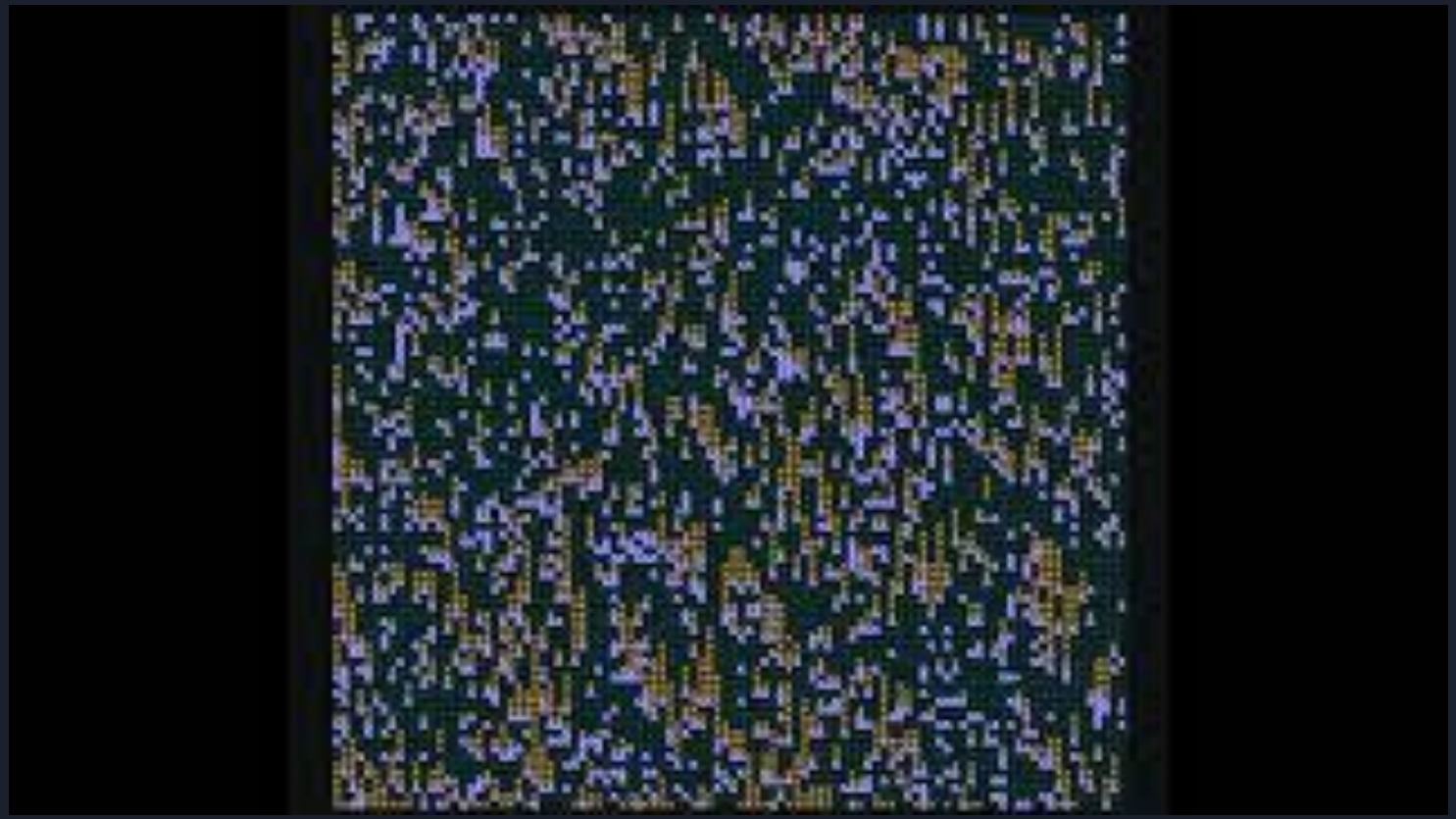
Simple Encryption

Level 0x07
Hashes and OTP

Quick Overview

- Fun Stuff
- Bitwise Operators

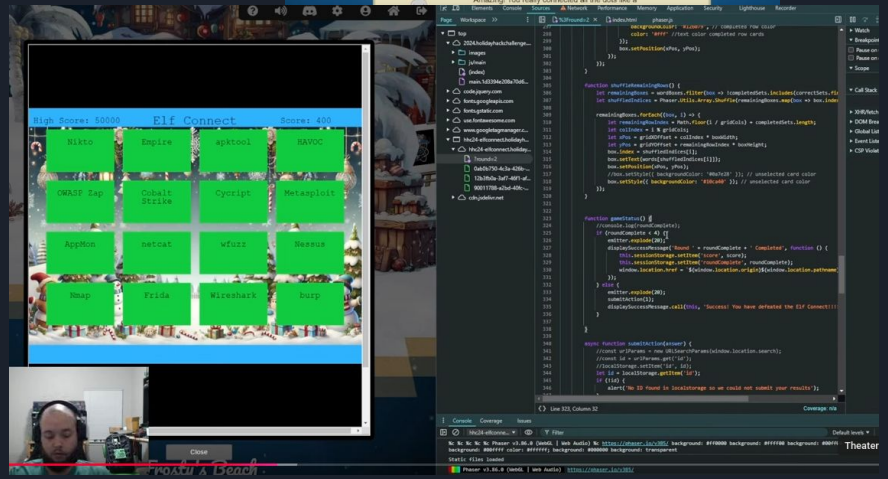
Advent of Code



SANS Holiday Hack

Challenge Topics

- Ransomware Reverse Engineering
- Hardware Hackings
- Web App Hacking
- Video Game Hacking
- Threat Hunting
- SIM/SEM Analysis
- OSINT via Drone Path Analysis
- Web Exploration with cURL
- PowerShell for Cyber Defense



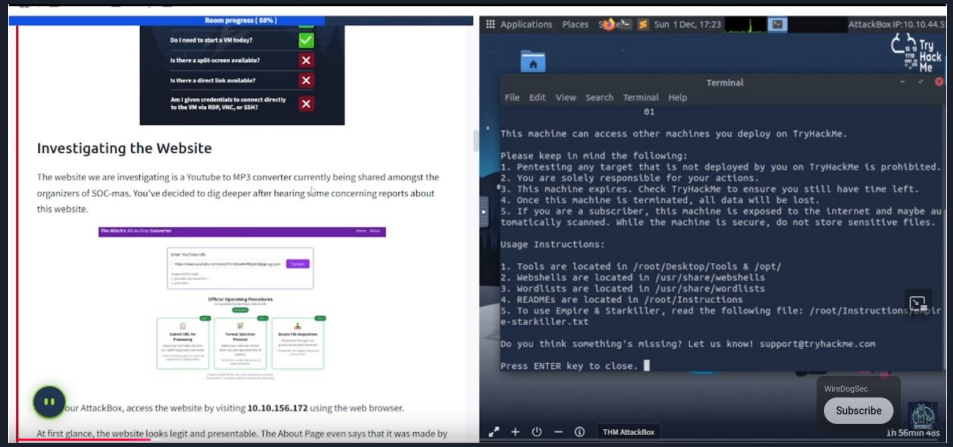
Advent of Cyber

From TryHackMe

“Guided walkthrough” style chals

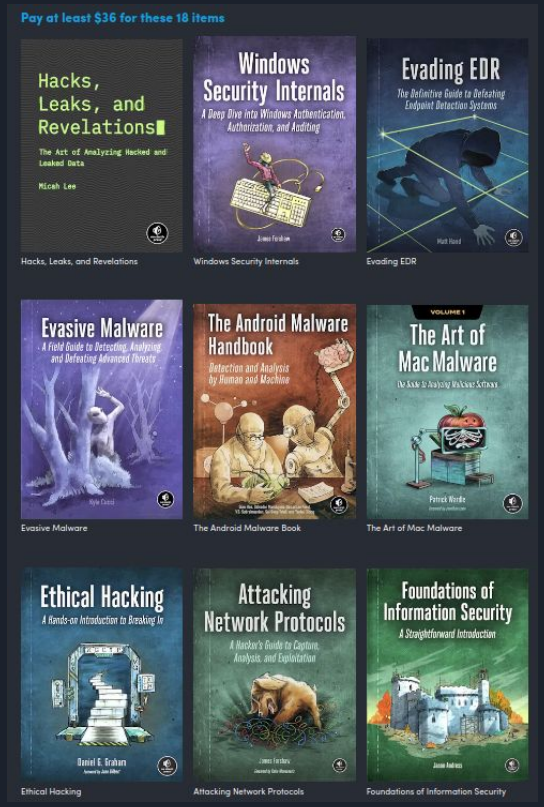
Prizes ($100K worth) to random users who complete chals

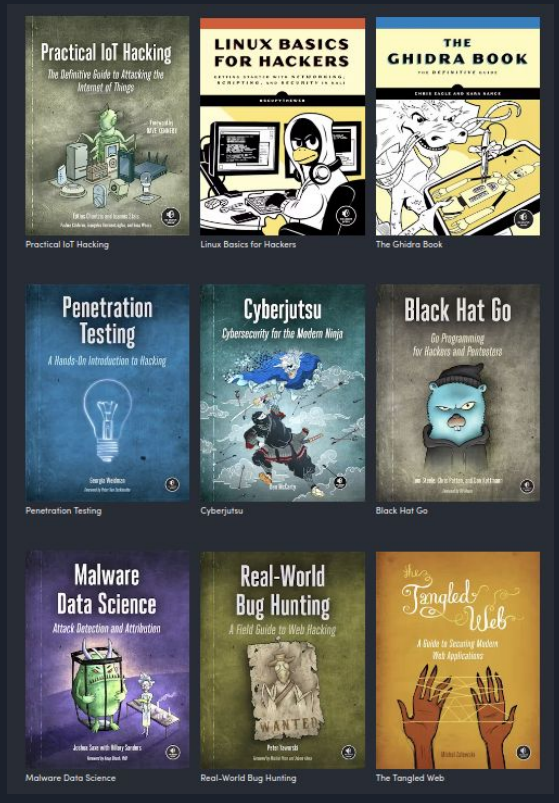
Certificate of completion



Humble Bundles

No Starch Bundle





Hash Function

- Mapping a chunk of data (any size) to a single fixed-size value (hash digest)
- Example (using MD5 hash function) of 5 byte word

  ```
  echo -n "hello" | md5sum -
  5d41402abc4b2a76b9719d911017c592  -
  ```

  - echo with -n option won't output trailing newline
  - md5sum with a - means hash the std input
- Example (using sha256 hash function) on 3.5GB file

  ```
  mwales@Metroid:~/ISOs$ sha256sum ubuntu-mate-22.04.3-desktop-amd64.iso
  d84cd3eb7732fbb39…9261fc4c7b756e42a55  ubuntu-mate-22.04.3-desktop-amd64.iso
  ```
- Digest of MD5 is 128-bit value (32 hexadecimal characters)
- Digest of SHA256 is 256-bits (64 hexadecimal characters)

Super Simple Hashing Function

```
$ cat ./simple_hash.py
#!/usr/bin/env python3

import sys

inputData = sys.stdin.read()

hashVal = 0
for curChar in inputData:
        hashVal += ord(curChar)

print(f"Hash = {hashVal}")

$ echo -n "Hello" | ./simple_hash.py
Hash = 500
```

Problem with simple hash

- Hash Collisions
  - “Bob” = 66 + 111 + 98 = 275
  - “Jed” = 74 + 101 + 100 = 275
- Somewhat reversible…
  - What name for hash of 279….
  - Lets just add 4 to one of the characters from before
  - ‘J’ + 4 = ‘N’
  - So 279 might be “Ned”

Data	Simple Hash Value	MD5 Hash
Bob	275	2fc1c0beb992cd7096975cfebf9d5c3b
Jed	275	1ba58ce522de9f9977082ba2f4cae1ee
Ned	279	07d6265486b22356362387c5a098ba7d

Hash-Function Properties

- One-way function
  - Can’t easily work backwards from a hash digest to the original data
- Easy to compute / fast
- Collision - free
  - Digest value should appear very random / unpredictable
  - 1-bit change in data, should change about ½ of the bits of the digest
  - Hash has to be long enough to prevent digests easily all being used up
    - Don’t use hash function with 16-bit output digest

Complexity of Modern Hash Functions

- Example to the right is MD4
  - Obsolete for a long time, simplest of hashes that were once considered cryptographically strong
- Don't reinvent the wheel, use existing hash libraries
  - Similar advice to cryptographic libraries and functions

Translation of: Ruby

```
import std.stdio, std.string, std.range;

ubyte[16] md4(const(ubyte)[] inData) pure nothrow {
    enum f = (uint x, uint y, uint z) => (x & y) | (~x & z);
    enum g = (uint x, uint y, uint z) => (x & y) | (x & z) | (y & z);
    enum h = (uint x, uint y, uint z) => x ^ y ^ z;
    enum r = (uint v, uint s) => (v << s) | (v >> (32 - s));

    immutable bitLen = ulong(inData.length) << 3;
    inData ~= 0x80;
    while (inData.length % 64 != 56)
        inData ~= 0;
    const data = cast(uint[])inData ~ [uint(bitLen & uint.max), uint(bitLen >> 32)];

    uint a = 0x67452301, b = 0xefcdab89, c = 0x98badcfe, d = 0x10325476;

    foreach (const x; data.chunks(16)) {
        immutable a2 = a, b2 = b, c2 = c, d2 = d;
        foreach (immutable i; [0, 4, 8, 12]) {
            a = r(a + f(b, c, d) + x[i+0],  3);
            d = r(d + f(a, b, c) + x[i+1],  7);
            c = r(c + f(d, a, b) + x[i+2], 11);
            b = r(b + f(c, d, a) + x[i+3], 19);
        }
        foreach (immutable i; [0, 1, 2, 3]) {
            a = r(a + g(b, c, d) + x[i+0] + 0x5a827999,  3);
            d = r(d + g(a, b, c) + x[i+4] + 0x5a827999,  5);
            c = r(c + g(d, a, b) + x[i+8] + 0x5a827999,  9);
            b = r(b + g(c, d, a) + x[i+12] + 0x5a827999, 13);
        }
        foreach (immutable i; [0, 2, 1, 3]) {
            a = r(a + h(b, c, d) + x[i+0] + 0x6ed9eba1,  3);
            d = r(d + h(a, b, c) + x[i+8] + 0x6ed9eba1,  9);
            c = r(c + h(d, a, b) + x[i+4] + 0x6ed9eba1, 11);
            b = r(b + h(c, d, a) + x[i+12] + 0x6ed9eba1, 15);
        }
        a += a2, b += b2, c += c2, d += d2;
    }

    //return cast(ubyte[16])[a, b, c, d];
    immutable uint[4] result = [a, b, c, d];
    return cast(ubyte[16])result;
}
```

XOR - Exclusive OR

Exclusive-OR gate

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise XOR ^ [edit]

bit a	bit b	a ^ b (a XOR b)
0	0	0
0	1	1
1	0	1
1	1	0

```
  11001000
^ 10111000
  --------
= 01110000
```

Fun with XOR

$$A \oplus 0 = A,$$

$$A \oplus A = 0,$$

$$A \oplus B = B \oplus A,$$

$$(A \oplus B) \oplus C = A \oplus (B \oplus C),$$

$$(B \oplus A) \oplus A = B \oplus 0 = B,$$

Simple XOR Encryption

Example [edit]

The string "Wiki" (`01010111 01101001 01101011 01101001` in 8-bit ASCII) as follows:

```
  01010111 01101001 01101011 01101001
⊕ 11110011 11110011 11110011 11110011
= 10100100 10011010 10011000 10011010
```

And conversely, for decryption:

```
  10100100 10011010 10011000 10011010
⊕ 11110011 11110011 11110011 11110011
= 01010111 01101001 01101011 01101001
```

judge.py

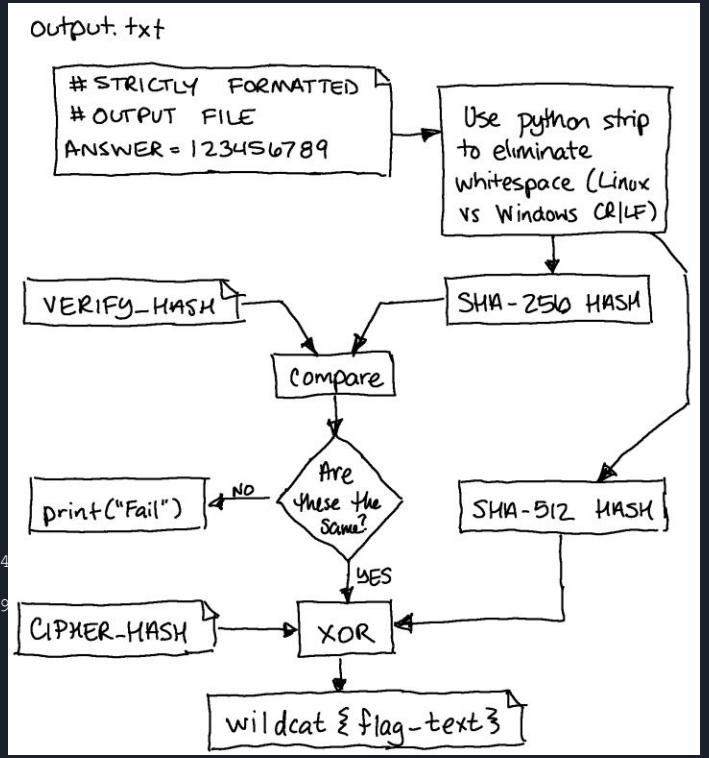
```
astronomer|moonstarer = ANAGRAM
cryptography|catyprogram = NOT AN ANAGRAM
west shore|horse stew = ANAGRAM
gameboy pokemon|moneybag cards = NOT AN ANAGRAM
hot sewers|west shore = ANAGRAM
atelescope|toseeplace = ANAGRAM
westernunion|nowireunsent = ANAGRAM
statueofliberty|builttostayfree = ANAGRAM
debit card|bad credit = ANAGRAM
florida gators|golf radiators = ANAGRAM

$ cat chal.txt | ./solve.py | ./judge.py
Congrats!
wildcat{mixed_up_letters|deer_mutt_pixels}

$ head judge.py
#!/usr/bin/env python3

import sys, hashlib, binascii

VERIFY_HASH = b"474a1422dbc01160342a91b812bcdf8335a4327c8982c627d18b70718a4
CIPHER_HASH =
b"0d5687f336f7b8c6f74b63fc199970ebe03e02132096da800c543cd4fbff1a18e06af4f49
614317ba80"
```



Attributions

- https://computerengineeringforbabies.com/blogs/engineering/xor-gate
- https://www.allaboutcircuits.com/textbook/digital/chpt-3/multiple-input-gates/
- https://en.wikipedia.org/wiki/Bitwise_operations_in_C
- https://godbolt.org/