CTF Categories

- Web
- PWN
- Crypto
- Reverse Engineering
- Forensics
- Misc
- Hardware

Web

A Guide to Solving Web Challenges in Capture The Flag (CTF)

Welcome to the dynamic world of Web challenges in Capture The Flag (CTF) competitions! Web challenges test your understanding of web technologies, security vulnerabilities, and your ability to think like an attacker. This guide is designed to help you navigate common web vulnerabilities and develop strategies to tackle these challenges effectively.

Table of Contents

Understanding Web Challenges

Web challenges in CTFs are designed to assess your ability to find and exploit vulnerabilities in web applications. These challenges may involve:

- Exploiting common web vulnerabilities like SQL Injection, XSS, and File Inclusion.
- Understanding server configurations and exploiting misconfigurations.
- Bypassing authentication mechanisms.
- Interacting with web technologies such as HTML, JavaScript, HTTP protocols, and more.

The key to success lies in methodically analyzing the web application and identifying potential weaknesses.

General Approach

1. **Information Gathering**:
   - Explore the web application thoroughly.
   - Identify input fields, parameters, and functionality.
2. **Understanding the Application**:
   - Determine the technologies used (e.g., PHP, Flask, databases).
   - Look for clues in URLs, form actions, HTTP headers, and cookies.
3. **Testing for Vulnerabilities**:
   - Use manual testing techniques to probe for weaknesses.
   - Inject test inputs to observe how the application responds.
4. **Analyzing Responses**:
   - Pay attention to error messages, unusual responses, and behavior changes.
   - Collect and interpret any feedback from the server.
5. **Exploiting Vulnerabilities**:
   - Develop and refine payloads to exploit identified vulnerabilities.

- Ensure that your exploits are safe and controlled.

6. **Extracting the Flag**:
   - Once exploited, retrieve the hidden information or flag.
   - Document your steps for future reference.

Tools of the Trade

Equip yourself with essential tools for web penetration testing:

- **Web Browsers with Developer Tools**:
  - Chrome, Firefox with the ability to inspect elements, view source, and monitor network activity.
- **Proxy Tools**:
  - **Burp Suite**: Intercept and modify HTTP requests and responses.
  - **OWASP ZAP**: Security testing tool with interception capabilities.
- **Command Line Tools**:
  - **cURL**: Transfer data with URLs, test HTTP requests.
  - **Wget**: Retrieve files from web servers.
- **Scanning and Enumeration Tools**:
  - **Nmap**: Network scanning and port discovery.
  - **dirb**: Content scanning and directory brute-forcing.
- **Specialized Tools**:
  - **sqlmap**: Automated SQL injection exploitation.
  - **XSStrike**: Advanced XSS detection and exploitation suite.
- **Online Resources**:
  - **Payloads All The Things**: Collection of payloads and bypasses.

Common Web Vulnerabilities

Understanding common vulnerabilities is crucial. Below, we discuss several prevalent ones and how to approach them.

SQL Injection

Understanding SQL Injection

SQL Injection occurs when user input is improperly sanitized, allowing an attacker to execute arbitrary SQL commands. This can lead to unauthorized data access or manipulation.

Techniques and Tips

- **Identify Injection Points**:
  - Test input fields and URL parameters with special characters like `'`, `"`, `--`, `;`.
- **Observe Error Messages**:
  - Errors indicating syntax issues may reveal injectable points.
- **Test for Boolean-Based Injection**:
  - Use inputs that result in true or false conditions to infer database responses.
  - Example: `input' OR '1'='1`
- **Use UNION Selects**:
  - Attempt to retrieve data from other tables.
  - Determine the number of columns with `ORDER BY` or `UNION SELECT NULL`.
- **Extract Data**:
  - Once injection is confirmed, craft payloads to retrieve sensitive information.
  - Example: `UNION SELECT username, password FROM users`
- **Bypass Filters**:
  - Modify payloads to evade simple sanitization.
  - Use URL encoding, case variations, or comments.
- **Automate with Tools**:
  - If manual testing is difficult, consider using `sqlmap` for automation.

Cross-Site Scripting (XSS)

Understanding XSS

XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by other users. This can lead to session hijacking, defacement, or redirection.

Techniques and Tips

- **Test Reflected Inputs**:
  - Input `<script>alert('XSS')</script>` in fields and see if it gets executed.
- **Explore Different Contexts**:
  - Try injections in HTML, JavaScript, CSS contexts.
  - Adjust payloads accordingly.
- **Bypass Filters and Protections**:
  - Use variations to bypass input validation.
  - Example: `"><script>alert('XSS')</script>`
- **Use Event Handlers**:
  - Attach scripts to events if tags are filtered.
  - Example: `<img src=x onerror=alert('XSS')>`

- **Leverage Protocols**:
  - Use `javascript:` protocol in URLs if applicable.
  - Example: `javascript:alert('XSS')`
- **Test Stored XSS**:
  - Check if inputs are stored and rendered elsewhere.
- **Use Browser Developer Tools**:
  - Inspect elements and modify the DOM to test hypotheses.

Server-Side Template Injection (SSTI)

Understanding SSTI

SSTI occurs when user input is embedded unsafely in server-side templates, potentially leading to code execution.

Techniques and Tips

- **Identify Template Engines**:
  - Look for signs of engines like Jinja2, Twig, or Freemarker.
- **Inject Template Expressions**:
  - Use placeholders like `{{7*7}}` or `${7*7}` and see if the result is evaluated.
- **Escalate to Code Execution**:
  - If expressions are evaluated, attempt to access sensitive functions or attributes.
  - Example: `{{config.items()}}`
- **Bypass Filters**:
  - Modify payloads to evade input restrictions.
  - Use alternative syntax or encodings.
- **Understand the Context**:
  - Determine how input is being processed by the template engine.
- **Exploit Safely**:
  - Be cautious with payloads that could disrupt the server.

File Inclusion Vulnerabilities

Understanding File Inclusion

File inclusion vulnerabilities occur when a web application allows unauthorized inclusion of files, potentially leading to arbitrary code execution.

Techniques and Tips

- **Test for Local File Inclusion (LFI)**:

    - Modify parameters to include local files.
    - Example: `?page=../../etc/passwd`
- **Test for Remote File Inclusion (RFI)**:
    - Attempt to include remote resources if possible.
- **Bypass Filters**:
    - Use encoding or alternative path representations.
    - Example: `%252e%252e%252f` for double-encoded `../`
- **Leverage Null Byte Injection**:
    - In some cases, appending a null byte (`%00`) can bypass extensions.
- **Combine with Other Vulnerabilities**:
    - Use LFI to read logs or session files containing sensitive data.
- **Write to Files**:
    - If possible, find ways to upload or write content that can be included.
- **Monitor Server Responses**:
    - Error messages can indicate whether inclusion attempts are working.

Cross-Origin Resource Sharing (CORS) Exploits

Understanding CORS Issues

CORS policies control how web applications interact with resources from different origins. Misconfigurations can allow unauthorized cross-origin requests.

Techniques and Tips

- **Inspect CORS Headers**:
    - Use browser developer tools to view `Access-Control-Allow-Origin` headers.
- **Test Origin Reflection**:
    - Check if the server reflects the `Origin` header value in responses.
- **Exploit Wildcard Origins**:
    - A wildcard `*` in `Access-Control-Allow-Origin` with sensitive responses can be problematic.
- **Check for Credential Leakage**:
    - See if `Access-Control-Allow-Credentials` is `true` alongside a wildcard origin.
- **Craft Malicious Requests**:
    - Create cross-origin requests to access restricted data.
- **Use JavaScript Fetch/AJAX**:
    - Write scripts to perform cross-origin requests and process responses.
- **Bypass Preflight Checks**:
    - Manipulate request methods and headers to avoid CORS preflight.

Additional Tips and Resources

- **Read the Documentation**:
  - Understanding how web technologies work aids in finding vulnerabilities.
- **Stay Updated on Vulnerabilities**:
  - Web security evolves rapidly; keep learning about new exploits.
- **Think Like an Attacker**:
  - Consider how user input can be manipulated in unexpected ways.
- **Practice Regularly**:
  - Use platforms like **Hack The Box**, **PortSwigger Web Security Academy**, and **OWASP Juice Shop**.
- **Collaborate and Discuss**:
  - Engage with communities on forums and chat groups to share knowledge.

Helpful Links

- **OWASP Top Ten**:
  - Familiarize yourself with common vulnerabilities.
  - OWASP Top Ten Project
- **Web Security Tutorials**:
  - **The Web Application Hacker's Handbook** by Dafydd Stuttard and Marcus Pinto.
- **Cheat Sheets**:
  - **OWASP Cheat Sheet Series**: Best practices for web security.

Final Thoughts

Web challenges require a blend of creativity, technical knowledge, and persistence. They not only test your understanding of web application security but also your problem-solving skills.

Remember, always approach challenges methodically. Start with information gathering, hypothesize, test, and iterate. Pay attention to details, as sometimes minor clues can lead to significant breakthroughs.

Above all, maintain a mindset of continuous learning. The field of web security is vast and ever-changing. Embrace each challenge as an opportunity to expand your expertise and have fun unraveling the intricacies of web vulnerabilities!

PWN

A Guide to Solving PWN Challenges in Capture The Flag (CTF)

Welcome to the exciting realm of PWN (exploitation) challenges in Capture The Flag (CTF) competitions! PWN challenges test your ability to find and exploit vulnerabilities in binary executables, requiring a blend of programming knowledge, reverse engineering skills, and a deep understanding of system internals.

This guide aims to provide you with a framework to approach PWN challenges effectively, offering insights into common vulnerabilities and techniques used to exploit them.

Table of Contents

Understanding PWN Challenges

PWN challenges typically provide you with a binary executable (and sometimes the source code) running on a server. Your goal is to find vulnerabilities in the program and exploit them to gain unauthorized access, often to read a flag file.

These challenges test your ability to:

- Analyze binary executables.
- Understand and manipulate low-level code.
- Craft exploits to manipulate program execution.

General Approach

1. **Gather Information**:
   - Identify the binary type (e.g., ELF, PE) and architecture (e.g., x86, x86_64).
   - Determine the operating system and environment.
2. **Analyze Protections**:
   - Check what security mechanisms are in place (e.g., ASLR, NX, PIE, Stack Canaries).
3. **Static Analysis**:
   - Disassemble and decompile the binary to understand its functionality.
   - Look for dangerous functions (e.g., `gets`, `strcpy`).
4. **Dynamic Analysis**:
   - Run the binary locally to observe its behavior.
   - Use a debugger to step through execution.
5. **Identify Vulnerabilities**:
   - Look for input handling routines that may be exploitable.

- Test inputs to trigger unexpected behavior.

6. **Develop an Exploit**:
   - Craft payloads to exploit the identified vulnerability.
   - Bypass protections as necessary.
7. **Test Locally**:
   - Ensure your exploit works on your machine before attempting it on the remote server.
8. **Exploit Remotely**:
   - Use network tools to interact with the service running the binary.
   - Retrieve the flag or fulfill the challenge requirements.

Tools of the Trade

Equip yourself with essential tools for binary analysis and exploitation:

- **Disassemblers and Decompilers**:
  - **Ghidra**: Free, open-source reverse engineering suite.
  - **IDA Free**: Interactive disassembler for analyzing binaries.
- **Debuggers**:
  - **GDB**: The GNU Debugger for Linux.
  - **Pwntools GDB Extension**: Enhances GDB for exploit development.
  - **LLDB**: Debugger for Unix systems.
- **Exploitation Frameworks**:
  - **Pwntools**: CTF framework and exploit development library.
- **Binary Analysis Tools**:
  - **Binwalk**: Analyzes binary files for embedded files and executable code.
  - **Radare2**: Unix-like reverse engineering framework.
- **Utilities**:
  - **Checksec**: Checks binary protections.

    ```
    checksec --file=binary_name
    ```

  - **ROPgadget**: Helps in finding ROP gadgets.

Common Vulnerabilities and Exploitation Techniques

Buffer Overflows

Understanding Buffer Overflows

A buffer overflow occurs when a program writes more data to a buffer than it can hold, overwriting adjacent memory. This can overwrite function return addresses on the stack, allowing an attacker to control program execution.

Techniques and Tips

- **Identify Vulnerable Functions**:
  - Look for functions that don't check input sizes (e.g., `gets`, `strcpy`, `scanf` with `%s`).
- **Determine Buffer Size**:
  - Analyze the code or use pattern inputs to find how much data fills the buffer before overwriting the return address.
- **Create an Overflow Payload**:
  - Craft input that overflows the buffer and overwrites the return address with your desired address.
- **Control Execution Flow**:
  - Redirect execution to a function like `system("/bin/sh")` or to shellcode placed on the stack.
- **Account for Endianness**:
  - Remember that multi-byte values may need to be in little-endian format on x86 architectures.
- **Bypass Stack Protections**:
  - If stack execution is disabled (NX), use Return-Oriented Programming (ROP) techniques.

Format String Vulnerabilities

Understanding Format Strings

Format string vulnerabilities occur when user input is used as the format string in functions like `printf`, leading to unintended behavior, such as reading and writing arbitrary memory.

Techniques and Tips

- **Identify Vulnerable Usage**:
  - Look for `printf(user_input)` instead of `printf("%s", user_input)`.
- **Leak Memory**:
  - Use `%x` to read stack values, `%s` to read strings from memory.
- **Write to Memory**:
  - Use `%n` to write the number of bytes printed to an arbitrary location.
- **Calculate Offsets**:
  - Determine the position of your input on the stack to reference it in your format string.
- **Construct Payloads**:
  - Carefully craft the format string to read sensitive data or overwrite critical memory addresses.

Return-Oriented Programming (ROP)

Understanding ROP

ROP is an exploitation technique that chains together small sequences of instructions ending with a `ret` to perform arbitrary operations, bypassing protections like NX and ASLR.

Techniques and Tips

- **Find Gadgets**:
  - Use tools like ROPgadget to locate useful instruction sequences in the binary or linked libraries.
- **Build a ROP Chain**:
  - Chain gadgets together to perform complex tasks, like setting up registers and calling functions.
- **Control the Stack**:
  - Ensure that the stack is properly aligned and control the return addresses to execute your ROP chain.
- **Bypass ASLR**:
  - Leak memory addresses through vulnerabilities to calculate the base addresses of libraries.

Heap Exploitation

Understanding Heap Vulnerabilities

Heap-based vulnerabilities involve manipulating the memory allocator to achieve arbitrary code execution or data corruption. Common issues include use-after-free, double-free, and buffer overflows on the heap.

Techniques and Tips

- **Analyze Heap Operations**:
  - Understand how the program allocates, frees, and uses heap memory.
- **Exploit Allocation Patterns**:
  - Manipulate allocations to control the layout of heap memory.
- **Use House of Techniques**:
  - Apply known exploitation techniques like House of Einherjar, House of Storm, and others.
- **Abuse Metadata**:
  - Overwrite heap metadata to control the behavior of the allocator.
- **Leverage Use-After-Free**:
  - Free an object and then allocate it again to overwrite its contents.

Binary Protection Mechanisms

Understanding Protections

Modern binaries often include security features to prevent exploitation:

- **ASLR (Address Space Layout Randomization)**:
  - Randomizes memory addresses to prevent predictable exploitation.
- **NX (No-eXecute) Bit**:
  - Marks certain memory regions as non-executable.
- **Stack Canaries**:
  - Places a random value on the stack to detect buffer overflows.
- **PIE (Position Independent Executable)**:
  - Randomizes the base address of executable code.
- **RELRO (Relocation Read-Only)**:
  - Makes certain sections of the binary read-only to prevent GOT overwrites.

Bypassing Protections

- **Leak Addresses**:
  - Use information disclosure vulnerabilities to determine memory addresses.
- **Ret2libc Attacks**:
  - Redirect execution to libc functions, using known offsets.
- **Partial Overwrites**:
  - Overwrite parts of addresses to reduce the impact of ASLR.
- **Brute Force**:
  - In some cases, repeatedly attempt exploitation to guess random values (less practical).

Additional Tips and Resources

- **Practice Regularly**:
  - Exploit development requires hands-on experience. Work on practice challenges from resources like **pwnable.tw** and **pwnable.kr**.
- **Understand Assembly Language**:
  - Deep knowledge of assembly is crucial for analyzing binaries and crafting exploits.
- **Familiarize with C Programming and Compiler Behavior**:
  - Understanding how high-level code translates to assembly helps in identifying vulnerabilities.
- **Stay Updated on Exploitation Techniques**:
  - New methods emerge frequently. Follow blogs, conferences, and security research.

Helpful Links

- **Online Tutorials and Guides**:
  - **Heap Exploitation**: Guide to heap exploitation techniques.
  - **Pwn College**: Educational platform for learning binary exploitation.
- **Books**:
  - **Hacking: The Art of Exploitation** by Jon Erickson.
  - **The Shellcoder's Handbook** by Chris Anley et al.
- **Community Forums**:
  - **CTF Time**: Calendar of upcoming CTF events and write-ups.
  - **/r/ReverseEngineering**: Subreddit for reverse engineering discussions.
- **Tools Documentation**:
  - **Pwntools Documentation**: Comprehensive guide to using Pwntools.

Final Thoughts

PWN challenges are among the most technically demanding in CTFs, requiring a solid grasp of computer architecture, programming, and security concepts. They are also incredibly rewarding, offering deep insights into how systems work at a low level.

Remember to approach each challenge methodically:

- **Be Patient and Persistent**:
  - Complex vulnerabilities can take time to understand and exploit.

- **Break Down the Problem**:
  - Analyze the binary piece by piece; small insights can lead to breakthroughs.
- **Learn from Others**:
  - Reading write-ups from past CTFs can expose you to new techniques and thought processes.
- **Ethical Considerations**:
  - Use your skills responsibly. The techniques learned should be applied within legal and ethical boundaries, such as in controlled environments and with proper authorization.

Above all, keep learning and stay curious. The field of binary exploitation is vast and continuously evolving. Each challenge conquered enhances your skills and prepares you for the next.

Crypto

A Guide to Solving Crypto Challenges in Capture The Flag (CTF)

Welcome to the fascinating world of Cryptography challenges in Capture The Flag (CTF) competitions! Crypto challenges test your understanding of cryptographic concepts and your ability to apply them to break or analyze cryptographic systems. This guide is designed to help you navigate common cryptographic challenges involving RSA, AES, classic ciphers, zero-knowledge proofs, and pseudorandom number generators (PRNGs).

Table of Contents

Understanding Crypto Challenges

Crypto challenges require you to apply cryptographic knowledge to:

- Decrypt encrypted messages without the key.
- Exploit weaknesses in cryptographic implementations.
- Recover secret information or keys.
- Understand and manipulate cryptographic protocols.

These challenges test both theoretical knowledge and practical application. They often require creativity and a strong understanding of cryptographic principles.

General Approach

1. **Gather Information**:
    - Read the challenge description carefully.
    - Identify the type of cryptography used.
    - Collect any provided ciphertexts, plaintexts, keys, or hints.
2. **Understand the Cryptosystem**:
    - Determine which cryptographic algorithm is in use.
    - Analyze any provided code or scripts.
3. **Identify Weaknesses**:
    - Look for implementation flaws or misuse of cryptographic primitives.
    - Consider known attacks against the cryptosystem.
4. **Develop a Strategy**:
    - Decide on the appropriate attack based on your analysis.
    - Plan the steps needed to recover the plaintext or key.
5. **Implement the Attack**:
    - Use or write scripts and tools to perform the attack.

    - Verify your results at each step.
6. **Extract the Flag**:
    - Apply your findings to retrieve the flag.
    - Ensure that the decrypted message or recovered key leads to the solution.

Tools of the Trade

Equip yourself with essential cryptographic tools:

- **Programming Languages**:
    - **Python**: With libraries like `PyCrypto`, `Cryptography`, and `SymPy`.
    - **SageMath**: For advanced mathematical computations.
- **Cryptographic Tools**:
    - **RsaCtfTool**: Automates attacks against weak RSA keys.
    - **John the Ripper**: Password cracking tool.
    - **Hashcat**: Advanced password recovery.
    - **CyberChef**: Web-based tool for encoding and decoding.
- **Mathematical Software**:
    - **SageMath**: Open-source mathematics software system.
    - **Mathematica**: Computational software.
- **Online Resources**:
    - **dCode**: Collection of tools for decoding and cryptanalysis.
    - **Factordb**: Database of known integer factorizations.

Fundamental Concepts

Before diving into specific cryptosystems, ensure you have a solid understanding of:

- **Number Theory**:
    - Prime numbers, modular arithmetic, greatest common divisors (GCD), modular inverses.
- **Cryptographic Concepts**:
    - Encryption vs. hashing, symmetric vs. asymmetric encryption, block vs. stream ciphers.
- **Mathematical Algorithms**:
    - Extended Euclidean algorithm, Chinese remainder theorem, Fermat's little theorem.

RSA Encryption

Understanding RSA

RSA is an asymmetric cryptographic algorithm that relies on the difficulty of factoring large composite numbers. It uses a public key for encryption and a private key for decryption.

Key Components:

- **n**: Modulus, product of two large primes (p) and (q).
- **e**: Public exponent, usually a small value like 65537.
- **d**: Private exponent, satisfies (d \equiv e^{-1} \mod \phi(n)).
- **(\phi(n))**: Euler's totient function, ((p - 1)(q - 1)).

Techniques and Tips

- **Factoring Modulus (n)**:
  - If (n) is small, attempt to factor it directly using tools like `YAFU` or online services.
  - Check `Factordb` to see if (n) is a known composite.
- **Low Public Exponent Attacks**:
  - **Small (e)** with a small message (m) can be vulnerable to Wiener's attack or Hastad's attack.
- **Common Prime Factors**:
  - If multiple RSA moduli share a common prime, calculate GCDs between them to find shared factors.
- **Recovering Private Key (d)**:
  - Use Wiener's attack if (d) is small ((d < \frac{1}{3} n^{1/4})).
- **Faulty Key Generation**:
  - If (p) and (q) are too close, Fermat's factorization method can be effective.
- **Exploit Misused Padding Schemes**:
  - Absence of proper padding (e.g., PKCS#1) can make RSA vulnerable to attacks.
- **Use Mathematical Relationships**:
  - Apply equations and relationships involving (e), (d), (p), (q), and (n) to find unknowns.
- **CRT Optimization**:
  - If Chinese Remainder Theorem (CRT) parameters are available, reconstruct the private exponent.
- **Error Messages and Hints**:
  - Analyze any errors or hints provided in the challenge for clues.

Advanced Encryption Standard (AES)

Understanding AES

AES is a symmetric block cipher that operates on 128-bit blocks and supports key sizes of 128, 192, or 256 bits. It uses rounds of substitution and permutation based on key-derived values.

Techniques and Tips

- **Identify the Mode of Operation**:
  - Common modes include ECB, CBC, CFB, OFB, and CTR.
  - Each mode has different characteristics and vulnerabilities.
- **Electronic Codebook (ECB) Mode**:
  - Identical plaintext blocks result in identical ciphertext blocks.
  - Look for patterns in ciphertext to exploit.
- **Cipher Block Chaining (CBC) Mode**:
  - Exploit predictable Initialization Vectors (IVs) or reuse of IVs.
  - Perform padding oracle attacks if padding errors are revealed.
- **Padding Oracle Attacks**:
  - When an application behaves differently based on padding correctness, infer plaintext bytes.
- **Key-Reuse and IV Issues**:
  - Reusing keys or IVs improperly can lead to vulnerabilities.
- **Side-Channel Information**:
  - Time-based or error messages can leak information about the encryption process.
- **Brute Force Attempts**:
  - If the key space is small (e.g., passwords or short keys), attempt dictionary attacks.
- **Known Plaintext Attacks**:
  - If parts of the plaintext are known or predictable, leverage this to recover the key.
- **Analyze Code Implementations**:
  - Review any provided code for implementation flaws, such as weak random number generation or improper handling of keys and IVs.

Classic Ciphers

Understanding Classic Ciphers

Classic ciphers refer to historical encryption techniques, such as:

- **Caesar Cipher**: Shifts letters by a fixed number.
- **Vigenère Cipher**: Uses a keyword to shift letters.
- **Substitution Cipher**: Replaces letters based on a fixed system.
- **Transposition Cipher**: Rearranges the letters according to a system.
- **Playfair Cipher**, **Enigma Machine**, etc.

Techniques and Tips

- **Frequency Analysis**:
  - Analyze the frequency of letters or symbols to make educated guesses.
  - English plaintexts have characteristic frequency distributions.
- **Kasiski Examination (for Vigenère Cipher)**:
  - Identify repeated sequences to determine the key length.
- **Known Plaintext Attacks**:
  - Use portions of known plaintext to recover the key or decrypt messages.
- **Crib Dragging**:
  - Slide a known or guessed piece of plaintext along the ciphertext to find matches.
- **Using Online Tools**:
  - Utilize cipher-specific solvers available on sites like `dCode`.
- **Manual Decryption**:
  - For simple ciphers, try decrypting by hand to understand the encryption process.
- **Pattern Recognition**:
  - Look for repeating patterns, which may indicate cipher type or key length.
- **Polygraphic Substitution Ciphers**:
  - For ciphers like Playfair, consider that they encrypt pairs of letters.

Zero-Knowledge Proofs

Understanding Zero-Knowledge Proofs

Zero-knowledge proofs allow one party to prove knowledge of a secret without revealing it. In CTFs, challenges may involve:

- Understanding protocols like zk-SNARKs or zk-STARKs.
- Exploring commitment schemes.
- Analyzing simplified zero-knowledge systems.

Techniques and Tips

- **Protocol Analysis**:
  - Carefully read the protocol steps and understand how the proof works.
- **Look for Weaknesses**:
  - Identify if the challenge implementation deviates from the theoretical protocol.
- **Replay Attacks**:
  - Determine if previous messages can be reused to forge a proof.
- **Extracting Secrets**:
  - If the proof reveals more information than intended, use it to recover the secret.
- **Mathematical Exploits**:
  - Apply knowledge of discrete logarithms, quadratic residues, or other mathematical concepts as needed.
- **Understand Commitment Schemes**:
  - Analyze how the commitment is generated and whether it can be manipulated.
- **Side-Channel Attacks**:
  - Observe variations in timing or responses to infer secrets.

Pseudorandom Number Generators (PRNGs)

Understanding PRNGs

PRNGs generate sequences of numbers that approximate true randomness, but are actually deterministic. In cryptography, PRNGs must be secure, but they can be vulnerable if not properly implemented.

Techniques and Tips

- **Identify the PRNG Used**:

    - Common PRNGs include Linear Congruential Generators (LCGs), Mersenne Twister, etc.
- **Recovering the Seed**:
    - If the PRNG outputs are known, work backwards to find the seed.
- **Predicting Future Outputs**:
    - Use previous outputs to predict future ones.
- **Exploit Weaknesses in the PRNG**:
    - Simple PRNGs may have small periods or repeat cycles.
- **Brute Force the Seed**:
    - If the seed space is small (e.g., based on timestamps), attempt to brute force it.
- **Observe Usage Patterns**:
    - Analyze how the PRNG outputs are utilized in the challenge.
- **Use Mathematical Techniques**:
    - Apply algorithms to reverse engineer the PRNG's state, such as using lattice reduction methods for LCGs.
- **Examine Implementation Details**:
    - Review any provided code for non-cryptographic PRNGs being used in cryptographic contexts.

Additional Tips and Resources

- **Strengthen Mathematical Foundations**:
    - Study number theory, algebra, and probability.
- **Learn Cryptographic Algorithms**:
    - Understand how common algorithms work and their typical vulnerabilities.
- **Practice with Challenges**:
    - Use platforms like **Cryptopals** for structured practice.
- **Read Academic Papers and Write-ups**:
    - Learn from others' experiences and documented attacks.
- **Stay Updated on Security News**:
    - Cryptography is an evolving field; keep abreast of new vulnerabilities.

Helpful Links

- **Online Cryptography Courses**:
    - **Coursera - Cryptography I** by Stanford University.
- **Books**:
    - **"Serious Cryptography"** by Jean-Philippe Aumasson.
    - **"Applied Cryptography"** by Bruce Schneier.
- **Communities**:

- **Crypto Stack Exchange**: Q&A platform for cryptography.

Final Thoughts

Cryptography challenges blend mathematical rigor with creative problem-solving. They require both theoretical knowledge and practical skills in applying that knowledge to unconventional problems.

Remember to:

- **Approach Methodically**:
  - Break down the problem into manageable parts.
- **Think Critically**:
  - Question assumptions and consider alternative perspectives.
- **Experiment and Iterate**:
  - Test hypotheses and learn from failed attempts.
- **Collaborate and Learn**:
  - Engage with the community to broaden your understanding.

Above all, enjoy the process of unraveling the secrets hidden within cryptographic challenges. Each challenge conquered enhances your mastery and prepares you for future puzzles.

Reverse Engineering

Welcome to the crying world of Reverse Engineering (RE) in Capture The Flag (CTF) challenges! Reverse engineering requires you to analyze software to understand its inner workings, often without access to the source code. This guide is designed to help you navigate RE challenges involving binaries compiled from various programming languages like C, C++, Python, and Android APKs.

Table of Contents

Understanding Reverse Engineering Challenges

In RE challenges, you're typically provided with a compiled program (binary) and tasked with understanding its functionality to:

- Extract hidden information or flags
- Bypass certain checks or protections
- Modify behavior to achieve a desired outcome
- Discover vulnerabilities

These binaries can be compiled from various programming languages, and each presents unique challenges and requires specific tools and approaches.

General Approach

1. **Identify the Type of Binary**:
   - Determine the target platform (e.g., Windows, Linux, Android).
   - Identify the programming language or environment if possible.
2. **Set Up Your Environment**:
   - Use a virtual machine or sandbox to analyze potentially malicious binaries.
   - Install required tools and dependencies.
3. **Perform Static Analysis**:
   - Examine the binary without executing it.
   - Use disassemblers or decompilers to understand code structure.
4. **Perform Dynamic Analysis**:
   - Run the binary in a controlled environment.
   - Use debuggers to observe runtime behavior.
5. **Document Your Findings**:
   - Keep detailed notes on functions, variables, and control flow.
   - Map out the program logic.
6. **Extract the Flag**:
   - Apply your understanding to retrieve the flag or meet the challenge objectives.

Tools of the Trade

Before diving into specific types of binaries, familiarize yourself with essential reverse engineering tools:

- **Disassemblers**:
  - Ghidra: Free, open-source suite for software reverse engineering.
  - IDA Free: Free version of the Interactive Disassembler.

- **Debuggers**:
  - GDB: The GNU Debugger for Linux binaries.
  - x64dbg: Open-source debugger for Windows applications.
- **Hex Editors**:
  - HxD: Fast hex editor for Windows.
  - HxE: Cross-platform hex editor.
- **Binary Analysis Tools**:
  - Radare2: Advanced command-line framework for binary analysis.
  - Binary Ninja: User-friendly reverse engineering platform (paid).
- **Decompilers**:
  - Integrated into Ghidra and IDA Pro for high-level code reconstruction.

Analyzing Native Binaries (C/C++)

Getting Started with C/C++ Binaries

Native binaries compiled from C or C++ are common in RE challenges. These binaries may have been compiled with optimization or obfuscation, making analysis more challenging.

Initial Steps:

- **Determine the File Type**:
  - Use the `file` command in Linux to identify the binary format.

    ```
    file binary_name
    ```

- **Check for Symbols**:
  - Symbols can aid analysis. If symbols are stripped, variable and function names will be missing.
- **Scan for Protections**:
  - Use `checksec` to identify security mechanisms like NX, ASLR, PIE, Canary.

    ```
    checksec --file=binary_name
    ```

Techniques and Tips

- **Disassembly and Decompilation**:
  - Load the binary into Ghidra or IDA to view assembly code and decompiled C code.

    - Rename functions, variables, and label code blocks to reflect their purpose.
- **Understand the Entry Point**:
    - Identify `main` or the starting function.
    - Trace function calls and data flow.
- **Identify Key Functions**:
    - Look for functions related to input handling, encryption/decryption, and validation.
- **String Analysis**:
    - Use the `strings` command to find ASCII and Unicode strings.

      ```
      strings binary_name
      ```

    - Examine strings in the disassembler for hardcoded messages or data.
- **Control Flow Analysis**:
    - Map out loops, conditionals, and branching to understand program logic.
- **Dynamic Analysis with a Debugger**:
    - Set breakpoints at critical functions.
    - Step through the execution to observe behavior.
- **Modify Execution Flow**:
    - If the binary performs checks (e.g., password verification), consider patching the binary to bypass them.
    - Use a hex editor or built-in patching features in Ghidra/IDA.
- **Dealing with Obfuscation**:
    - Simplify complex expressions.
    - Inline function calls if functions are small and called frequently.

Python Bytecode Disassembly

Getting Started with Python Binaries

Python is an interpreted language, but compiled Python files (`.pyc`) contain bytecode that can be reverse-engineered.

Initial Steps:

- **Identify Python Bytecode Files**:
    - Look for `.pyc` files or compiled packages.
- **Check Python Version**:
    - The magic number in the `.pyc` file header indicates the Python version used.

Tools for Python Reverse Engineering

- **Uncompyle6**:
  - Decompiles Python 2.x and 3.x bytecode back to readable Python source code.

```
uncompyle6 -o output_directory compiled_file.pyc
```

- **Decompyle++**:
  - Decompiler for Python 3 bytecode.
- **PyInstaller Extractor**:
  - Extracts Python files from executables packaged with PyInstaller.

Tips for Python Challenges

- **Decompile Bytecode**:
  - Use `uncompyle6` to get the original source code.
- **Analyze the Source**:
  - Read the decompiled code to understand program logic.
- **Handle Obfuscated Code**:
  - If variable names are obfuscated, rename them for clarity.
  - Inline functions or decrypt strings if necessary.
- **Dynamic Analysis**:
  - Run the script in a controlled environment.
  - Use a debugger like `pdb` to step through execution.
- **Inspect Constants**:
  - Look at constant values in the bytecode which may contain encoded information.

Reverse Engineering APKs (Android Applications)

Getting Started with APKs

APKs are package files for Android applications, which can be reverse-engineered to analyze their contents.

Initial Steps:

- **Unpack the APK**:
  - APKs are zipped archives. Use `unzip` to extract their contents.

```
unzip app_name.apk -d output_directory
```

- **Identify the Structure**:
  - Important directories: `smali`, `lib`, `res`, `assets`, `META-INF`.
  - Key files: `AndroidManifest.xml`, `classes.dex`.

Tools for APK Analysis

- **Apktool**:
  - Decompile and recompile APKs, decode resources, and view manifest files.

```
apktool d app_name.apk
```

- **JD-GUI**:
  - Java Decompiler GUI; view decompiled Java source from `.class` files.
- **JADX**:
  - Decompile `.dex` files to Java source code.
- **Bytecode Viewer**:
  - Integrated tool for analysis of `.class`, `.jar`, and `.apk` files.
- **smali/baksmali**:
  - Disassemble and assemble `.dex` files to and from Smali assembly.

Tips for APK Challenges

- **Analyze the Manifest**:
  - Review `AndroidManifest.xml` for app permissions and components.
- **Decompile to Java**:
  - Use JADX or JD-GUI to convert `.dex` files to Java source code.
  - Read and understand the decompiled code.
- **Examine Native Libraries**:
  - Check the `lib` directory for native binaries (`.so` files).
  - Apply techniques from analyzing native binaries if present.
- **Look for Hardcoded Data**:
  - Search the code for hardcoded credentials, URLs, or flags.
- **Handle Obfuscation**:
  - If code is obfuscated, use deobfuscation tools or manually rename classes and methods.
- **Dynamic Analysis**:
  - Run the app in an emulator like Android Studio Emulator or Genymotion.
  - Use tools like Frida for dynamic instrumentation.
- **Inspect Resources**:

    - Check `assets` and `res` directories for images, configurations, and other files.
- **Network Traffic Analysis**:
    - Use mitmproxy or Burp Suite to intercept and analyze network communications.

Additional Tips and Resources

- **Stay Organized**:
    - Keep track of your progress, notes, and modifications.
    - Use version control for tracking changes in decompiled code.
- **Understanding Compiler Optimizations**:
    - Compilers may optimize code heavily; recognize common patterns.
    - Inlined functions, loop unrolling, and other optimizations can obfuscate code.
- **Learn Assembly Language**:
    - Familiarity with x86/x64 and ARM assembly languages is crucial.
    - Understand calling conventions, registers, and instruction sets.
- **Learn Scripting for Automation**:
    - Use Python or other scripting languages to automate analysis tasks.
    - Leverage APIs provided by tools like Ghidra and IDA.
- **Practice Regularly**:
    - Reverse engineering is a skill honed by practice.
    - Work on challenges from past CTFs and platforms.

Helpful Links

- **Reverse Engineering Tutorials**:
    - Beginner's Guide to Reverse Engineering Android Apps
- **CTF Practice Platforms**:
    - Crackmes.one: Collection of reverse engineering challenges.
    - Reversing.Kr: RE challenges with varying difficulties.
- **Community Forums**:
    - Reverse Engineering Stack Exchange
    - /r/ReverseEngineering Subreddit
- **Books**:
    - **Practical Reverse Engineering** by Bruce Dang et al.
    - **Reversing: Secrets of Reverse Engineering** by Eldad Eilam.

Final Thoughts

Reverse engineering challenges are both intellectually stimulating and rewarding. They require a deep understanding of programming concepts, assembly language, and system internals.

Remember, patience and persistence are key. Don't be discouraged by complexity; breaking down the problem into smaller, manageable parts is an effective strategy.

Most importantly, have fun exploring and unraveling the mysteries within the binaries!

Forensics

Welcome to the world of Forensics in Capture The Flag (CTF) challenges! Forensics challenges are an integral part of CTF competitions, requiring keen analytical skills and attention to detail. This guide aims to equip you with the knowledge and tools necessary to tackle forensics challenges involving network captures, memory dumps, and disk images.

Table of Contents

Understanding Forensics Challenges

Forensics challenges simulate real-world scenarios where you analyze digital artifacts to uncover hidden information or understand an incident. These artifacts can include:

- **Network Captures (PCAPs)**: Files containing recorded network traffic.
- **Memory Dumps**: Snapshots of a system's RAM at a given time.
- **Disk Images**: Complete copies of storage media, including all files and file system structures.

Success in forensics challenges requires a methodical approach, familiarity with various tools, and an eye for detail.

General Approach

1. **Identify the Type of Artifact**: Determine whether you're dealing with a PCAP, memory dump, or disk image.
2. **Understand the Challenge Context**: Read the challenge description carefully for clues.
3. **Prepare Your Tools**: Ensure you have the necessary software installed and configured.
4. **Formulate a Hypothesis**: Based on initial observations, decide what you're looking for.
5. **Analyze Systematically**: Follow a structured methodology to examine the artifact.
6. **Document Your Findings**: Keep detailed notes of your analysis steps and discoveries.
7. **Extract the Flag**: The ultimate goal is to find the flag, which may be hidden or encoded.

Analyzing Network Captures (PCAPs)

Getting Started with PCAPs

Network captures record data packets transmitted over a network. Analyzing PCAP files can reveal:

- User credentials
- Transferred files
- Malicious activities
- Hidden messages

Tools for PCAP Analysis

- **Wireshark**: The most popular network protocol analyzer.
- **tcpdump**: Command-line packet analyzer.
- **Tshark**: Command-line version of Wireshark.
- **NetworkMiner**: Extract files and data from PCAPs.
- **Brim**: Advanced PCAP analysis tool with powerful query capabilities.

Tips for PCAP Challenges

- **Start with Protocol Analysis**:
  - Identify the protocols used (HTTP, FTP, DNS, etc.).
  - Use Wireshark's protocol hierarchy to see the breakdown.
- **Follow Streams**:
  - Use "Follow TCP/UDP Stream" to reconstruct conversations.
  - Check both directions of communication.
- **Search for Keywords**:
  - Look for common flag formats (e.g., `CTF{}`, `FLAG{}`).
  - Use Wireshark's search feature with regex patterns.
- **Extract Files**:
  - Use Wireshark's "Export Objects" feature to extract transferred files.
  - Analyze extracted files for hidden data.
- **Apply Filters**:
  - Use display filters to focus on relevant traffic.
    - Examples: `http`, `ftp`, `dns`, `smtp`, `tcp.port == 80`
- **Check for Unusual Activities**:
  - Look for malformed packets or anomalies.
  - Analyze any encrypted or obfuscated traffic.

Memory Dump Analysis

Getting Started with Memory Dumps

Memory dumps capture the contents of system RAM, which may contain:

- Running processes and their data
- Passwords and cryptographic keys
- Hidden or malicious programs
- Fragments of files and registry hives

Tools for Memory Analysis

- **Volatility Framework**: An advanced memory forensics framework.
- **Rekall**: Memory analysis tool similar to Volatility.
- **DumpIt**: Acquire memory dumps from Windows systems.
- **LiME**: Linux Memory Extractor for acquiring memory.

Tips for Memory Challenges

- **Identify the OS Profile**:
  - Use the `imageinfo` command in Volatility to determine the operating system profile.
- **Enumerate Processes**:
  - List running processes with `pslist` or `pstree`.
  - Look for suspicious or unfamiliar processes.
- **Analyze Network Connections**:
  - Use `netscan` to find open connections and ports.
- **Dump Process Memory**:
  - Extract process memory with `procdump` for further analysis.
- **Search for Strings**:
  - Use `strings` command-line tool or Volatility's `strings` plugin.
  - Look for plaintext passwords, URLs, or flags.
- **Registry Analysis**:
  - Use `hivelist` and `printkey` to examine registry hives.
- **Check for Malware**:
  - Analyze potential malicious code with `malfind`.

Disk and File System Forensics (Dead Box)

Getting Started with Disk Images

Disk images are exact copies of storage media, allowing you to:

- Recover deleted files
- Analyze file system structures
- Examine installed applications

- Discover hidden data

Tools for Disk Forensics

- **Autopsy**: Graphical interface for The Sleuth Kit.
- **The Sleuth Kit (TSK)**: Command-line tools for disk analysis.
- **FTK Imager**: Disk imaging and data preview tool.
- **Bulk Extractor**: Scans disk images to extract useful information.
- **ExifTool**: Reads and writes metadata in files.

Tips for Disk Forensics Challenges

- **Mount the Disk Image**:
  - Use `mount` (Linux) or tools like OSFMount (Windows) to mount the image as a file system.
- **File Carving**:
  - Recover deleted or hidden files using tools like `photorec` or `foremost`.
- **Search for Hidden Data**:
  - Look for hidden partitions or alternate data streams (ADS).
  - Examine unallocated space for residual data.
- **Analyze File System Metadata**:
  - Check timestamps, file permissions, and ownership.
- **Examine User Data**:
  - Browse user directories for documents, images, and notes.
  - Don't forget to check the Recycle Bin/Trash.
- **Look for Steganography**:
  - Analyze media files for embedded information.
  - Use tools like **StegSolve** or **StegoSuite**.
- **Check for Encrypted Files**:
  - Identify encrypted archives or containers.
  - Attempt to crack passwords if permissible.

Additional Tips and Resources

- **Stay Organized**:
  - Keep your workspace and files well-organized.
  - Use consistent naming conventions.
- **Automate Repetitive Tasks**:

    - Write scripts to automate parts of your analysis.
    - Use batch processing where applicable.
- **Collaborate and Communicate**:
    - Discuss findings with teammates.
    - Share insights and methodologies.
- **Keep Learning**:
    - Follow blogs, forums, and write-ups.
    - Practice with sample challenges and past CTFs.

Helpful Links

- **Forensics Tutorials**:
    - SANS Digital Forensics and Incident Response Blog
    - DFIR Training
- **CTF Practice Platforms**:
    - Cyber Defenders: Forensics labs and challenges.
    - Open Cyber Challenge Platform: Open-source CTF platform.
- **Cheat Sheets**:
    - SANS Forensics Cheat Sheets
    - Wireshark Display Filter Reference

Final Thoughts

Forensics challenges offer a unique opportunity to delve into the intricacies of digital artifacts and develop a deep understanding of investigative techniques. They require patience, attention to detail, and a systematic approach.

Remember, practice is key. The more challenges you tackle, the more proficient you'll become. Don't hesitate to reach out to the community, participate in discussions, and share your experiences.

Good luck on your forensics journey!

Feel free to join us in our next Hack N' Chill session, where we collaborate on challenges and learn together!

Misc

Hardware

If you are reading this with expectation of getting help for the challenges. God help you