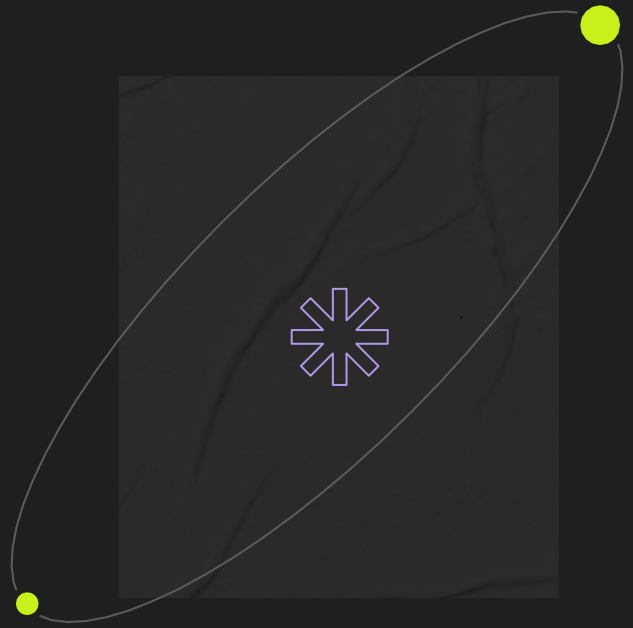
htb signing factory

our approach to solving this challenge

01

What is htb?



Website Overview

What is hackthebox

Create an engaging, interactive learning environment.
Provide scalable cybersecurity training solutions.
Implement robust security measures for enhanced learning.

What are the Key Objectives

- Interactive Labs: Realistic, downloadable practice boxes.
- CTF Exercises: Built-in Capture The Flag challenges.
- Challenges: Puzzles involving hidden HTML, obfuscated JavaScript, and CSS.
- Community: Access to a vibrant community of like-minded individuals.

HACKTHEBOX

Products Solutions Pricing Resources Company Business Login Get Started

Your Cyber Performance Center

Build and sustain high-performing cyber teams keeping your organization protected against real world threats.

Read more Get a demo

NEW Start a 14-day business trial FOR FREE ›

Deloitte intel SIEMENS Synack Adeptis aws

Signing Factory

Objectives

Understand and exploit cryptographic vulnerabilities.
Enhance problem-solving skills through complex scenarios

Premium Subscription Model

Cryptography: Deep dive into encryption and decryption mechanisms.
Web Security: Focus on securing web applications against common threats.
Real-World Scenarios: Practical, real-world inspired scenarios to test and improve skills.

Signing Factory

MEDIUM

DIFFICULTY RATING

30 POINTS

OFFLINE

Start Instance
Start playing the challenge.

Download Files
Necessary files to play the challenge.

Submit Flag
Submit a flag to this challenge.

Add To-Do List
Add this challenge to your list.

Review Challenge
Rate and send your feedback.

Forum Thread
Join the Forum discussion.

INFORMATION | ACTIVITY | CHANGELOG | REVIEWS | WALKTHROUGHS

SHARE RESULTS

CHALLENGE DESCRIPTION

After studying about vulnerabilities on signing servers, a group of researchers gathered one night and comitted into creating a modern and more secure way of signing tokens for authentication. They are certain that their product is ready for distribution and want to do a last security audit before publicizing their work. They provided you with access to the server. Is their way of signing messages the solution to all previous attacks?

4.4	189	Crypto
CHALLENGE RATING	USER SOLVES	CATEGORY

208 Days	connar
RELEASE DATE	CHALLENGE CREATOR GIVE RESPECT

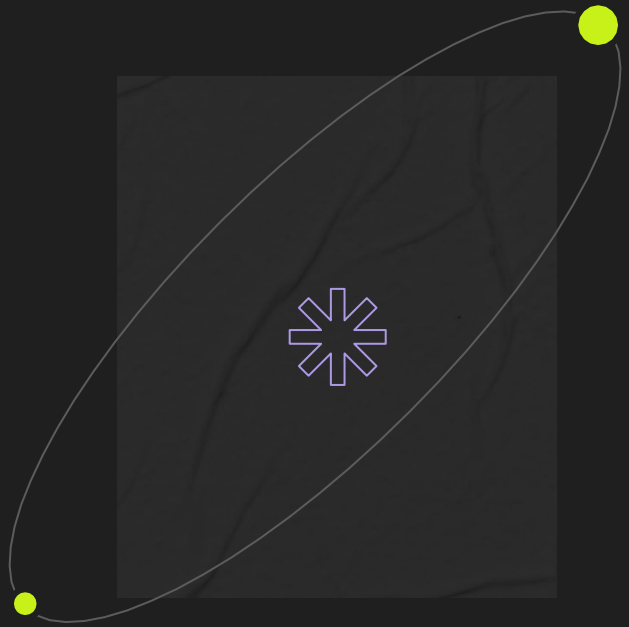
Challenge Flag

fsharp

After studying about vulnerabilities on signing servers, a group of researchers gathered one night and committed into creating a modern and more secure way of signing tokens for authentication. They are certain that their product is ready for distribution and want to do a last security audit before publicizing their work. They provided you with access to the server. Is their way of signing messages the solution to all previous attacks?

02

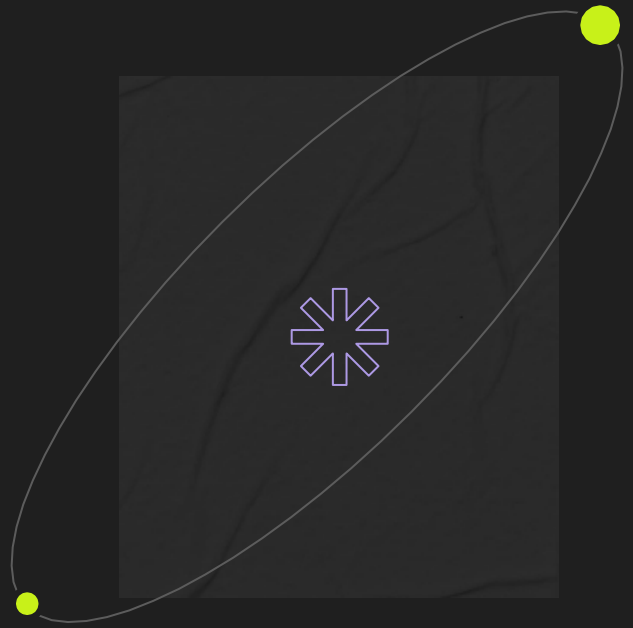
The Crown Jewel or the Flag



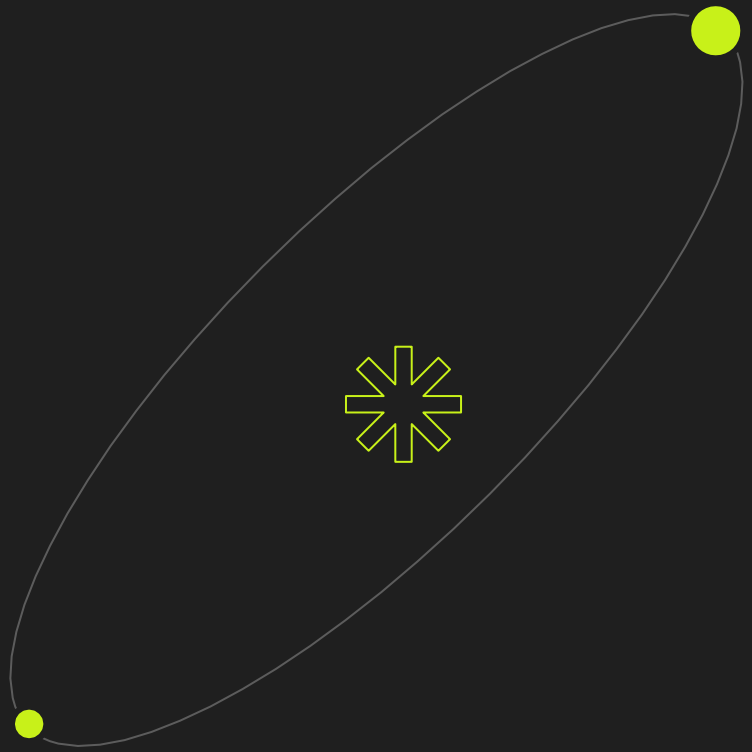
The crown jewel or the flag in this challenge is going to be a way of getting the key of the signing mechanism..... but we didn't know that before ;)

03

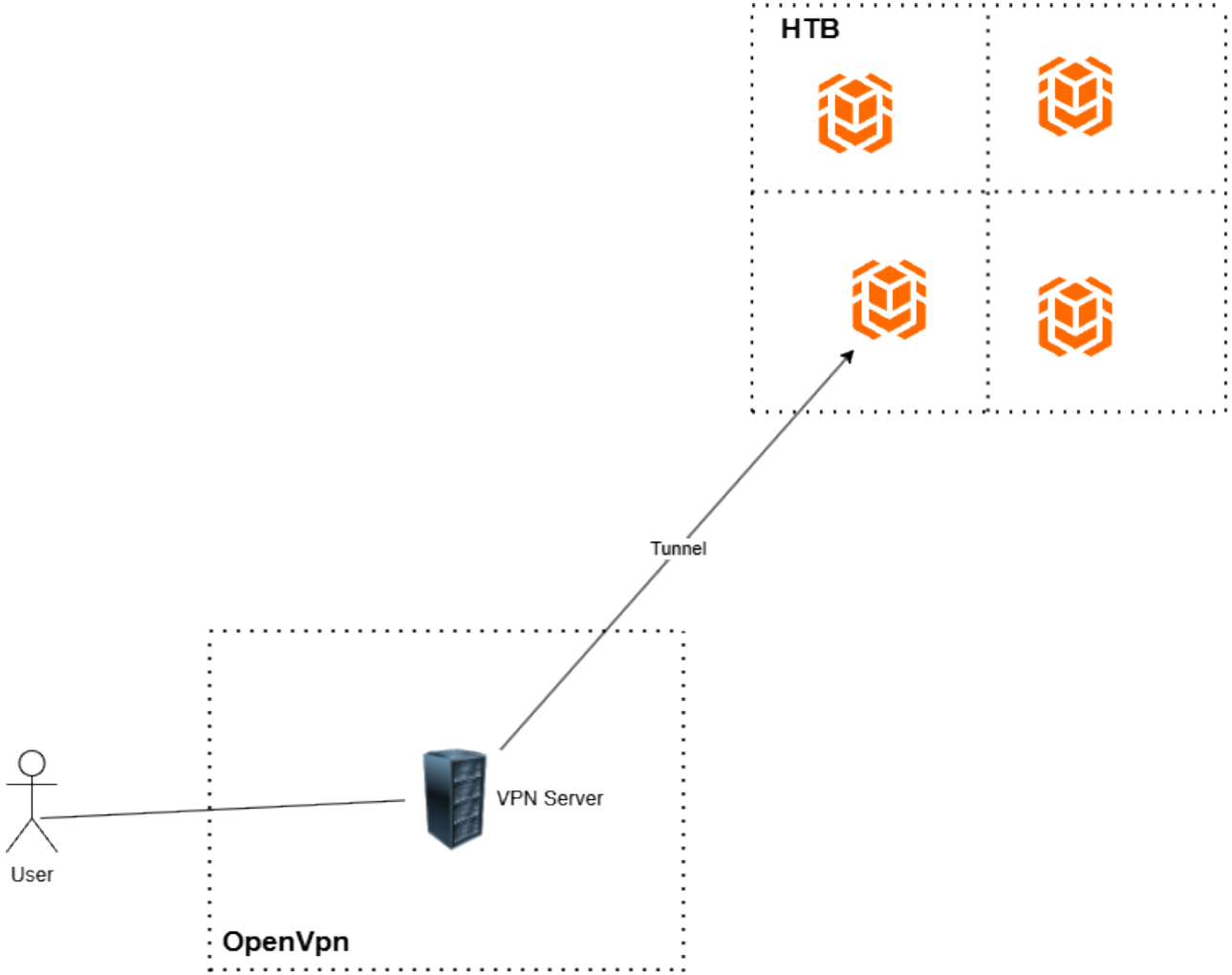
Approach to solving it



Connecting to the box



OPENVPN


HTB
Tunnel
VPN Server
User
OpenVpn

NMAP Scan

```
PORT      STATE    SERVICE VERSION
22/tcp    open     ssh     OpenSSH 9.2p1 Debian 2+deb12u3 (protocol 2.0)
25/tcp    filtered smtp
111/tcp   open     rpcbind 2-4 (RPC #100000)
45100/tcp open     unknown
49999/tcp open     http    Node.js Express framework
51103/tcp open     http    nginx
55056/tcp open     http    Apache httpd 2.4.41 ((Ubuntu))
42939/tcp open  unknown
```

BruteForce Attack on ssh

```
┌──(kali㉿ms2510)-[~/Downloads]
└─$ sudo hydra -l root -P /usr/share/wordlists/metasploit/unix_passwords.txt -t 6 ssh://94.237.62.6
[sudo] password for kali:
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics an
yway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-12-05 10:26:00
[DATA] max 6 tasks per 1 server, overall 6 tasks, 1009 login tries (l:1/p:1009), ~169 tries per task
[DATA] attacking ssh://94.237.62.6:22/
[ERROR] target ssh://94.237.62.6:22/ does not support password authentication (method reply 4).

┌──(kali㉿ms2510)-[~/Downloads]
└─$
```

Attempting to Connect to Apache Server

```
┌──(kali㉿ms2510)-[~/Downloads]
└─$ nc 94.237.62.6 55056
(UNKNOWN) [94.237.62.6] 55056 (?) : Connection refused

┌──(kali㉿ms2510)-[~/Downloads]
└─$ nc 94.237.62.6 55056
(UNKNOWN) [94.237.62.6] 55056 (?) : Connection refused

┌──(kali㉿ms2510)-[~/Downloads]
└─$
```

Attempting to scan the apache server

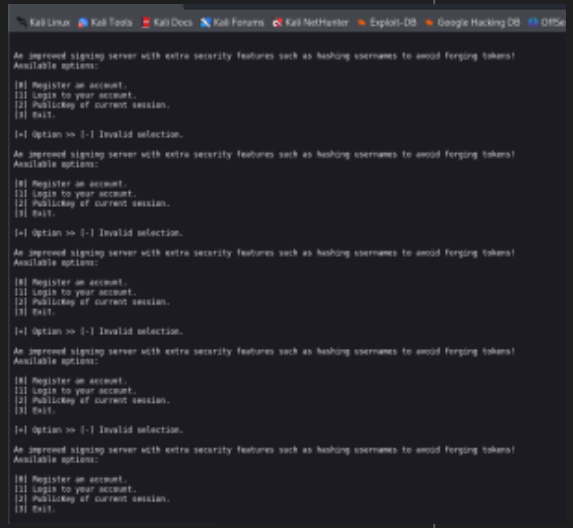
```
┌──(kali㉿ms2510)-[~/Downloads]
└─$ sudo nikto -h 94.237.62.6 -port 55056
- Nikto v2.5.0
---------------------------------------------------------------------------
---------------------------------------------------------------------------
+ 0 host(s) tested
```

Attempting to connect to port 42939

TCP/UDP uncommon port generally used by gaming applications

Curl

```
┌──(vhm0013㉿csce5552)-[~]
└─$ curl http://94.237.59.180:42939
curl: (1) Received HTTP/0.9 when not allowed

┌──(vhm0013㉿csce5552)-[~]
└─$ curl --http0.9 http://94.237.59.180:42939

An improved signing server with extra security features such as hashing usernames to av
oid forging tokens!
Available options:

[0] Register an account.
[1] Login to your account.
[2] PublicKey of current session.
[3] Exit.

[+] Option >> [-] Invalid selection.

An improved signing server with extra security features such as hashing usernames to av
oid forging tokens!
Available options:
```



Netcat

```
┌──(vhm0013㉿csce5552)-[~]
└─$ nc 94.237.59.180 42939

An improved signing server with extra security features such as hashing usernames to av
oid forging tokens!
Available options:

[0] Register an account.
[1] Login to your account.
[2] PublicKey of current session.
[3] Exit.

[+] Option >>
```

Server Options - Signing Process

- [0] Register an Account
  - Users can register providing username
  - Username is hashed based Golden Ratio **2654435761**
  - Hashed username is signed private RSA -> tol
- [1] Login to your account
  - Authentication provided by username Base64
  - Server verifies the information provided
  - Validates the RSA signature with sess (e,N)
- [2] Retrieve the Public key
  - Modular equation provided
- [3] Exit
  - Close the connection

```
┌──(vhm0013㉿csce5552)-[~]
└─$ nc 94.237.59.180 42939

An improved signing server with extra security features such as hashing usernames to av
oid forging tokens!
Available options:

[0] Register an account.
[1] Login to your account.
[2] PublicKey of current session.
[3] Exit.

[+] Option >>
```

```
[+] Option >> 0
Enter a username: vhm0013
Your session token is b'MTA2ODAxODQ1MTA2Njg2Mjg2NzgwNTEwNTAyMzk4ODMwOTcwNjMyNDY5MTYzMDcyNjE0OTAxNDY0NzQ4Mzg4Mzc1NTc2NTAzMjIyMjc1OD
Y5ODE4NzU5OTA1MjQxNDE4MzQ3NTE2MTAyMTYzMDE4NzI3NTEyMDc5MjE3NjE4Mzk2MjY5MjA1ODg4ODI4NDQwNjM3MDMxMjY2NDIyMDcyMzQxMTg5OTEwNTcyMDk5ODQ1
MzI1MjcxNzc5NDEzOTA2NDkyOTcxMzYwNTI4OTU1MDU5OTA1Njg3MDg4OTI5MTI0NzY2NTk2ODU1NjUyOTc3NjIyMDIwOTg5MzIzMzUxNTMxMDcxMDY2NTExNzAzMjEzMj
E3OTQxNTM4Nzg2MTE4NjI0NTY4MDg2ODYwOTU2MzY1MzUzNDcyMTMwNzgzNTYzMjMxOTcyNTQxODYwOTcyODc4Mjc2MTcxNjA0Njg3MDAzNDAzNTM3NDI1MzI2NDg1OTg0
OTEwOTE5OTc0MTI3MTczMDk3NzM0NjExNzM5MDAxODQ4NDQ2NjE0NTI1MTc4NTg2MDE5MjI1OTY3NjkzMjQxODAxOTc4MzcyMDYxNTIxNjc1Njc0MjgyMzM0OTQyNzM1OT
c3MjAwNzQwMjU3ODYwNDI3MjI5ODc5OTE1MDcxMjA5ODEyNTk0NjQ2ODIwNTQzNzA1ODMyMDE5NzE5OTA2OTE2NDcxNjIwNTQxOTMwMjcyMzk1ODc0NTc4MzAyMTYyNDk2
NjA4MjQ5MDEyMjA1MDAyMTUzMzQ1NTU3ODg5Nzg1NDAxODM4MzY1NDkxNjczMTM0MzY=

An improved signing server with extra security features such as hashing usernames to avoid forging tokens!
Available options:
```

```
[+] Option >> 1
Enter your username: vhm0013
Enter your authentication token:
```

```
[+] Option >> 1
Enter your username: vhm0013
Enter your authentication token: MTA2ODAxODQ1MTA2Njg2Mjg2NzgwNTEwNTAyMzk4ODMwOTcwNjMyNDY5MTYzMDcyNjE0OTAxNDY0NzQ4Mzg4Mzc1NTc2NTAzM
jIyMjc1ODY5ODE4NzU5OTA1MjQxNDE4MzQ3NTE2MTAyMTYzMDE4NzI3NTEyMDc5MjE3NjE4Mzk2MjY5MjA1ODg4ODI4NDQwNjM3MDMxMjY2NDIyMDcyMzQxMTg5OTEwNTc
yMDk5ODQ1MzI1MjcxNzc5NDEzOTA2NDkyOTcxMzYwNTI4OTU1MDU5OTA1Njg3MDg4OTI5MTI0NzY2NTk2ODU1NjUyOTc3NjIyMDIwOTg5MzIzMzUxNTMxMDcxMDY2NTExN
zAzMjEzMjE3OTQxNTM4Nzg2MTE4NjI0NTY4MDg2ODYwOTU2MzY1MzUzNDcyMTMwNzgzNTYzMjMxOTcyNTQxODYwOTcyODc4Mjc2MTcxNjA0Njg3MDAzNDAzNTM3NDI1MzI
2NDg1OTg0OTEwOTE5OTc0MTI3MTczMDk3NzM0NjExNzM5MDAxODQ4NDQ2NjE0NTI1MTc4NTg2MDE5MjI1OTY3NjkzMjQxODAxOTc4MzcyMDYxNTIxNjc1Njc0MjgyMzM0O
TQyNzM1OTc3MjAwNzQwMjU3ODYwNDI3MjI5ODc5OTE1MDcxMjA5ODEyNTk0NjQ2ODIwNTQzNzA1ODMyMDE5NzE5OTA2OTE2NDcxNjIwNTQxOTMwMjcyMzk1ODc0NTc4MzA
yMTYyNDk2NjA4MjQ5MDEyMjA1MDAyMTUzMzQ1NTU3ODg5Nzg1NDAxODM4MzY1NDkxNjczMTM0MzY=
[+] Welcome vhm0013!

An improved signing server with extra security features such as hashing usernames to avoid forging tokens!
Available options:
```

Option [2] Solving Modular Equations

- When selecting option [2] the server provides modular equations:

```
[+] Option >> 2

To avoid disclosing public keys to bots, a modern captcha must be completed. Kindly compute the hash of 'N' to get the full public
 key based on the following equations:
['equation(unknown, 2362569089, 2654435761) = 1778835745', 'equation(unknown, 2275542397, 2654435761) = 2182213', 'equation(unknow
n, 2228596429, 2654435761) = 1403891972', 'equation(unknown, 2256262699, 2654435761) = 682790597']
```

- Steps we follow to Solve the equations:
  - equation(unknown, rnd, golden_ratio) = output1
  - Identified the Golden Ratio = 2654435761
  - We compute the inverse of rnd
  - Solve the unknown (prime factor of N's hash)
  - Multiply the derived factors to compute the hash of N
  - Submit the hash to the server to retrieve the public key components (e,N)

```
[+] Option >> 2

To avoid disclosing public keys to bots, a modern captcha must be completed. Kindly compute the hash of 'N' to get the full public
 key based on the following equations:
['equation(unknown, 2362569089, 2654435761) = 1778835745', 'equation(unknown, 2275542397, 2654435761) = 2182213', 'equation(unknow
n, 2228596429, 2654435761) = 1403891972', 'equation(unknown, 2256262699, 2654435761) = 682790597']

Enter the hash(N): 1438347351
[+] Captcha successful!
(e,N) = (65537, 230777806430966263458345699722877622611029490904158327457419890723619263486797877529270395607582938097445094432517
295438396691047299722050515259763726907035764556373587411453003311218447660680400204936659052878208606363311053640825623958676932 3
428563507924328294845267853624211220952372099936321107894322768095328083392138550561493769947824345868819593734227888402739738370 5
154107650635332575833213926056849565724859597382538209907573708690111649587272270028694724390380914808269038116695624017391802295 1
424585606493983773108210012849749947204899358322089025689210588390099915292560008455060564488746936586743094059 73)
```

Cocalc, Colab

HTB challenge.sagews

Save | TimeTravel | Run | Stop | Restart | in | out

Help | Modes | # | Data | Control | Program | x | Plots | Calculus

```python
# Define your values
rnd_values = [2449658621, 2594049421, 2549842063, 2510866091, 2240665303]
results = [1835327201, 2231731381, 1388090544, 1706804700, 2190175457]

# Find GCD and divide moduli
gcd = gcd(rnd_values)
new_rnd_values = [r // gcd for r in rnd_values]

# Compute the solution with CRT
hash_n = crt(results, new_rnd_values)
print("Computed hash(N):", hash_n)
```

```
Computed hash(N): 42517206027217179701765029168175927241744473227
```

HTB challenge.sagews | Challenge.sagews

File | Save | TimeTravel | Run | Stop | Restart | in | out

Help | Modes | # | Data | Control | Program | x | Plots | Calculus | Line

```python
# Define the public key parameters
N = 42517206027217179701765029168175927241744473227
e = 835878 * 3473

# Define the admin username and convert to an integer representation
username = "System_Administrator"
message_hash = int(username.encode().hex(), 16)  # Convert to hexadecimal integer

# Simulate RSA signature using power_mod for modular exponentiation
signature = power_mod(message_hash, e, N)
print("Signature:", signature)
```

```
Signature: 10351858794841875914213769405168059448123058015
```

```python
from sympy import mod_inverse

golden_ratio = 2654435761
equations = [
    {"rnd": 2584448683, "output": 1356042644},
    {"rnd": 2378535373, "output": 1939146287},
]

# Solve modular equations
unknowns = []
for eq in equations:
    rnd = eq["rnd"]
    output = eq["output"]

    # Solve for x such that (x * rnd) % golden_ratio = output
    try:
        rnd_inv = mod_inverse(rnd, golden_ratio)  # Compute modular inverse of rnd
        x = (output * rnd_inv) % golden_ratio  # Solve for x
        unknowns.append(x)
    except ValueError:
        print(f"Cannot compute modular inverse for rnd = {rnd} mod {golden_ratio}")

print(f"Derived unknowns: {unknowns}")

# Combine unknowns to compute N
N = 1
for val in unknowns:
    N *= val

print(f"Derived N: {N}")
```

```
Derived unknowns: [1877, 854443]
Derived N: 1603789511
```

Exploitation Attempts

- Brute Force N
  - Brute force the Modules N used by RSA
  - Focused on small ranges of N change the ranges
- Hash collisions
  - Searched username the hashes same values as System Admin
- Replay Attacks
  - Reused valid tokens from accounts with System Administrator username
  - Modified tokens incrementing numerical values

```python
golden_ratio = 2654435761
hash_var = lambda key: (((key % golden_ratio) * golden_ratio) >> 32)

# Admin username
admin_username = int.from_bytes(b"System_Administrator", "big")
admin_hash = hash_var(admin_username)

print(f"Admin hash (message to factorize): {admin_hash}")
```

```
Admin hash (message to factorize): 1115247629
```

Exploitation

```
[ ] admin_username = int.from_bytes(b"System_Administrator", "big")
    hashed_message_admin = ((admin_username % golden_ratio) * golden_ratio) >> 32
    print(f"Hashed message for 'System_Administrator': {hashed_message_admin}")

Hashed message for 'System_Administrator': 1115247629

[ ] from sympy import mod_inverse
    forged_token_value = mod_inverse(hashed_message_admin, 2834010038537001217344092772246082888651937988258340954639051060602563823674522358376436896240990323804135851591364083121283482061004723017853950969464614813096
    print(f"Forged token value: {forged_token_value}")

Forged token value: 1344361762321981015227165674605525641316253737861170124631857045907053597219557158562385600419811487775895510702847937765019322667423153774184536987633385372691859945890167362402374333642182908559

[ ] forged_token = b64encode(str(forged_token_value).encode()).decode()
    print(f"Base64 Encoded Forged Token for System_Administrator: {forged_token}")

Base64 Encoded Forged Token for System_Administrator: MTM0NDM2MTc2MjMyMTk4MTAxNTIyNzE2NTY3NDYwNTUyNTY0MTMxNjI1MzczNzg2MTE3MDEyNDYzMTg1NzA0NTkwNzA1MzU5NzIxOTU1NzE1ODU2MjM4NTYwMDQxOTgxMTQ4Nzc3NTg5NTUxMDcwMjg0NzkzNzc2N
```

```
# Compute hashed message for System_Administrator
golden_ratio = 2654435761
admin_username = int.from_bytes(b"System_Administrator", "big")
hashed_message_admin = ((admin_username % golden_ratio) * golden_ratio) >> 32

print(f"Hashed message for System_Administrator: {hashed_message_admin}")

# Check coprimality
if gcd(hashed_message_admin, N) == 1:
    print("Hashed message and N are coprime.")
else:
    print("Hashed message and N are NOT coprime.")

# Forge token if coprime
try:
    S = mod_inverse(hashed_message_admin, N)
    print(f"Forged token value (S): {S}")

    # Verify forged token
    verified_value = pow(S, e, N)
    print(f"Verification result: {verified_value}")
    if verified_value == hashed_message_admin:
        print("Verification successful.")
    else:
        print("Verification failed.")

    # Encode the token
    forged_token = b64encode(str(S).encode()).decode()
    print(f"Base64 Encoded Forged Token for System_Administrator: {forged_token}")
except ValueError:
    print("Failed to compute modular inverse. Ensure hashed_message_admin and N are coprime.")
```

```
Hashed message for System_Administrator: 1115247629
Hashed message and N are coprime.
Forged token value (S): 2472152909375804308649457158105893825513584915863856207638541786623154148211891338323354760047657379618236597783772325053601790782173679918173420031748956149604211308870419524026065533382259
Verification result: 6850851515427508136365608543308522152965038443903842123409468781340483465394845144273508040654416750175155317186395460803634301115216233631441842314155423181845693089367872962723547360233198985134
Verification failed.
Base64 Encoded Forged Token for System_Administrator: MjQ1MjE1MjkwOTMxMTAzODEwNDk1NDk1NDc1NjYyMzUzNTYxMzgyNTUwMzUwOTI1OEgzMCA1OTIwMzY2ODkwMTc0NjYyMzUzNTYxMDIzOTU5NjcxMzc0MjMyNTg1M
```

Challenges Faced

Challenges

Connecting to OpenVPN

Issue: Difficulty connecting to the OpenVPN.
Cause: Many ports were blocked.

Port Scanning

Issue: Limited success in scanning due to blocked ports.
Impact: Increased difficulty in identifying open ports and services.

Challenges

Wrong Direction

Issue: Wasted time attempting to crack SSH.
Impact: Delay in progressing towards the actual solution.

Cryptography

Issue: Difficulty in cracking the cipher and hashing mechanism.
Solution: Utilized scripts and tools to eventually break through.

What We Learned

Learning

Hashing Mechanism

Cryptographic Analysis: Gained experience in analyzing and breaking cryptographic mechanisms.
Tool Utilization: Leveraged scripts and tools to automate and enhance the cracking process.

Cracking the Signing Mechanism

Algorithm Understanding: Deepened knowledge of hashing algorithms and their weaknesses.

Practical Application: Applied theoretical knowledge to a real-world challenge, reinforcing learning through practical application.

Learning

Hashing Mechanism

Cryptographic Analysis: Gained experience in analyzing and breaking cryptographic mechanisms.
Tool Utilization: Leveraged scripts and tools to automate and enhance the cracking process.

Cracking the Signing Mechanism

Algorithm Understanding: Deepened knowledge of hashing algorithms and their weaknesses.

Practical Application: Applied theoretical knowledge to a real-world challenge, reinforcing learning through practical application.

Multi Layered Security Protection

1 Website Protection?

We are using Two-Factor Authentication and SSL/TLS Security and honeypots

2 Domain Protection?

We are making using of CDNs and Web Application Firewalls

3 Server Protection?

We are using Azure Sentinel to protect our VPS

Mitigation

Upgrade HTTP version or Deprecate HTTP/0.9 support

- Transition to HTTP/ 1.1 or HTTP/2 with stronger security features and support for TLS encryption; implement HTTPS with TLS
- Disable legacy protocols in Apache's configuration files:

```
<IfModule mod_rewrite.c>
        RewriteEngine On
        RewriteCond %{THE_REQUEST} !^HTTP/1\.[01]$
       RewriteRule .* - [F,L]
</IfModule>
```

Use web application firewalls

- Deploy a WAF to detect and block token manipulation or injection attempts.
- Configure rules specific to legacy HTTP requests if HTTP/0.9 is unavoidable.

Mitigation Con't

Validate Tokens on server side

- Validate tokens for integrity and authenticity using:
  - HMAC (hash-based message authentication code)
    - Used to digitally sign JWTs

```
hex_sha256(secret + base64header + "." + base64payload)
```

  - JWT (JSON web tokens)
    - RSA public/private key pairs

```
// Sign with private key
sign(base64header + "." + base64payload, privatekey)

// Verify with public key
verify(base64header + "." + base64payload, publickey, signature)
```