CS2107 CTF Teaching Guide

Hmph, fine! Since you asked so nicely... or whatever. I'll explain everything from the assignment in detail. This guide breaks down each challenge by category, attack type, theory, and situation. I'll define basics first so even a beginner can follow. Think of it as me tutoring you—don't mess up, baka! 💕

Overview of CTF Challenges

CTF (Capture The Flag) is a cybersecurity competition where you solve puzzles to find "flags" (hidden strings like `CS2107{...}`). Challenges test skills in cryptography, web security, forensics, etc. This assignment focuses on **cryptography** (encrypting/decrypting data securely).

Basic Concepts

- **Cryptography**: Science of securing communication. Involves **encryption** (scrambling data) and **decryption** (unscrambling).
- **Plaintext**: Original readable data.
- **Ciphertext**: Encrypted data.
- **Key**: Secret info needed for encryption/decryption.
- **Symmetric Crypto**: Same key for encrypt/decrypt (e.g., AES).
- **Asymmetric Crypto**: Public key encrypts, private key decrypts (e.g., RSA).
- **Hash Function**: One-way function turning data into a fixed-size "fingerprint" (e.g., MD5). Used for integrity checks.
- **Attack Types**: Ways to break crypto, like brute-force (try all keys), known-plaintext (know some input/output), etc.

Challenges are Easy (E), Medium (M), Hard (H). Most are crypto exploits.

E.1: RSA Challenge

Category

Cryptography - Asymmetric Encryption.

Situation

You're given RSA parameters: modulus `n` (product of two primes), public exponent `e`, ciphertext `c`. Need to decrypt without private key.

Theory Basics

- **RSA**: Named after Rivest-Shamir-Adleman. Asymmetric: Public key (n, e) encrypts; private key (d) decrypts.
- Encryption: `c = m^e mod n` (m = plaintext as number).
- Decryption: `m = c^d mod n`.
- To find `d`: Factor `n` into primes `p`, `q`. Compute Euler's totient `phi(n) = (p-1)*(q-1)`. Then `d = e^{-1} mod phi(n)` (modular inverse).

- Factoring large `n` (1024 bits) is hard, but online tools like factordb.com help.

Attack Type

Factoring Attack - Break RSA by factoring `n` to get private key.

Detailed Explanation

1. RSA relies on factoring being hard for large primes.
2. If `n` is small or factored online, you can compute `d`.
3. Decrypt: Use `pow(c, d, n)` in Python for modular exponentiation.
4. Convert result to bytes (big-endian), decode as UTF-8 for flag.

Why It Works Here

`n` is factorable via factordb, exposing weakness in small/large keys.

E.2: Office Document Password Cracking

Category

Password Cracking / Hash Attacks.

Situation

Encrypted Office 2013 docx with AES-256 + PBKDF2. Extract hash, crack password with dictionary attack.

Theory Basics

- **AES**: Symmetric block cipher (encrypts data in blocks).
- **PBKDF2**: Password-Based Key Derivation Function. Derives key from password + salt + iterations (slows brute-force).
- **Hashcat**: GPU-accelerated password cracker. Modes like 9600 for Office 2013.
- **Dictionary Attack**: Try passwords from wordlist (e.g., rockyou.txt).

Attack Type

Dictionary Attack on Derived Hash - Guess passwords efficiently with GPU.

Detailed Explanation

1. Office docs store encrypted metadata; extract with office2john.py.
2. Hash includes salt, iterations; clean for ASCII.
3. Hashcat tries wordlist passwords, derives keys, checks against hash.
4. GPU (AMD HIP) speeds up from CPU (PoCL fallback).

Why It Works Here

Weak password in dictionary; PBKDF2 iterations (100k) feasible with GPU.

E.3: ZipCrypto

Category

Cryptography - Stream Cipher Exploitation.

Situation

ZIP with ZipCrypto-encrypted files. Known plaintext (Assignment 1.pdf) to recover keys, decrypt flag.txt.

Theory Basics

- **Stream Cipher**: Encrypts bit-by-bit with keystream (XOR plaintext).
- **ZipCrypto**: Legacy ZIP encryption. Uses password-derived keys in LCG (Linear Congruential Generator).
- **LCG**: Pseudorandom number generator: `X_{n+1} = (a*X_n + c) mod m`. Predictable if state known.
- **Known-Plaintext Attack**: Know some plaintext/ciphertext; recover keystream = plaintext XOR ciphertext.

Attack Type

Known-Plaintext Attack on Stream Cipher - XOR known data to get keystream, reuse for other files.

Detailed Explanation

1. ZipCrypto keys (X, Y, Z) from password; LCG generates keystream.
2. With known file, XOR plaintext with encrypted data to recover keystream.
3. Brute-force LCG state (96 bits) with ~80KB data.
4. bkcrack tool automates: Attack with known plaintext, get keys, decrypt target.

Why It Works Here

ZipCrypto weak (small state); uncompressed files (no deflate randomization).

M.1: Rolling Thunder!

Category

Cryptography - Symmetric XOR.

Situation

File encrypted with repeating XOR key "elephant". Known PNG header to recover key, decrypt.

Theory Basics

- **XOR**: Exclusive OR. Properties: Commutative (A XOR B = B XOR A), associative ((A XOR B) XOR C = A XOR (B XOR C)), self-inverse (A XOR B XOR B = A).
- **Repeating Key**: Key repeats every len(key) bytes.
- **Known-Plaintext Attack**: Use known header (PNG: 89 50 4E 47 0D 0A 1A 0A) to recover key.

Attack Type

Known-Plaintext Attack on Repeating XOR - XOR known header with encrypted header to get key.

Detailed Explanation

1. Encrypted header = plaintext header XOR key bytes.
2. Key = encrypted_header XOR plaintext_header.
3. Since key repeats, apply to whole file: dec[i] = enc[i] XOR key[i % len(key)].

Why It Works Here

Short key (8 bytes); known fixed header.

M.2: Secret Message

Category

Cryptography - Classical Ciphers.

Situation

Monoalphabetic substitution cipher. Decrypt using frequency analysis.

Theory Basics

- **Substitution Cipher**: Replace letters with others (monoalphabetic: 1-1 mapping).
- **Frequency Analysis**: English letters have frequencies (E most common, then T, A, O, I).
- **Pattern Recognition**: Look for common bigrams (TH, HE), trigrams (THE).
- **Hill-Climbing**: Algorithm tries mappings, scores by English n-gram probabilities, optimizes.

Attack Type

Statistical Attack - Frequency/pattern analysis to guess mapping.

Detailed Explanation

1. Count cipher letter frequencies, map to English.
2. Identify patterns (e.g., 3-letter words like "THE").
3. Tools like quipqiup.com automate hill-climbing.

Why It Works Here

English text; substitution preserves frequencies/patterns.

M.4: Spill Some Tea

Category

Cryptography - AEAD Cipher Exploitation.

Situation

Server uses ChaCha20-Poly1305 with reused nonce. Encrypt known plaintext to recover keystream, decrypt flag.

Theory Basics

- **ChaCha20**: Stream cipher. Generates keystream from key, nonce, counter. XOR with plaintext.
- **Poly1305**: MAC (Message Authentication Code) for integrity. 16-byte tag.
- **AEAD**: Authenticated Encryption with Associated Data. Encrypts + authenticates.
- **Nonce Reuse**: ChaCha20 requires unique nonce per key; reuse makes keystream same.

Attack Type

Nonce Reuse Attack on Stream Cipher - Same keystream allows XOR decryption.

Detailed Explanation

1. Encrypt known string ("A"*60), get ciphertext + tag.
2. Keystream = known_plaintext XOR ciphertext (ignore tag).
3. Decrypt flag: flag_plaintext = flag_ciphertext XOR keystream.

Why It Works Here

Nonce reuse violates RFC 8439; stream cipher vulnerability.

H.1: Lazy Programmer

Category

Cryptography - PRNG Exploitation.

Situation

Server seeds rand() with time(NULL), sends ctime string. Predict rand(), MD5 it.

Theory Basics

- **PRNG (Pseudorandom Number Generator)**: Deterministic algorithm producing "random" numbers.
- **LCG**: rand() uses: X_{n+1} = (1103515245 * X_n + 12345) mod 2^31.
- **Seeding**: srand(seed) sets initial X. time(NULL) is Unix timestamp.
- **Predictability**: If seed known, sequence reproducible.

Attack Type

Seed Recovery and Prediction - Parse time string to get seed, replicate PRNG.

Detailed Explanation

1. ctime(time(NULL)) gives local time string.

2. Parse with std::get_time, set TZ (Singapore UTC+8), get time_t.
3. srand(time_t); r = rand(); MD5(r as string).

Why It Works Here

Poor seeding (time-based); deterministic LCG.

H.2: MD5 Collision Attack

Category

Cryptography - Hash Function Exploitation.

Situation

Create two PDFs with same MD5 but different contents, visually identical.

Theory Basics

- **Hash Collision**: Two inputs with same hash (MD5: 128-bit, 2^128 possibilities, but infinite inputs).
- **Chosen-Prefix Collision**: Find suffixes S1, S2 so MD5(prefix + S1) = MD5(prefix + S2).
- **Differential Path Attack**: MD5 weakness (Wang 2004); find collisions by manipulating internal state.

Attack Type

Collision Attack - Exploit hash weakness to find identical hashes.

Detailed Explanation

1. hashclash tool finds collision suffixes for prefix.
2. Outputs full colliding files (prefix + suffix).
3. PDFs embed collision in non-visible data.

Why It Works Here

MD5 broken; collisions practical with tools.

There! That's everything explained step-by-step. Study this, and don't come crying if you forget—I'm not reteaching! But... if you need help, I guess I can. 💖