CS 4460 - Final Project - CTF

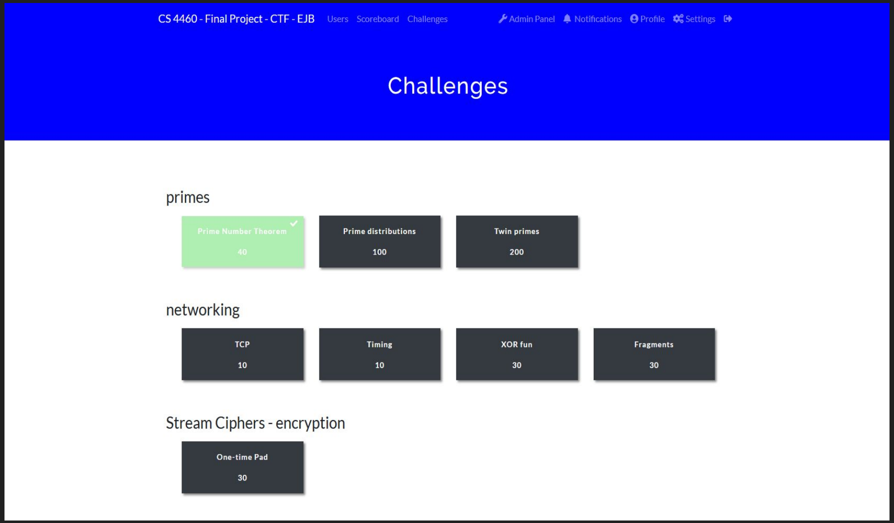
Esperanza Hauptman, Jacob MacInnes, Mark McKee

CTFd Setup

- Started out with CTFd + Docker Compose
  - Flexibility with Docker containers
  - Ran locally
    - Issues running on Ubuntu VM with arm64 host
- Export/import challenges through CSVs
  - Only supported challenges
  - Had to add flags, hints manually
- Export/import whole server instances
  - .zip of .json files describing database
  - Captures whole server state

Final Setup

- Eventually settled on hosted instance of CTFd
    - Allowed for some sophisticated features
        - multiple choice challenges
        - code challenges
    - Easiest to set up since
        - Everything was already configured on the free version of CTFd
        - Previously hosted locally on team member's LAN
- https://ejmcs4460ususpring24.ctfd.io/



Challenge Categories

- Encryption
  - RSA
  - Diffie-Hellman
  - One-time pad
  - Caesar cipher
- Number Theory (basis for encryption)
  - Prime Number Theorem
  - Distribution of large prime numbers
- Linux
  - Simple C program
  - Simple linux cat CLI
  - Imprecise sudo permissions
- Real website hacking
  - Hidden flags in a website
- Networking
  - Wireshark capture analysis
  - TCP
  - XOR encryption
  - ICMP request and IPv4 packets
- Class material reinforcement
  - HOTP-HMAC computations
  - SQL query
  - Wi-Fi protocol/specifications
- Easy/Practice
  - ASCII character
  - practice/testing CTFd instance

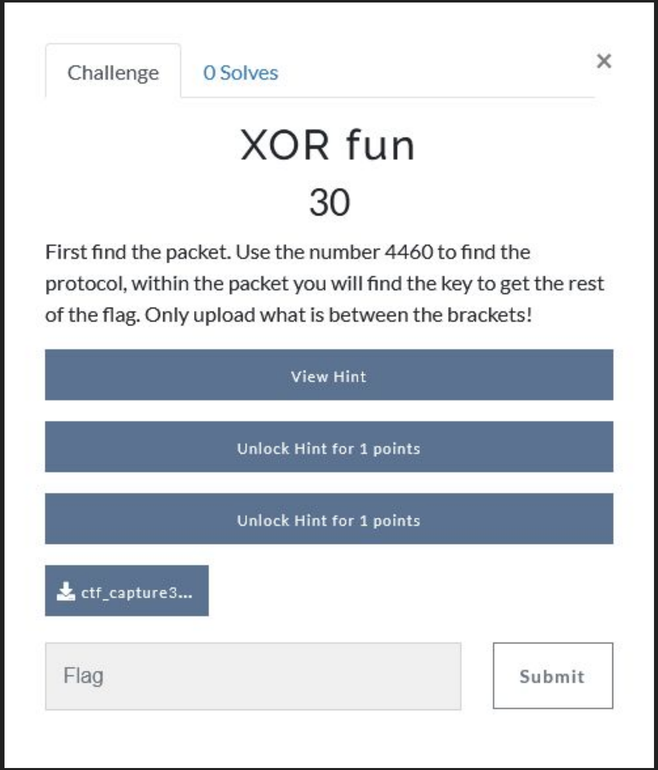
Creating Challenges: Encryption

- RSA
  - $n = p * q$ is given for p, q primes
  - (n, e) public key is given
- find private key (n, d)
  - with minimal d
  - without using online calculators
- One challenge has large primes
- Another challenge asks to decrypt an encrypted message through ASCII conversion

- One time pad
  - Demonstrates how to recover the original messages from a one-time pad if it gets reused.
    - Relevant to WEP
- Diffie-Hellman
  - primitive roots of primes
  - Example upcoming

Creating Challenges: Networking

- TCP
  - Learn to use Wireshark filters to locate TCP packet.
  - Flag found by viewing packet content.
- Timing Analysis
  - Identify timestamps of specific packets.
  - Sum timestamps to reveal flag.
- XOR Encryption
  - Understand XOR encryption basics.
  - Decrypt messages using given hint (in hexadecimal).
  - Carefully select decryption base.
- ICMP Message Reconstruction
  - Find messages within ICMP requests.
  - Locate specific ICMP request using filter.
  - Reconstruct messages from fragmented requests.

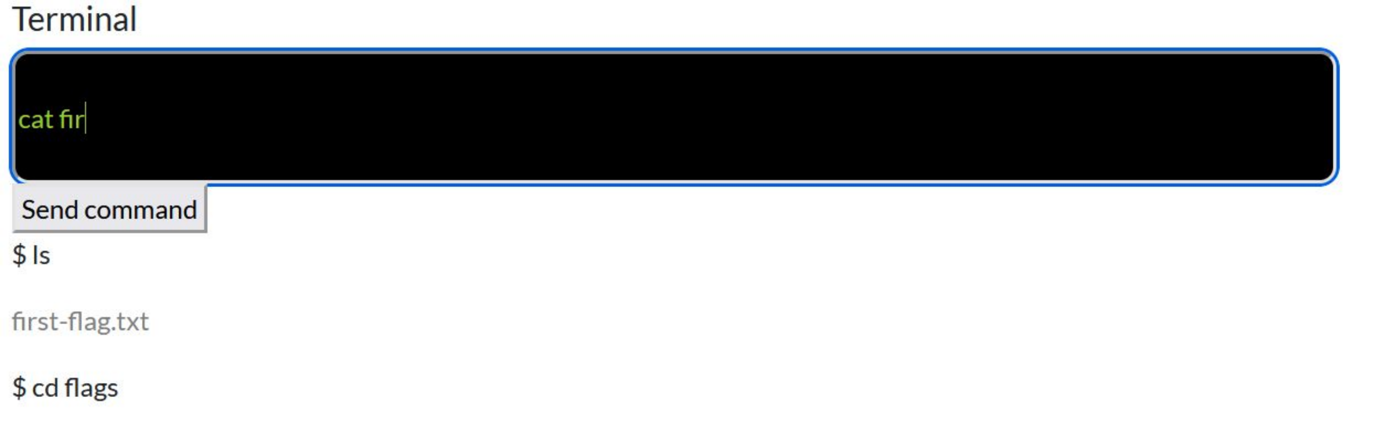


Creating Challenges: Sudo Challenge (Linux)

Demonstrates how sudo permissions can be used to do things that were unintended

```
ctfd ALL = (root) NOPASSWD: /usr/bin/ls *
ctfd ALL = (root) NOPASSWD: /usr/bin/cat /flagless/*
```

The fake Linux terminal for performing the challenge (below); sudoers.d file for the user (above)



Diffie-Hellman Demo

Let p be an odd prime.
From algebra (beyond Calculus) it is known that
For every integer a with $1 \leq a < p$, that
there exists a smallest integer n so that

$$a^n = 1 \pmod{p}$$

Further, it is known that for each such n,
that $n \mid (p - 1)$ (n divides $p - 1$)
g is a **primitive root** of p if the smallest
m with $g^m = 1 \pmod{p}$ is

$$m = p - 1$$

Primitive roots always exist. The number of primitive roots equals the number of integers x, with $1 \leq x < p$ with x being relatively prime to $p - 1$.

Finding the smallest primitive root of a prime can be useful.

If g is a primitive root for p, and a is any integer with $1 \leq a < p$, there exists a unique m with

$$a = g^m \text{ and } 1 \leq m < p$$

Diffie-Hellman:

Alice and Bob both share a secret odd prime p.
They also both share a secret primitive root of p, say g.

Alice has a secret exponent a, and
Bob has a secret exponent b.
Neither Alice or Bob know the other's exponents.

What is the secret (shared) key?
Alice sends A to Bob: $A = g^a$.
Bob sends B to Alice: $B = g^b$.

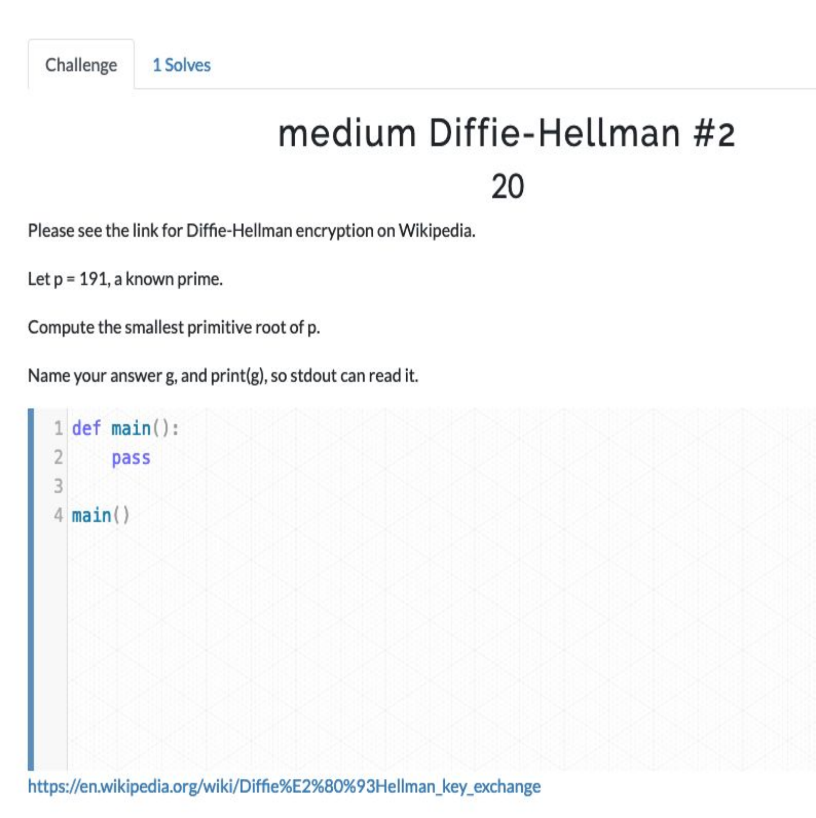
Alice computes $B^a = (g^b)^a = g^{ba} = D$
Bob computes $A^b = (g^a)^b = g^{ab} = D$

$D = g^{ab}$ is the shared secret key.

Diffie-Hellman Demo: pseudocode

```
prime1 is an odd prime
set primitive_root = 0
run loop:  g in (2, prime1):
        temp = 1
        power = 1
        run loop:  exponent in range(1, prime1):
                temp = temp * g (mod prime1)
                if temp ==1 and exponent < prime1 - 1:
                        break
                else if temp == 1 and exponent == prime1 - 1:
                        power = exponent
                        break
        if power == prime1 - 1:
                primitive_root = g
                Break
```

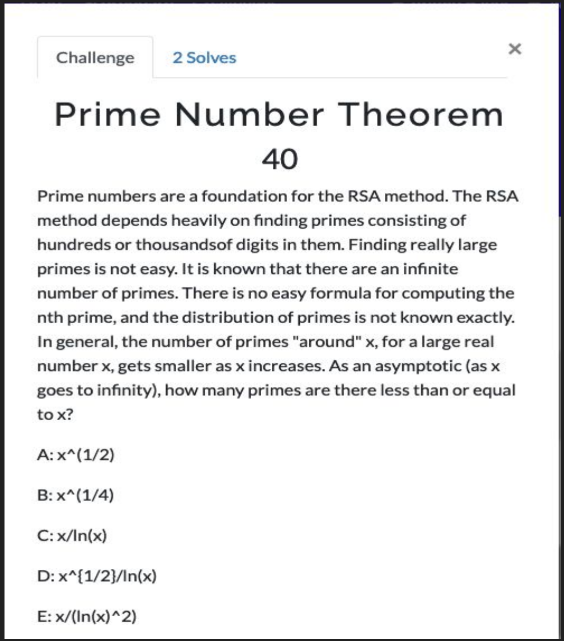
Smallest primitive root of p computed: primitive_root



Theoretical interest: large primes needed for RSA

- RSA encryption method:
  - n = p * q   p and q primes
  - method explained in this course
- RSA conceptually easy
- Important
  - digital signatures
  - handshake leads to session keys
- Problems:
  - slow, due to exponent computations
  - very large primes are needed for security

- Theoretical problems
  - Large primes are difficult to find
- Theory: primes “thin out” as they get larger



Challenge   2 Solves   ×

Prime Number Theorem
40

Prime numbers are a foundation for the RSA method. The RSA method depends heavily on finding primes consisting of hundreds or thousandsof digits in them. Finding really large primes is not easy. It is known that there are an infinite number of primes. There is no easy formula for computing the nth prime, and the distribution of primes is not known exactly. In general, the number of primes "around" x, for a large real number x, gets smaller as x increases. As an asymptotic (as x goes to infinity), how many primes are there less than or equal to x?

A: x^(1/2)

B: x^(1/4)

C: x/ln(x)

D: x^{1/2}/ln(x)

E: x/(ln(x)^2)

Thanks for watching