verification failures

```
# Key functionality of gpg_verify.sh
for sig_file in *.sig; do
  original_file="${sig_file%.sig}"
  # Verify signature and report results
  gpg --verify "$sig_file" "$original_file"
  # Report success or failure
done
```

Findings:

1. Tampered file: Email_48.txt containing HEX `4b767470753861707675a`
2. Decoded to: `Domin8tion`

Password Cracking

Challenge 1: Hash me outside! (Easy)

Generated hashes for provided passwords:

1. MD5 hash: `2b164ea92dbd46f72318e55ec634a83a` `echo -n "white1561lotus" | md5sum`
2. SHA1 hash: `516f652fa03b7f711ada6a1acdd9786cde89dc8a` `echo -n "6891JasmineDragon" | sha1sum`
3. SHA256 hash: `0b09f41de4769201222f1e4a42acdc3a63750700f5d3fd2b37eb643282ba303c` `echo -n "317paisho698" | sha256sum`

Challenge 2: We Will RockYou (Easy)

Cracked MD5 hashes with the RockYou wordlist:

- `Hashcat -m 0 wewillrockyou.txt /usr/share/wordlists/rockyou.txt`
- `3da1dd44e86ce30ff07d32065e9b68c3`: queen24
- `5243dc768dfca6d80993b4803bed95e4`: freddie11
- `a6c8f8fe09042f4ab28d0048575cd9d4`: mercury5134

Challenge 3: Oph the Grid (Medium)

Used Ophcrack with rainbow tables to crack Windows LM/NTLM hashes:

Process:

1. Downloaded rainbow tables
2. Created hash file
3. Imported hashes and tables into Ophcrack
4. Obtained results:
   - `BA50E19589950A959C5014AE4718A7EE:74A942B14C50D4ED03D9A4CC8866199C`: 25332535
   - `2A2A4346DDF2630ABFE061A5F4DBCCB2:8295ABD305A649454D0709E350A8C601`: shanegrace
   - `876CA1B27D8B258D6966CA05A9CDAE2D:F8F9BF0623560DA52451C1893821087A`: beautyelaine

Challenge 4: Totally Safe PDF (Medium)

1. Used a pdf2john script that:

- Extracts encryption parameters from PDFs
- Formats hash for John the Ripper

- Tracks encryption algorithm, permissions, and revision

```
# Key functionality of pdf2john.py
# Extract PDF security parameters
# Format hash string for John the Ripper
# Example: "$pdf$4*4*128*-3904*1*16*hash-data-here*32*more-hash-data*32*final-hash-part"
```

2. Ran John the Ripper against the hash:
    - Used rockyou.txt wordlist for password cracking

Results:

- Password: pdfscott86
- Flag: SKY-PDFS-2472

Challenge 5: put0nth3ma5k (medium)

Used John the Ripper with mask attack:

- Pattern: "SKY-MASK-?a?a?a?a"
- `john -mask="SKY-MASK-?a?a?a?a" put0nth3ma5k.txt` **Results:**
- `$1$MASK$.IxEV.UcEJNjNX.UCE7/A/`: SKY-MASK-2552
- `$1$MASK$u2nKEYGuYo1DYtu2K/yAn/`: SKY-MASK-4778
- `$1$MASK$56Jw4nrfMvazyiOu68Lge.`: SKY-MASK-9310

Log Analysis

Challenge 1: Ancient History (Easy)

Analyzed HTTP logs:

Queries Used:

1. Extracted domain of the third standard HTTP request:
    - Result: http://httpforever.com/js/init.min.js
2. Identified timestamp for server response:

- Result: 1743959680.158

3. Found IP of www.delta.com in first CONNECT request:
   - Result: 96.16.70.40
4. Counted NONE_NONE/000 errors:
   - Result: 40
5. Counted successful connections to push.services.mozilla.com:
   - Result: 134
6. Counted total POST requests:
   - Result: 8
7. Found third most accessed domain:
   - Result: firefox.settings.services.mozilla.com

Challenge 2: Leaked (Medium)

Analyzed leaked SQL data from a social media database:

SQL Queries:

1. Count of compromised users:
   - Simple count query revealing 982 affected users
2. First account join date:
   - Identified earliest timestamp
   - Converted to: March 22, 2025 12:01:38 AM UTC
3. Email with most followers:
   - Sorted by follower count in descending order
   - Result: kvolantge@cityinthe.cloud (1000 followers)
4. Count of verified users:
   - Filtered for verified status
   - Result: 464 verified users
5. Most common phone area code state:
   - Extracted area codes from phone numbers
   - Counted occurrences and identified most common
   - Result: 364 (Kentucky)

Challenge 3: Logins (Hard)

Created login analysis scripts that:

- Parse binary log format with proper byte offsets
- Calculate suspicion scores based on multiple factors
- Identify potentially compromised accounts

Methodology:

- Parsed binary login logs with custom script
- Tracked suspicious login patterns including:
  - Multiple IP addresses per user
  - Failed login attempts followed by successful logins
  - Unusual login hours (1am-5am)
  - Successful logins after multiple failures
  - Geographic anomalies based on IP patterns
- Implemented weighted scoring system for suspicious behaviors
- Calculated comprehensive suspicion scores for each user
- Identified most likely compromised account based on highest score

```
# Key functionality of binary_parser.py
def parse_binary_logs(file_path):
    # Read file and parse using correct binary structure:
    # - Username length (4 bytes)
    # - Username (variable length)
    # - IPv4 address (4 bytes)
    # - Timestamp (4 bytes)
    # - Success flag (1 byte)

# Enhanced in compromised_user_detector.py with:
# - Login time analysis (unusual hours)
# - Repeated failure pattern detection
# - Geographic anomaly detection
# - Comprehensive scoring algorithm
```

Results:

1. Log start date (UTC): March 18, 2024

2. Login attempt events recorded: 3579
3. Unique usernames: 174
4. Unique IP addresses: 208
5. Compromised user: [Username identified by algorithm]

Network Traffic Analysis

Challenge 1: Lost in Resolution (Easy)

Analyzed DNS traffic in a PCAP file:

Wireshark Filters:

1. DNS transaction ID in frame 36:

   ```
   frame.number == 36
   ```

   Result: 0x75b8

2. Email provider:

   ```
   udp.port == 53 or tcp.port == 53
   ```

   Result: Proton (from packet 2505)

3. Second A record for chatgpt.com in frame 10061:

   ```
   frame.number == 10061
   ```

   Result: 172.64.155.209

4. Transaction ID for first pwn.college query:

   ```
   dns.qry.name == "pwn.college"
   ```

   Result: 0xaeee

5. Flag in DNS records for flag.com.localdomain:

   ```
   dns.qry.name contains "flag.com.localdomain"
   ```

   Result: SKY-DENS-5353

Challenge 2: Wifi (Medium)

Analyzed WiFi capture files:

- Identified router MAC: c0:4a:00:80:76:e4
- ESSID: Wii Fii
- Victim MAC: 02:38:aa:ae:9f:e6
- Channel: 4
- Cracked password with aircrack-ng: soccer17
- `aircrack-ng -w /usr/share/wordlists/rockyou.txt wifi.cap`

Forensics

Challenge 1: Overuse (Easy)

Analyzed image with hidden data:

```
strings ForYou.jpg  # Revealed embedded filenames
binwalk -e ForYou.jpg  # Extracted hidden files
```

Used a steganography script that:

- Extracts LSB data from images
- Analyzes color planes for anomalies
- Detects hidden files by signatures

```
# Key functionality of steg.py
# Extract LSB data
# Check image metadata
# Analyze bit distribution
# Look for file signatures
```

Process:

1. Extracted strings:

```
strings ForYou.jpg
```

Discovered file names:

- 1Scroll.jpg
- 2NeverGoingToGIve.txt
- 3Sky.jpg
- 4Congrats.txt
- 5Wise.jpg
- 6Bussin.txt
- 7Buzz.jpg
- 8More.txt

2. Extracted hidden files:

```
binwalk -e ForYou.jpg
```

3. Found flags:
   - From 6Bussin.txt: SKY-BUSN-4419
   - Base64 encoded flag: SKY-UATE-1057

Results:

- Found flags in embedded files:
  - SKY-BUSN-4419
  - SKY-UATE-1057

Challenge 2: Oops (Medium)

Recovered deleted file from a disk image:

Approach:

1. Opened image in Autopsy forensic tool
2. Navigated to deleted files section
3. Recovered file with flag: SKY-UNDL-3373

Challenge 3

Analyzed a file to determine its original format:

Findings:

1. Identified as 3D printing instructions (STL file)

Scanning & Reconnaissance

Challenge 1: Portscan (Easy)

Network reconnaissance:

```
ifconfig
nmap 10.8.93.0/24
nmap -sV 10.8.93.100
curl 10.8.93.100/flag.txt  # Found SKY-HTTP-4553
```

Findings:

1. Lowest port: 17
2. Highest port: 4000
3. Service on port 80: version 0.6
4. Flag found with curl command:

   ```
   curl 10.8.93.100/flag.txt
   ```

   Result: SKY-HTTP-4553

Challenge 2: Dig (Medium)

DNS record investigation:

- Used dig to query various record types
- Found IPv4, IPv6, TXT records
- Discovered flag: SKY-XJPO-5751

Results:

1. IPv4 addresses: 23.151.187.212, 43.71.247.55
2. IPv6 address: 2ecd:b3d:2f0c:e72b:da9:f4ee:81e:d62d
3. Flag TXT record: SKY-XJPO-5751
4. Redirect domain: r3d1r3ct3d.liber8.cityinthe.cloud
5. TTL for redirect: 600 seconds
6. Primary mail exchanger: mx1.liber8.cityinthe.cloud

HMAC Integrity Verification

Created HMAC verification script that:

- Calculates SHA-256 HMAC signatures
- Verifies file integrity
- Detects tampering patterns in DNS records

```
# Key functionality of hmac_integrity_checker.py
def calculate_hmac(message):
    # Calculate HMAC using SHA-256

def verify_hmac(message, signature):
    # Verify using constant-time comparison

def detect_tampering(log_entry):
    # Check for suspicious domains/patterns
    # Score risk level
    # Identify possible attack vectors
```

Script Functionality:

- Calculates HMAC signatures for messages using the SHA-256 algorithm
- Verifies signatures using constant-time comparison to prevent timing attacks
- Processes batches of message and signature files
- Identifies mismatched or tampered message/signature pairs
- Analyzes tampering patterns for potential security threats

The tool enabled me to:

1. Count messages with mismatched HMACs
2. Identify stored HMAC values and validate message integrity

Conclusion

The NCL Individual Game required diverse cybersecurity skills including:

- Scripting (Python, Bash)
- Forensic analysis
- Cryptography
- Network traffic analysis
- Database queries
- Steganography
- Password cracking

Custom tools were essential for automating complex tasks and providing deeper insights into the challenge data. The experience demonstrated the importance of both technical depth and breadth in cybersecurity analysis.

Revision #3
Created 2025-11-25 18:01:15 UTC by David Rizzo
Updated 2025-11-25 21:52:20 UTC by David Rizzo