Level 0x0b

Hashing

Topics

- Events
- Shirts

Code Quest

- Saturday, February 24
- 2.5 hours. Free breakfast and lunch
- Teams are 2-3 students
  - 1 laptop per person
  - Novice division
  - Advanced division (1 programmer with 1 year programming exp)
- High school students, ages 13-18 years old
- Parent Meeting / Transportation Discussion: Fri Feb 16, 3:45 PM, Library

Wildcat Coders 1	Wildcat Coders 2	Wildcat Coders 3	Wildcat Coders 4
Novice	Advanced	Novice	Advanced
- Eshan V - Saaketh K - Shoaib A	- Justin T - Dylan G - Sam S	- Gabriella Z - Dominic M - Avery M	- Jay J - Parker W - Zachary W

Code Quest Schedule

Start	Stop	Event
7:00 AM	8:50 AM	Registration, Breakfast, Setup
8:50 AM	9:30 AM	Welcome & Competition Rules
9:30 AM	12:00 PM	Code Quest Competition / Coaches Corner
12:00 PM	2:20 PM	Lunch and Activities
2:20 PM	3:00 PM	Awards
3:00 PM		Pick up swag bags and exit

Upcoming Events

LOCKHEED MARTIN CYBERQUEST® COMPETITION

- Lockheed Martin Cyber Quest
  - Saturday, March 23rd
  - 3 hours. Free Breakfast and Lunch
  - Teams are 3-5 students
    - 3 laptops per team
  - No team limit given
  - Registration Timeframe:
    - Wed Jan 03 2024 - Mon Feb 26 2024
- Pico CTF (Carnagie Mellon)
  - March 12-26
  - Online CTF

GenCyber Camp

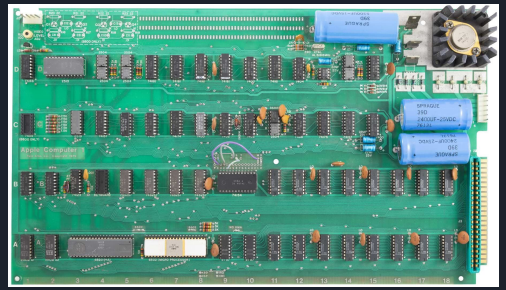
- Cyber Security Summer Camp at Florida Tech
- 60 hours of instruction / labs
- June 10 - 14 - Register at https://event.fit.edu/gencyber
- 40 students, entering 9th - 12th grades
- Big Brother CTF - March 2nd 10AM - 4 PM
- Student selection is on Monday Feb 19th, so register before then!!
- Funded via NSA grant, no cost to students

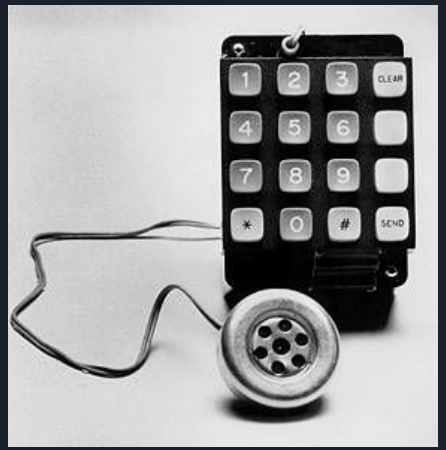
Cyber Fundamentals Course

- The fall course gave out scholarship to winning student
- After school class for 5 weeks in April
- Tuesdays and Thursdays
- Very early planning, stay tuned

Woz
Steve Wozniak

- Phone Phreaker
  - Made $150 blue boxes
- Prankster
  - Bicycle locks
  - Bring your pet to school day
- Breakout Arcade Game
  - -50x chips for $100 each
- Apple I
  - 8-bit computer
  - 1st to connect to TV display





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”

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

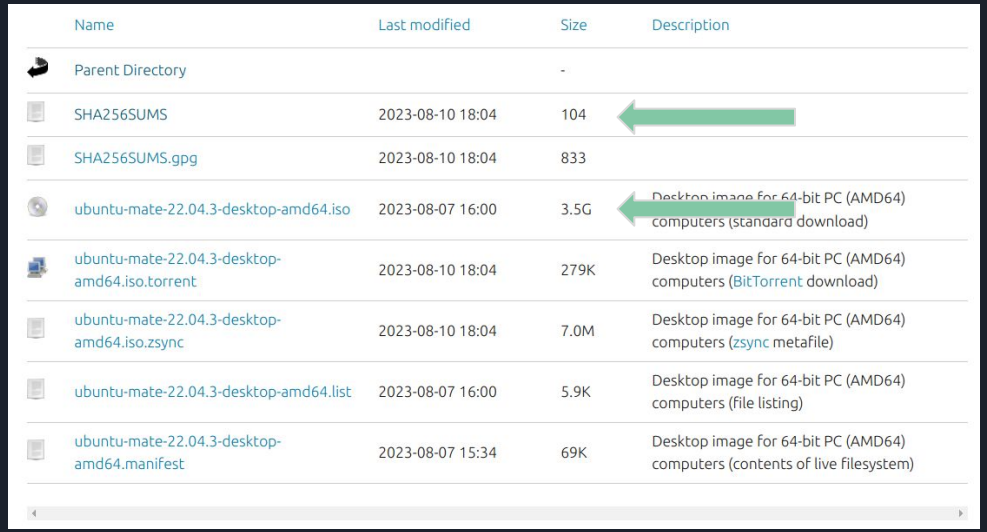
- Example to the right is MD4
  - Obsolete for a long time, simplest of hashes that were once considered cryptographically strong
- Don't reinvent the wheel, use existing hash libraries
  - Similar advice to cryptographic libraries and functions

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;
}
```

File Integrity Verification

```
$ cat SHA256SUMS.txt
d84cd3eb7732fbb39ce3cd24ba1b302a643fe0362f7ac9261fc4c7b756e42a55 *ubuntu-mate-22.04.3-desktop-amd64.iso
$ sha256sum -c SHA256SUMS.txt
ubuntu-mate-22.04.3-desktop-amd64.iso: OK
```



	Name	Last modified	Size	Description
	Parent Directory		-	
	SHA256SUMS	2023-08-10 18:04	104	
	SHA256SUMS.gpg	2023-08-10 18:04	833	
	ubuntu-mate-22.04.3-desktop-amd64.iso	2023-08-07 16:00	3.5G	Desktop image for 64-bit PC (AMD64) computers (standard download)
	ubuntu-mate-22.04.3-desktop-amd64.iso.torrent	2023-08-10 18:04	279K	Desktop image for 64-bit PC (AMD64) computers (BitTorrent download)
	ubuntu-mate-22.04.3-desktop-amd64.iso.zsync	2023-08-10 18:04	7.0M	Desktop image for 64-bit PC (AMD64) computers (zsync metafile)
	ubuntu-mate-22.04.3-desktop-amd64.list	2023-08-07 16:00	5.9K	Desktop image for 64-bit PC (AMD64) computers (file listing)
	ubuntu-mate-22.04.3-desktop-amd64.manifest	2023-08-07 15:34	69K	Desktop image for 64-bit PC (AMD64) computers (contents of live filesystem)

Hashing Passwords

- Bad ways to store password
  - Plaintext
  - Obfuscated
  - Symmetrically Encrypted
- Instead store the hash of the password!
  - Don’t save "TaylorsLuvs87" in plaintext file where hacker might be able to steal it
  - Store 45d3a340e64cdcce74408604b35a3f04 instead
  - Authentication using hashes
    - User sends plaintext password (through encrypted socket)
    - Server computes hash
    - Server compare hash from password user provided to hash in DB
    - User allowed if hashes match
- But we still try really hard to protect the password hash too if possible

Guessing / Cracking Hashes

- No way to figure out what data created a hash
- ***But*** we can ***guess*** data, and see if it generates a matching hash
- Rockyou.txt is a database of 14.3 million plaintext passwords (130MB)
  - My PC computed hash on all of them in 8.5s, and in 3.5s (single core)
  - Can then instantly look up any of them in the database
- 26 letters of alphabet, 10 nums, 10 symbols = 72 possible chars
  - 72^4 = 26 million = 16 seconds of compute time
  - 72^5 = 19 minutes
  - 72^6 = 23 hours
  - 72^7 = 69 days
- Purpose built hash cracking system much better
  - NVidia 3060 Ti does 32 billion MD5s per second
  - 69 days reduces to 5 seconds

How to resist cracking

- Longer passwords
- Don’t do 1 iteration of hash, do it thousands of times
  - Make hash calculation slow / computationally difficult
  - This kinda against the founding principle of good hash function!
- Hashing algorithms that are purposely hard to make fast
  - Bcrypt
  - Argon2
- Hashing algorithms that use up a lot of memory to compute
  - Can’t be accelerated by an ASIC

Precomputation and Rainbow Tables

- Attacker can pre-compute hashes aka Rainbow Tables
  - Record hash / password pair for each guess
  - Keep hashes in order / fast to lookup
  - Takes up a lot of storage potentially
- Crackstation.net
  - Has a 15 GB password database (1.4 billion passwords)
  - Collected from real-world leaks
  - Precomputed about 15 different hash types
- How can we fight against Rainbow Tables?

Salting our Hashes

- For each password we store in our database, add a random salt
- Compute the hash for salt+password
- User doesn't care / know about salt value
- Example:
  Salt: 58Yt9gV
  Pass: swifty87
  Hash: echo -n "58Yt9gVswifty87" | md5sum -
  1034c8cb578ffb4cf5f7fbab9c773f2a
- Each user gets random unique salt
- Attacker can't precompute hashes

Password Reuse Vulnerability

- eHarmony had 1.5 million password hashes get stolen
  - Unsalted md5 hashes
  - 1.2 million were cracked within hours
- Assume taylorswiftfan@gmail.com used eHarmony, and used her swifty87 password
- eHarmony passwords easily get cracked because unsalted md5 weak
  - Precomputation attacks / rainbow tables
  - Not crack resistant
- Attacker now knows taylorswiftfan@gmail.com used swifty87 as password
- Attacker now searches other services person uses, tries this password
- Even if password is properly salted and hashed, attacker can guess this 1 password very easily
- Can attacker find on social media other possible accounts for user?

hashcat

- World’s fastest password cracker
- Very configurable
  - Multiple systems
  - CPU + GPU cracking
- 350+ types of hashes
  - https://hashcat.net/wiki/doku.php?id=example_hashes

Links

- https://techcrunch.com/2014/11/04/nearly-40-years-later-steve-wozniak-still-brainstorms-ways-the-apple-ii-could-have-been-better/
- https://event.fit.edu/gencyber/
- https://crackstation.net/