# Computer and Network Security

## Lecture 08: Authentication

COMP-5370/6370
Fall 2024

# Project 1-A

**We have to be able to build your code.**

3. You must have a `Makefile` in the root directory with two targets:

   `build` — Must compile your code from scratch and exit successfully. If a non-compiled language (e.g., Python), this target is not required to do anything (i.e. "`exit 0`").

**We have to be able to build your code.**

`run` — Must pass the `FILE` argument to `make` to the above compiled program and transparently pass `stdout` and `stderr`.

# Regrade Requests

- Regrade requests come in 2 forms:
  - Fix things that break the auto-grader
  - Fix *minor* things that cause major deduction

# Auto-Grader Errors

- Removed the “@” in Makefile and have extra line of output to stdout?
  - You can add an “@” sign.
- Submission structure wrong?
  - You can move your files around
- Makefile isn’t a makefile?
  - You can fix your Makefile
- Etc.

**YOU DON’T GET AN EXTRA WEEK TO FINISH IMPLEMENTING**

# Fundamental Misunderstandings

- Print to the wrong handle (stdout vs. stderr)?
  - You can change to the correct handle
- Print "`--- simple-string ---` " and not "`--- string ---`"?
  - You can change your print statement's literal
- Etc.

**YOU DON'T GET AN EXTRA WEEK TO FINISH IMPLEMENTING**

# Grade Math is just Math

**Grading**

- **3x Course Projects (each)** — 10%
- **Final Exam** — 25%
- **2x In-Class Exams (each)** — 12.5%
- **Midterm Exam** — 20%

**Calculating Your Course Grade** With your returned scores as a percentile value (i.e. 0% – 100%), fill-in the below formula:

$$0.10 \times project_1 + 0.10 \times project_2 + 0.10 \times project_3 + 0.125 \times exam_1 + 0.125 \times exam_2 + 0.20 \times midterm + 0.25 \times final$$

- A zero (0) on Project 1A means that you have a max final grade of 95% (A)

# Project 1-B

- Released Friday
- Due next Friday, 20Sept2024

Project 1B

Computer and Network Security
COMP-5370/-6370

Released: 06Sept2024
Due: 20Sept2024 at 6pm CT

Part B of this project is due **Friday, 20Sept2024 at 6pm CT** and must be submitted through the Canvas assignment (if early/on-time) or by emailing the TA (if late). Late assignments will be penalized as described in the syllabus.

# Project 1-B --- Part 1

- Trade implementations with a partner and find/demonstrate incorrect behavior
  - Must have **different root-causes**

## 1 Break-It

**This portion of the project must be completed with a single partner.**

For this portion, you should select a partner within the course, exchange implementations, and use your in-depth knowledge of `nosj` from Project 1A to find corner-cases which demonstrate incorrect behavior in your partner's implementation. This may be by error-ing when given valid input, not error-ing when given invalid input, or by incorrectly handling valid input (i.e. the output is wrong). Your goal is to find three (3) different incorrect behaviors with *different root-causes*. It is important to note that three different examples with the same root-cause (i.e. triggering it with just different input) will be counted as only one (1). You **may not** submit any of the testcases from the specification as your testcases for incorrect behavior.

In your submission, you should provide input/output files for each erroneous behavior as well as a short, *less than 100 words*, ascii-only description/explanation identifying A) the root-cause and B) a sufficient explanation of how your partner's implementation could be patched to mitigate the issue. You **do not**

# Project 1-B --- Part 1

- You may only have 1 Partner
  - Your 1A implementation must be used by only one person for 1B
- If you can't find a partner by ***this*** Friday, let us know and we'll work it out
  - If we don't hear from you, assume have partner

**YOU SHOULD NOT BE WAITING ON YOUR PARTNER'S IMPLEMENTATION**

# Project 1-B --- Part 2

- Generate a partial SHA256 hash collision via brute-force attack

## 2 Brute-Force Attacks

**This portion of the project must be completed individually.**

Though brute-force attacks are rarely the best or most-efficient attack, they are always *an attack* that is possible and guaranteed to be successful given sufficient resources. In this portion of the project, you will demonstrate this by implementing a reduced-strength, partial collision attack against a cryptographic hash function (SHA256). You **are not** required to generate a complete collision.

You should be cognizant of the fact that there is often a "point of diminishing returns" where it is more efficient to intentionally perform a non-optimized/inefficient attack rather than continue to optimize the attack/implementation/etc. If it takes 3 hours to optimize code that will reduce the run-time by 3 seconds, you have passed that point (hint…**HINT**).

# Project 1-B --- Part 2

- Generate a ***partial*** SHA256 hash collision via brute-force attack

## 2 Brute-Force Attacks

**This portion of the project must be completed individually.**

Though brute-force attacks are rarely the best or most-efficient attack, they are always *an attack* that is possible and guaranteed to be successful given sufficient resources. In this portion of the project, you will demonstrate this by implementing a reduced-strength, partial collision attack against a cryptographic hash function (SHA256). You **are not** required to generate a complete collision.

You should be cognizant of the fact that there is often a "point of diminishing returns" where it is more efficient to intentionally perform a non-optimized/inefficient attack rather than continue to optimize the attack/implementation/etc. If it takes 3 hours to optimize code that will reduce the run-time by 3 seconds, you have passed that point (hint…**HINT**).

# Project 1-B --- Part 2

- Generate a partial SHA256 hash collision via brute-force attack
- Input requirements: prefix w/ AU email
  - `abc1234@auburn.edu`||{anything you want}
- 2 **DIFFERENT** inputs must collide
- Partial collision requirements:
  - Leading 4-bytes are of the digest are identical
  - Both digests leading 4 bytes are identical

# Project 1-B --- Part 2

- Generate a partial SHA256 hash collision via brute-force attack
- Input requirements: prefix w/ AU email
  - `abc1234@auburn.edu`||{anything you want}
- 2 **DIFFERENT** inputs must collide
- Partial collision requirements:
  - Leading 4-bytes are of the digest are identical
  - Both digests leading 4 bytes are identical

**HEX ENCODING IS 2 CHARACTERS PER BYTE**

# Project 1-B --- Part 2

- **GOOD EXAMPLE:**
  - Digest 1: 0XAAAAAAAA04FDEAB…
  - Digest 2: 0XAAAAAAAA9FDABC3…
- **BAD EXAMPLE 1:**
  - Digest 1: 0XAABBCCDD04FDEAB…
  - Digest 2: 0XAABBCCDD9FDABC3…
- **BAD EXAMPLE 2:**
  - Digest 1: 0XAAAAAAAA04FDEAB…
  - Digest 2: 0XBBBBBBBBFDABC3…

# Project 1-B --- Part 2

## PLEASE READ THE ASSIGNMENT CAREFULLY TO AVOID ISSUES

**WARNING**

Depending on your implementation, **it may take many hours** for your attack to be successful. A non-optimized, single-thread, well-implemented implementation can probabilistically finish using standard, commodity hardware in a short amount of time (order hours) even if using Python. It is *highly recommended* that you validate your implementation's logic with a further-reduced set of restrictions* prior to attempting to generate the partial collision you will submit. This will ensure that your implementation operates as you expect and you do not encounter errors such as:

- Your implementation runs but crashes before finding a solution.
- Your implementation does not find a solution even though guaranteed to be possible.
- Your implementation continues searching after finding a solution.
- Your implementation does not output the found solution.

# Project 1-B --- Part 2

## PLEASE READ THE ASSIGNMENT CAREFULLY TO AVOID ISSUES

7. By default, code will be ran on an up-to-date version of **Ubuntu 24.04** (amd64) without GUI functionality. If you believe your code must be compiled/ran on a different OS or ISA for any reason, you must contact the instructor prior to submission and obtain such approval in-writing.

# Project 1-B --- Part 2

## PLEASE READ THE ASSIGNMENT CAREFULLY TO AVOID ISSUES

`run` — Must execute your partial-collision implementation and output the newly generated partial collision inputs as two BASE64 encoded lines to `stdout` as described above. Your implementation **must not overwrite the** `1-input.txt` **or** `2-input.txt` **files submitted**.

# Exam 1 on Tuesday

- In-Person: During class, pen+paper
- Distance: Identify proctor and schedule
  - Further details in the syllabus

- Multiple choice, True/False, Matching, etc.
- **Short-answers must be short**
  - Will have an anticipated length to give you idea of *how short* your answers should be

- Bonus available but low point value

# Computer and Network Security

## Lecture 08: Authentication

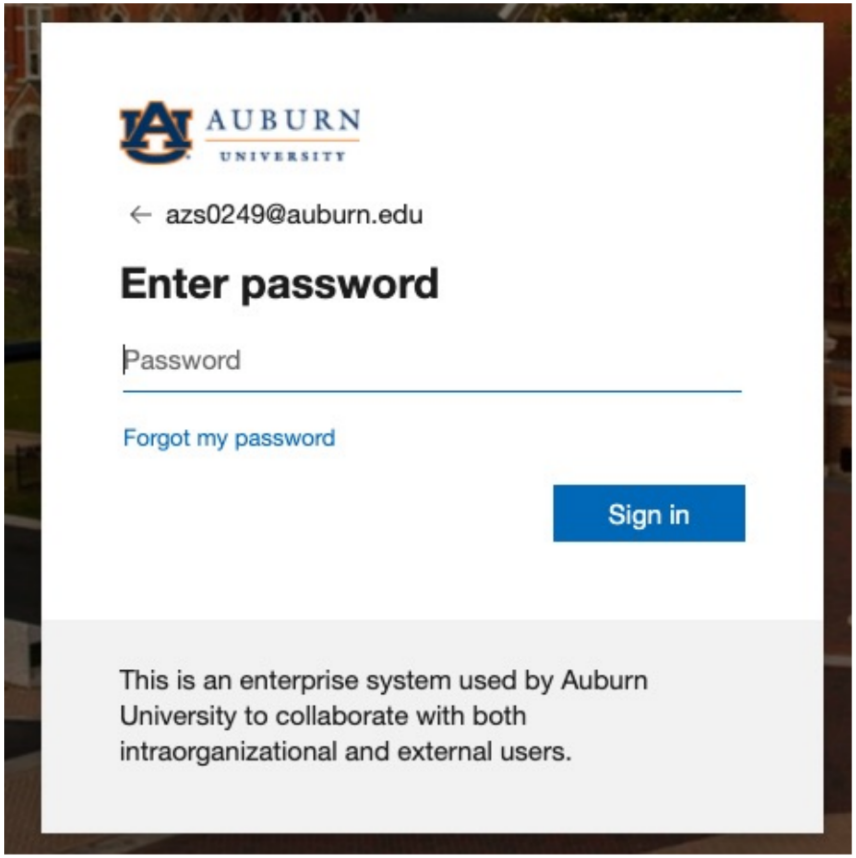
COMP-5370/6370
Fall 2024

# Authentication

**Authentication** is the act of confirming whether or not an actor is who they claim to be to determine subsequent actions.

- Often used in relation to an identity
  - Website authenticates user's identity via username and password
  - Phone authenticates user's identity via fingerprint, facial features, or PIN/password

# Password Authentication




- Original and most ubiquitous form of authentication
- Relatively weak mechanism
- Many well-known and widely exploited problems

# Why are Password Weak?

**because of people**

# Failure: Default/Static Passwords

**Humans often see *the existence of* a password as a sign of security.**

- Devs often hard-code admin access for emergency administrative access.
  - Example of a **static password**
- OEM often set devices identical for ease.
  - Example of a **default password**
- *I'll change it to something stronger…later.*

# Randomness

**Random data** is unpredictable bits to the attacker without any pattern or structure.

- Any bit has exactly the same chance of:
  - Being 0 (50%)
  - Being 1 (50%)

- **Computers are really bad at randomness**
- **Humans are also really bad at randomness**

# Failure: Poor Entropy

**Humans are a poor source of entropy**

Commonly Used Passwords (2019)

| | | |
|---|---|---|
| 123456 | abc123 | 888888 |
| 123456789 | qwerty123 | princess |
| qwerty | 1q2w3e4r | dragon |
| Password | admin | password1 |
| 1234567 | qwertyuiop | 123qwe |
| 12345678 | 654321 | |
| 12345 | 555555 | |
| iloveyou | lovely | |
| 111111 | 777777 | |
| 123123 | welcome | |

Predictable Patterns

- Incrementing suffix
  - *static*||1/2/3/4/…
  - *static*||Jan/Feb/Mar/…
- Repeated words
  - *static*||*static*||*tag*
- Usage reference
  - *static*||Google/Twitter/…

# Failure: Reuse

**Humans reuse passwords due to relatively small storage capacity**

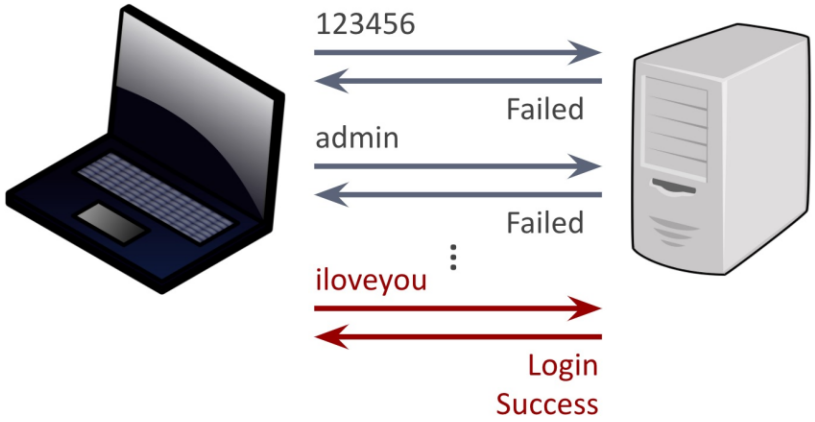
- Nearly everything requires a login
  - Important and unimportant services
- Passwords used passwords (~48 hours)

```
Phone (x4)                Gmail (x5)
BIOS (x2)                 AU SSO login (x1)
OS login (x9)             Amazon (x2)
Disk encryption (x7)      File Encryption (many)
Data Services (x3)        Banking (x5)
```

# How Passwords are Abused

# Problem: Online Guessing

An **Online Password Guessing Attack** is where an actor treats the authentication server as an oracle to identify when it has found the correct password.



# Problem: Online Guessing

## Thinking Like an Attacker

- What is the **easiest** way to win?

## Failure: Poor Entropy

**Humans are a poor source of entropy**

Commonly Used Passwords (2019)

| | | |
|---|---|---|
| 123456 | abc123 | 888888 |
| 123456789 | qwerty123 | princess |
| qwerty | 1q2w3e4r | dragon |
| Password | admin | password1 |
| 1234567 | qwertyuiop | 123qwe |
| 12345678 | 654321 | |
| 12345 | 555555 | |
| iloveyou | lovely | |
| 111111 | 777777 | |
| 123123 | welcome | |

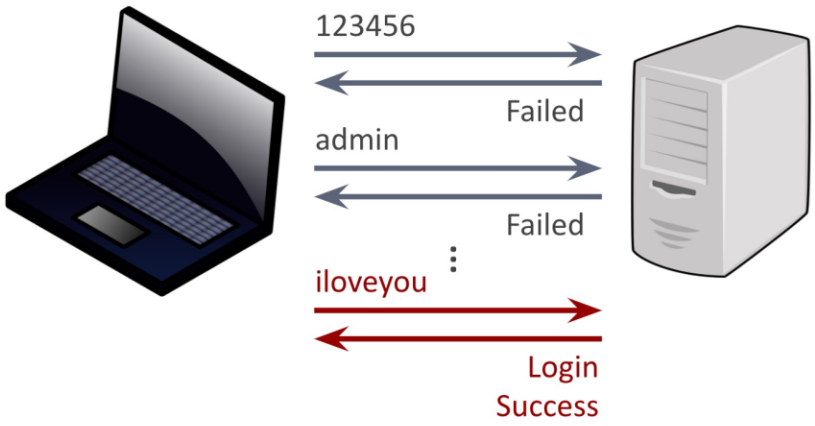
Predictable Patterns

- Incrementing suffix
  - *static*||1/2/3/4/…
  - *static*||Jan/Feb/Mar/…
- Repeated words
  - *static*||*static*||*tag*
- Usage reference
  - *static*||Google/Twitter/…

# Problem: Online Guessing

An **Online Password Guessing Attack** is where an actor treats the authentication server as an oracle to identify when it has found the correct password.

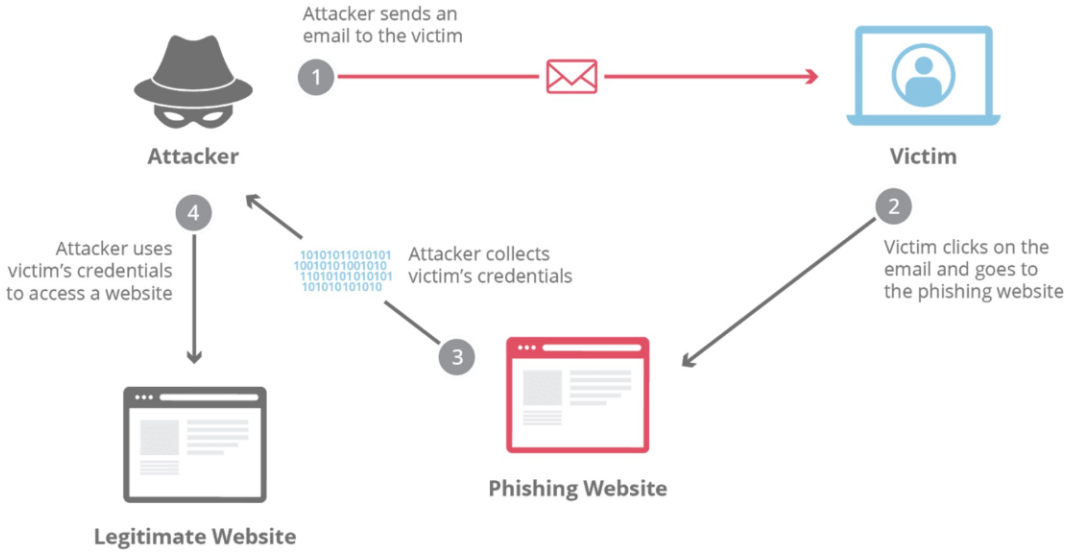
- Defenses
  - Lock-out threshold
  - Rate limits
  - (re)CAPTCHAs
  - Anomaly detection



# Problem: Phishing

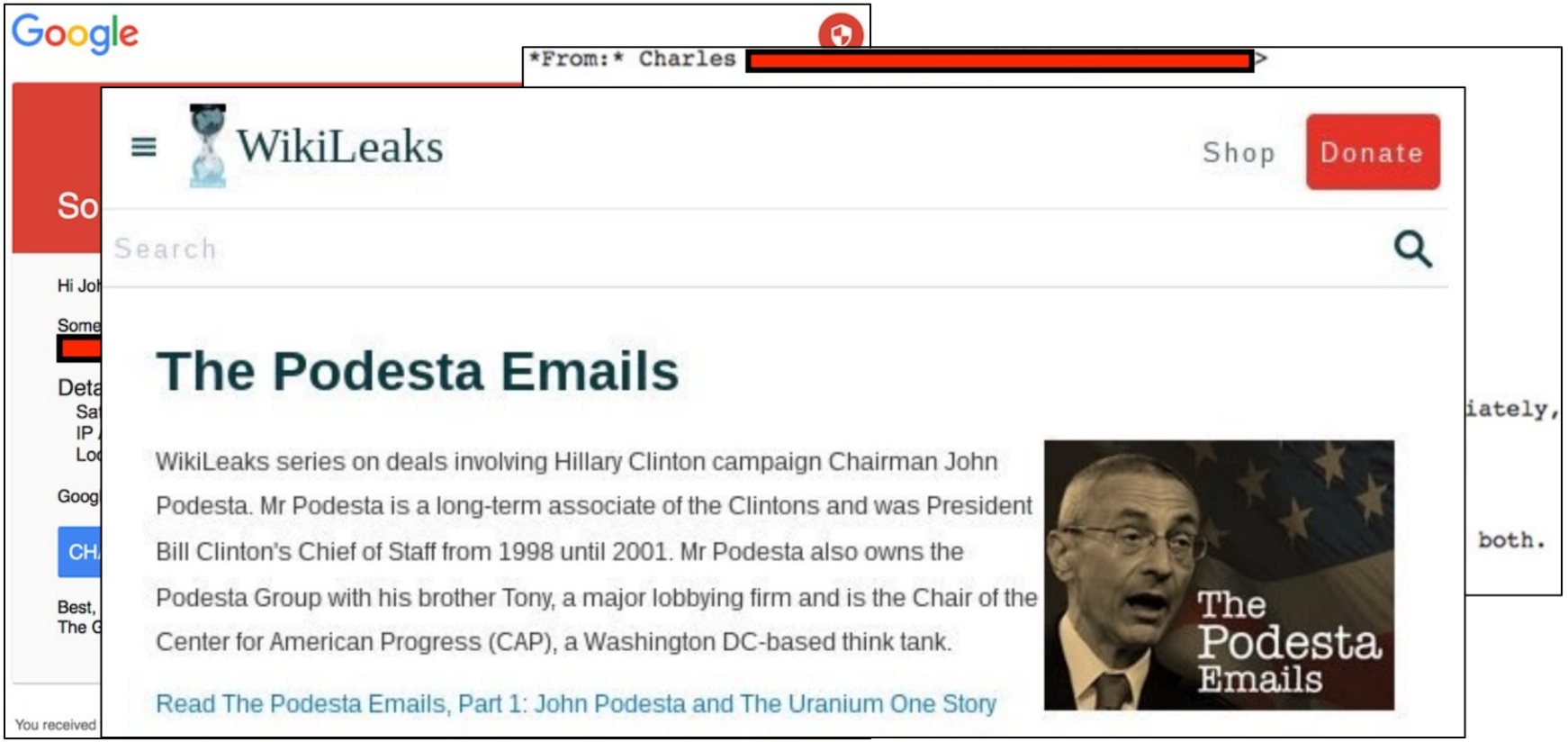
## Humans are poor judges of character

- Users predictably give passwords to attackers without knowing they did so



# Problem: Phishing

## Humans are poor judges of character



# Problem: Password Breaches


- Companies face major financial, reputational, and regulatory risks from breaches
  - Can be mitigated by properly designed infrastructure and protocols

# Problem: Password Breaches

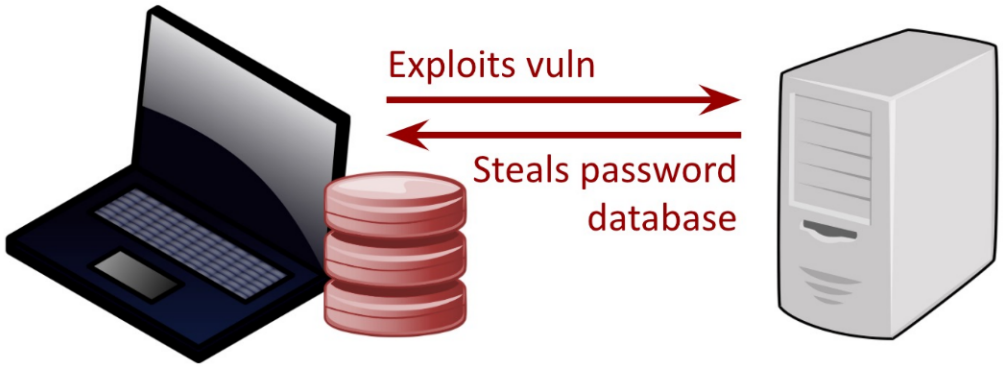
**Passwords should *NEVER* be stored in plaintext in a database *OR ELSEWHERE*.**

**Passwords should *NEVER* be stored in plaintext in a database *OR ELSEWHERE*.**

**Passwords should *NEVER* be stored in plaintext in a database *OR ELSEWHERE*.**

# Problem: Offline "Cracking"

An **Offline Password Guessing Attack** is where an actor treats a database of protected (or unprotected) passwords as an oracle for guessing passwords.



# Hardware Advantages

ars TECHNICA SUBSCRIBE SIGN IN

BIZ & IT —

## 25-GPU cluster cracks every standard Windows password in <6 hours

All your passwords are belong to us.

DAN GOODIN - 12/9/2012, 6:00 PM

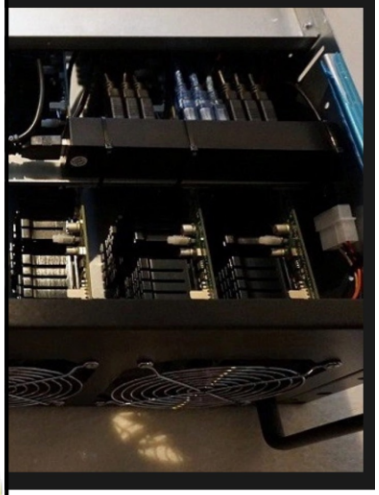
Welcome to Radeon City, population: 8. It's one of five cracking cluster.

HACKADAY

## ALL YOUR PASSWORDS ARE BELONG TO FPGA

by: Tom Nardi 26 Comments

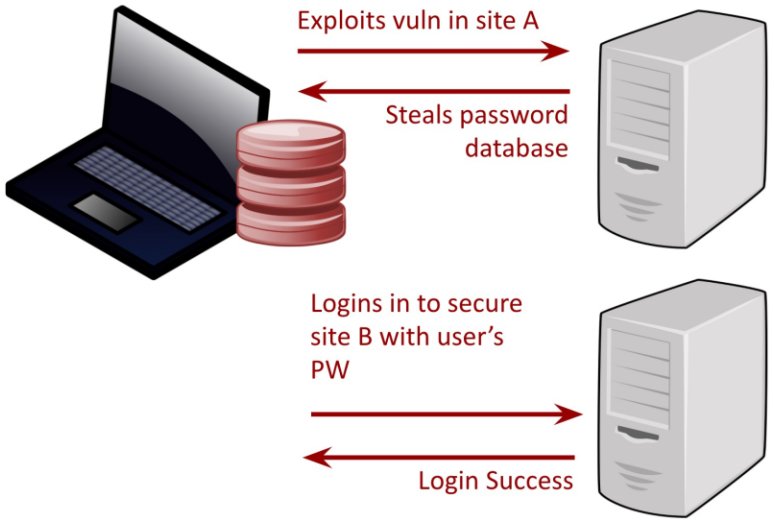
May 15, 2020



# Problem: Credential Stuffing

**Credential Stuffing** attacks involve reusing known username-password combinations from one breach on a separate service.

- Relies on:
  - Ubiquity of accounts
  - Static email addresses
  - Password reuse
  - Laziness

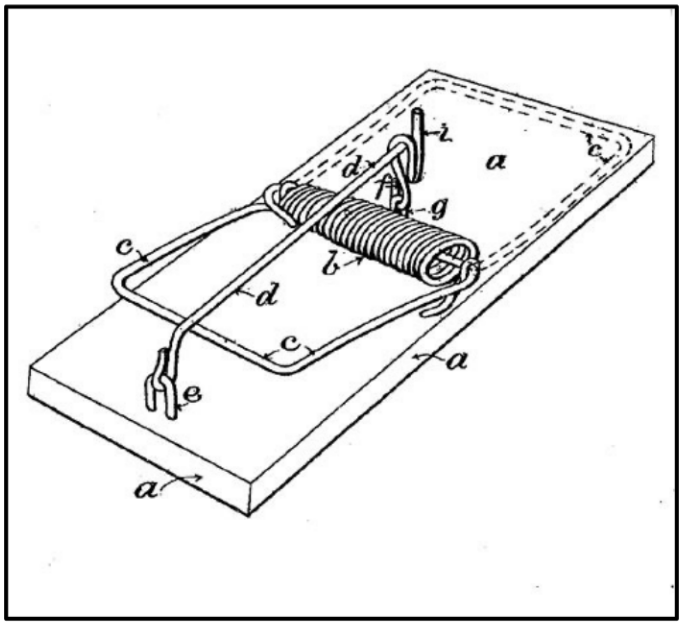


# Problem: Credential Pivoting

**Humans fail to account for implicit trust and reset mechanisms**

- Few services are willing (or able) to bind userdata to passwords
  - Treat passwords as a policy protection
- Provide automated “password reset” mechanisms usually through email

# Defenses: Technical




# Tech Fix: Basics


- Basics Include:
  - **DON'T USE STATIC PASSWORDS**
  - Never write raw secrets to non-volatile storage
  - Be *really* careful when logging related to auth
  - Use pre-existing frameworks
  - **DON'T USE STATIC PASSWORDS**

# Server-Side Password DBs

**Evaluate the below common but poor-practices and identify a scalable mechanism to recover the plaintext passwords.**

- Store in plaintext
  - `abc123`
- Store as ciphertext
  - Encrypt(s, `abc123`)
- Store as obfuscated
  - B64(hex(`abc123`))
- Store as hashes
  - SHA256(`abc123`)

# Tech Fix: Salted Passwords

- Identical passwords have different hashes
  - Unique salt
- Dictionary attacks don't scale
  - 1 common = `n` tries
- Length extension not useful attack
- Rainbow tables are infeasible for general case
  - Salt explodes space

# Tech Fix: Salted Passwords

Alice:

$salt_1$

$Hash(s_1||abcde)$

Bob:

$salt_2$

$Hash(s_2||fghij)$

Charlie:

$salt_3$

$Hash(s_3||klmno)$

...

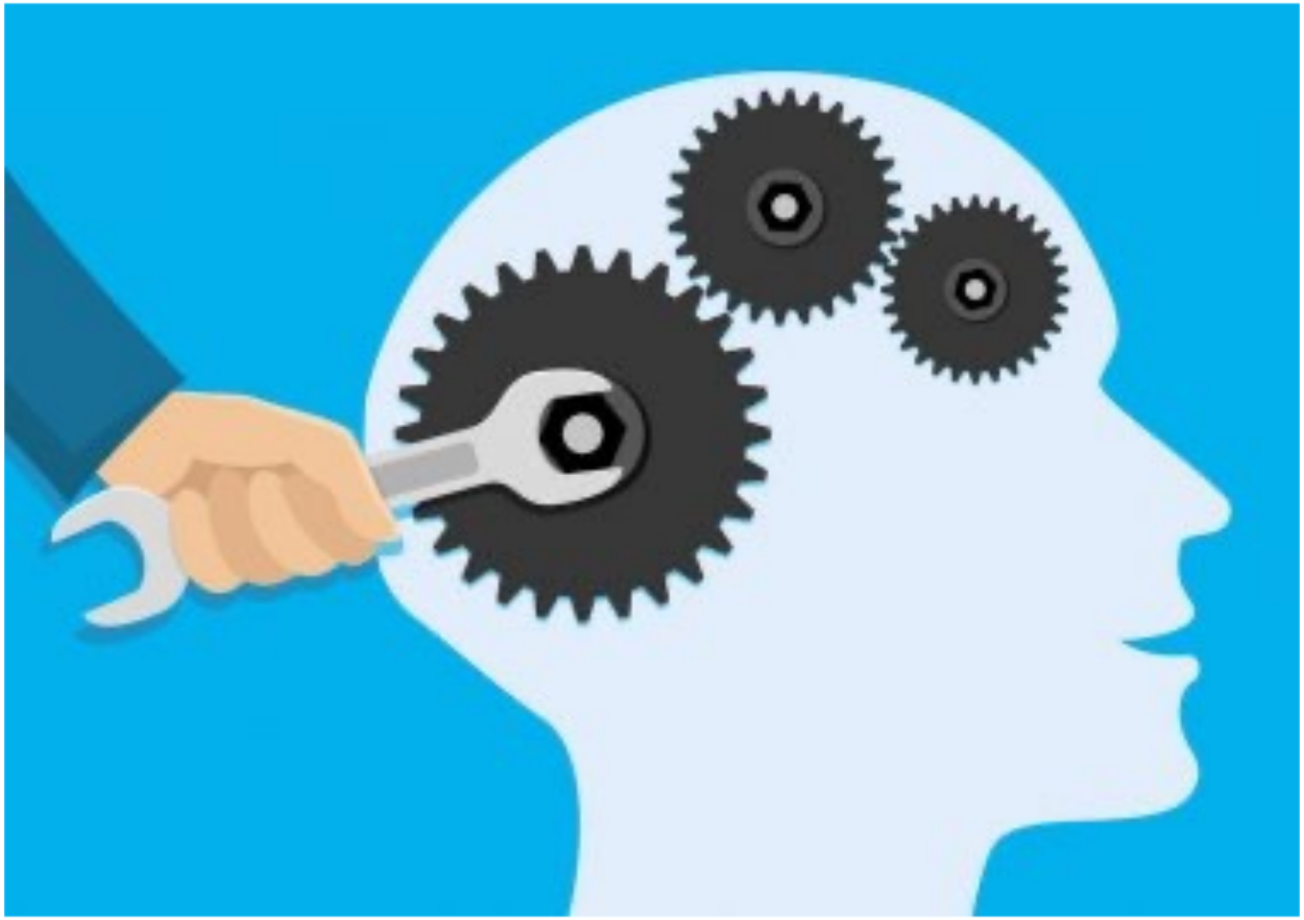
- Salted and hashed password
- Rely on difficulty of preimage attacks and uniqueness of salt

# Tech Fix: Special Hashes

**Use-case makes otherwise undesirable properties advantageous**

- Computationally slow is desired
  - Takes relatively long time to hash
  - Acts as natural rate-limit
- Memory-Hard computation is desired
  - Requires relatively large memory allocations
  - Makes HW optimizations very difficult
- Good functions: PBKDF2, bcrypt, scrypt

# Defenses: Humans



# Why are Password Weak?

**because of people**

# Human Fix: Make Life Easier

## USE A PASSWORD MANAGER

- Greatly improves best practice usability
- User only has to remember single secret
- Easy to rotate when necessary

**Not all password managers are created equal**

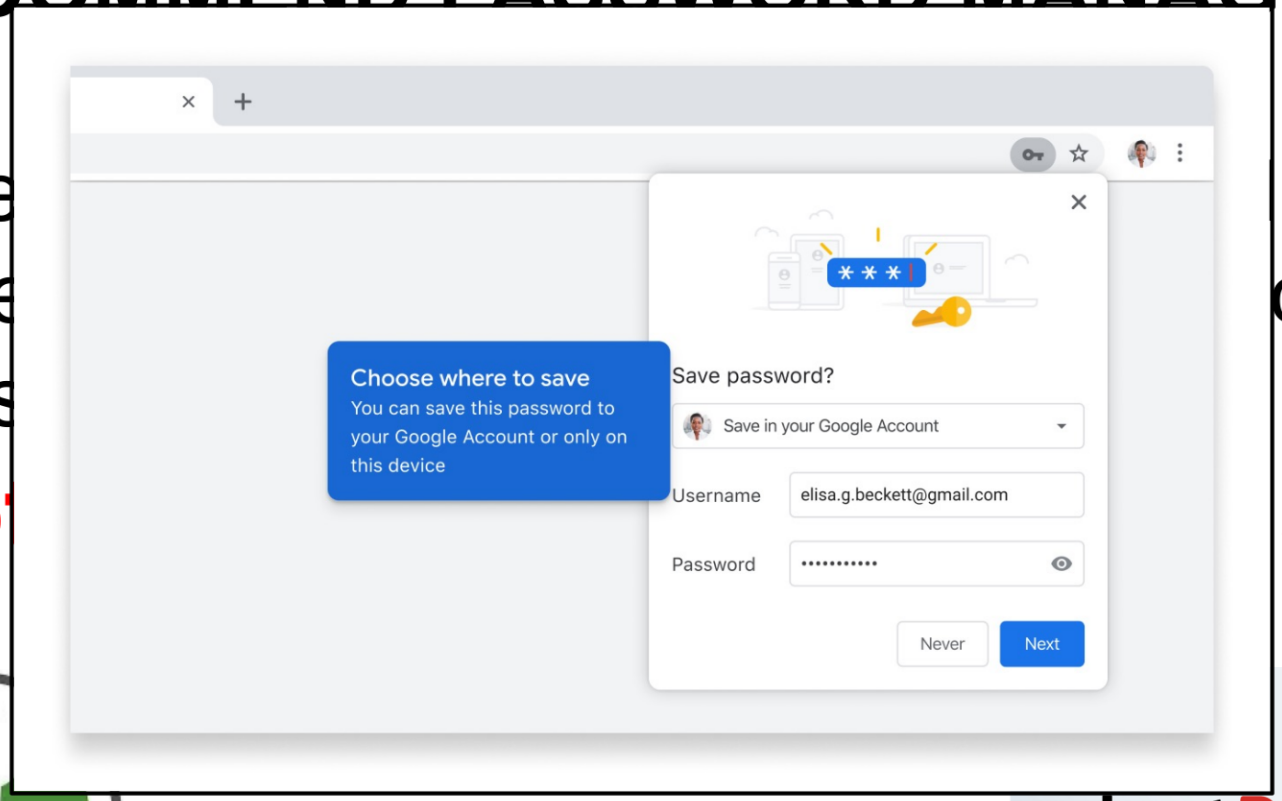
LastPass •••|

# Human Fix: Make Life Easier

## RECOMMEND PASSWORD MANAGERS

- Gre ity
- Use cret
- Eas

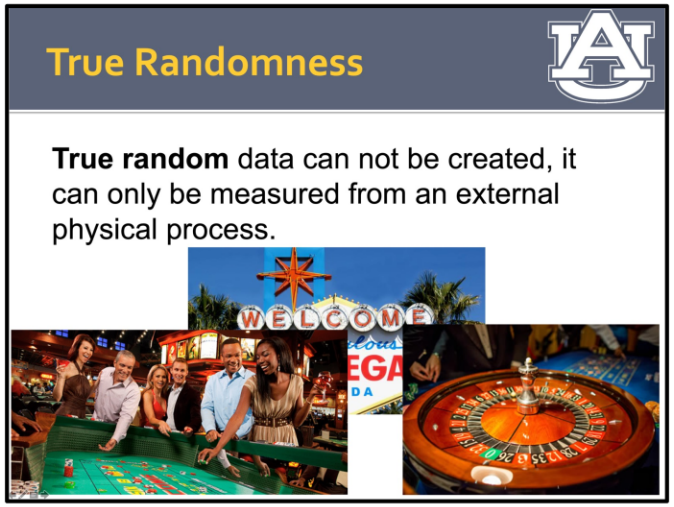
**No are**



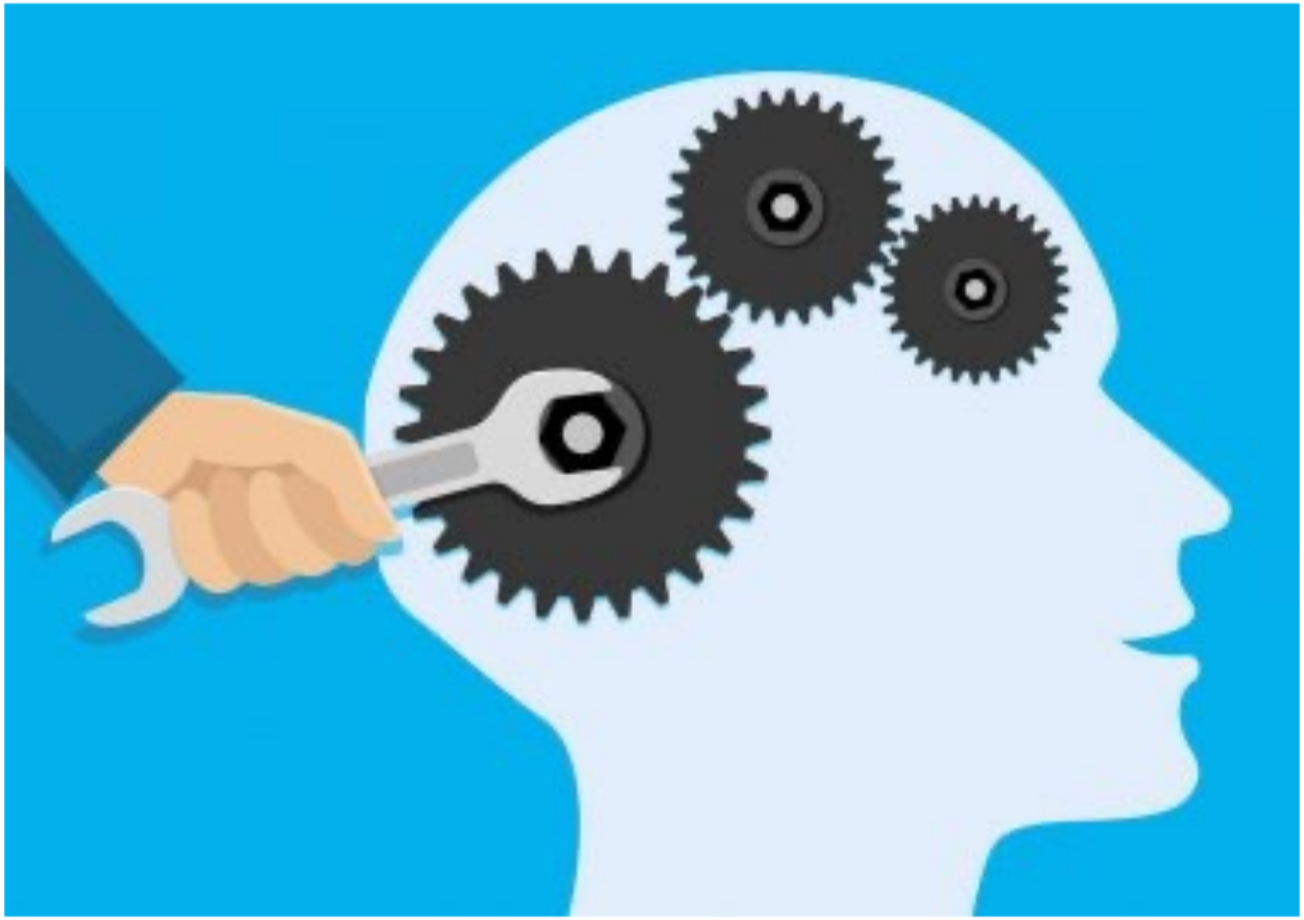

LastPass

# Human Fix: Usable Passwords

- Remove the human as much as possible
- Automated generation
  - `openssl rand -base64 15`
  - `dd if=/dev/random count=15 bs=1 | base64`
- Physical generation (“diceware”)
  - Use wordlist to improve usability and memorability



# Defenses: Fix The Root-Cause



# Multi-Factor Authentication (MFA)

**Multi-Factor Authentication (MFA)** is an approach that *stacks* authentication mechanisms to mitigate each's weaknesses.

- "two-factor authentication" (2FA)

# Multi-Factor Authentication (MFA)

- Something you **know**
  - Password, PIN, pattern
- Something you **have**
  - Phone, security token, ID card
- Something you **are**
  - Biometrics

# Multi-Factor Authentication

- Something you **know**
  - Password, PIN, pattern
  - *Something you can forget*
- Something you **have**
  - Phone, security token, ID card
  - *Something you can lose*
- Something you **are**
  - Biometrics
  - *Something you can mimic or cease to be*

# Computer and Network Security

## Lecture 08: Authentication

COMP-5370/6370
Fall 2024