Breaking "**PERFECT**" Crypto w/ Timing Attacks

grhkm

$ whoami

- 👎 bad crypto CTF player
- ~~👍 good pwn CTF player~~
- 2nd yr math major
- cool blog https://grhkm21.github.io/

solo

Place	Event
15	IrisCTF 2023
75	Real World CTF 5th

team

Overall rating place: **14** with **332.341** pts in 2023

Country place: 1

Place	Event
5	ångstromCTF 2023
13	PlaidCTF 2023
9	Cyber Apocalypse 2023: The Cursed Mission
18	b01lers CTF

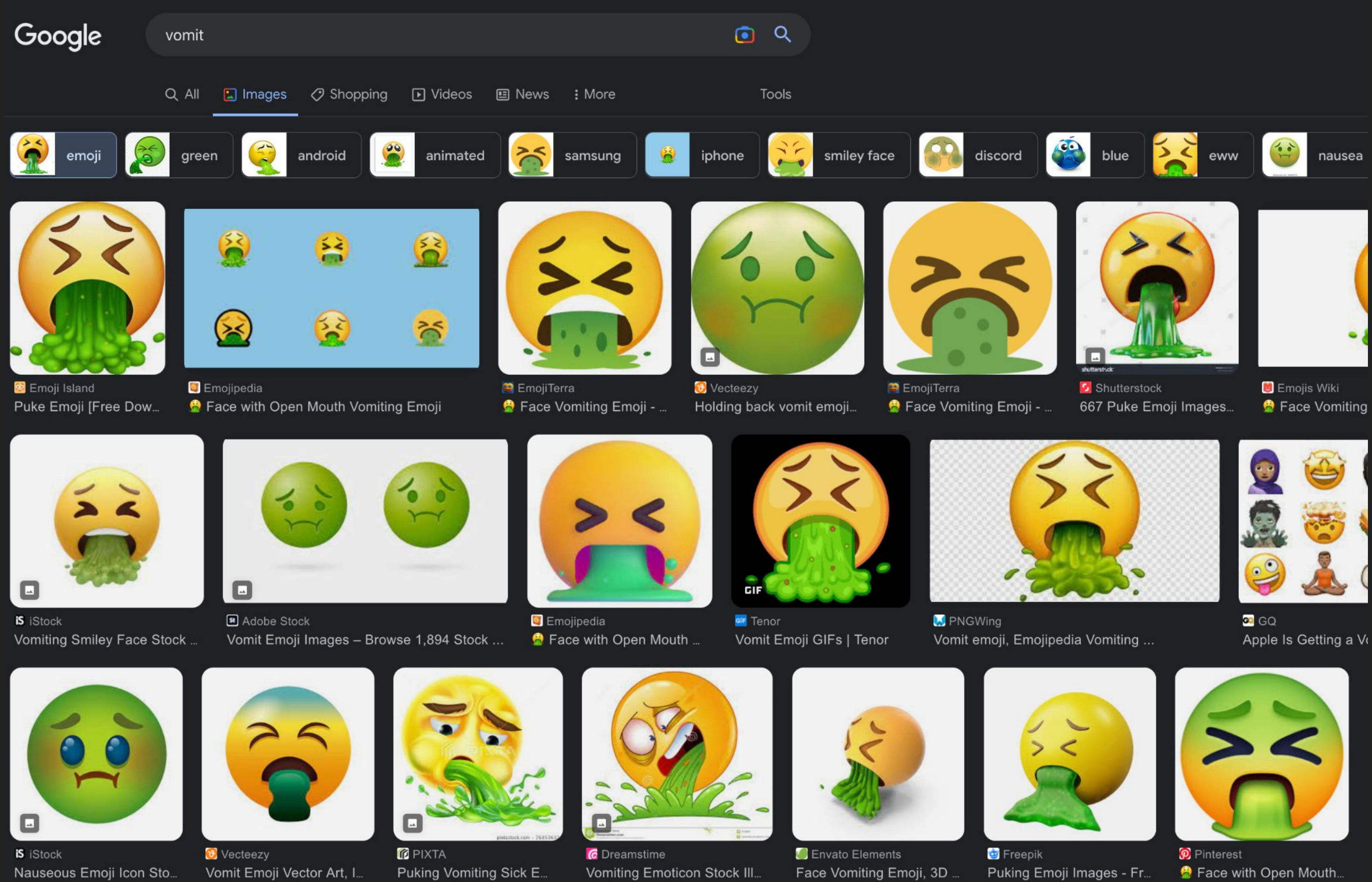
28	hxp CTF 2022
11	pbctf 2023
4	LA CTF 2023

24	DiceCTF 2023
7	idekCTF 2022*
8	TetCTF 2023

What is crypto?

What is crypto?

- This...?


Google
vomit
All
Images
Shopping
Videos
News
More
Tools
emoji
green
android
animated
samsung
iphone
smiley face
discord
blue
eww
nausea
Emoji Island
Puke Emoji [Free Dow...
Emojipedia
Face with Open Mouth Vomiting Emoji
EmojiTerra
Face Vomiting Emoji - ...
Vecteezy
Holding back vomit emoji...
EmojiTerra
Face Vomiting Emoji - ...
Shutterstock
667 Puke Emoji Images...
Emojis Wiki
Face Vomiting
iStock
Vomiting Smiley Face Stock ...
Adobe Stock
Vomit Emoji Images – Browse 1,894 Stock ...
Emojipedia
Face with Open Mouth ...
GIF
Tenor
Vomit Emoji GIFs | Tenor
PNGWing
Vomit emoji, Emojipedia Vomiting ...
GQ
Apple Is Getting a Vo
iStock
Nauseous Emoji Icon Sto...
Vecteezy
Vomit Emoji Vector Art, I...
PIXTA
Puking Vomiting Sick E...
Dreamstime
Vomiting Emoticon Stock Ill...
Envato Elements
Face Vomiting Emoji, 3D ...
Freepik
Puking Emoji Images - Fr...
Pinterest
Face with Open Mouth...

What is crypto?

- While that's *an* interpretation of "crypto" 😫
- We will talk about another type 😍

Cryptography

- Bonus: At the end, we discuss how your Bitcoin wallets might be hacked!

Part I: Perfect code

Unbreakable code 🛡️

Goal: Recover PASSWORD… Seems impossible.

```python
from secrets import PASSWORD, PASSWORD_LEN


def check_password(s):
    if len(s) != PASSWORD_LEN:
        return False

    for i in range(PASSWORD_LEN):
        if s[i] != PASSWORD[i]:
            return False

    return True
```

Unbreakable code 🛡️

Goal: Recover PASSWORD… Seems impossible.

```python
from secrets import PASSWORD, PASSWORD_LEN
```

After reviewing the code, I can confirm that there are no obvious vulnerabilities in this code, assuming that `PASSWORD` and `PASSWORD_LEN` are generated using a secure random number generator and kept secret.

```python
    for i in range(PASSWORD_LEN):
        if s[i] != PASSWORD[i]:
            return False

    return True
```

Unbreakable code 🛡️👀

- Let’s look closer at the code.

```python
def check_password(s):
    if len(s) != PASSWORD_LEN:
        return False

    for i in range(PASSWORD_LEN):
        if s[i] != PASSWORD[i]:
            return False

    return True
```

Unbreakable code 🛡️👀

- Let's look closer at the code.
- Say PASSWORD = "UWCS", and we tried s = "UAAA".
- The code will check the following:
  - Is s[0] = PASSWORD[0]? YES.
  - Is s[1] = PASSWORD[1]? NO.
  - EXIT.

```python
def check_password(s):
    if len(s) != PASSWORD_LEN:
        return False

    for i in range(PASSWORD_LEN):
        if s[i] != PASSWORD[i]:
            return False

    return True
```

Unbreakable code 🛡️👀

- Let's look closer at the code.
- Say PASSWORD = "UWCS",

  and we tried s = "UWAA".
- The code will check the following:
  - Is s[0] = PASSWORD[0]? YES.
  - Is s[1] = PASSWORD[1]? YES.
  - Is s[2] = PASSWORD[2]? NO.
  - EXIT.

```python
def check_password(s):
    if len(s) != PASSWORD_LEN:
        return False

    for i in range(PASSWORD_LEN):
        if s[i] != PASSWORD[i]:
            return False

    return True
```

Unbreakable code 🛡️👀

- Did you notice?

"UAAA" "UWAA"

YES; NO YES; YES; NO

```
def check_password(s):
    if len(s) != PASSWORD_LEN:
        return False

    for i in range(PASSWORD_LEN):
        if s[i] != PASSWORD[i]:
            return False

    return True
```

=> The second example takes <u>more time</u> to execute!

Part II: Timing attack

~~Un~~Breakable code 💀

- Guess “AAAA” -> 30.7ms
- Guess “BAAA” -> 30.3ms
- …
- Guess “TAAA” -> 29.9ms
- Guess “UAAA” -> 47.3ms 👍
- Guess “UBAA” -> 47.8ms
- …
- Guess ”UVAA” -> 46.9ms
- Guess ”UWAA” -> 61.2ms 👍
- …

- Code is completely BROKEN.

```python
def check_password(s):
    if len(s) != PASSWORD_LEN:
        return False

    for i in range(PASSWORD_LEN):
        if s[i] != PASSWORD[i]:
            return False

    return True
```

- Attack idea:
  - We guess passwords one by one.
  - If the new guess matches more characters, it will take longer.
  - The attacker can measure the time taken for the server to execute check_password.

Analysis

- The attack was made possible because the code exits early.
- In other words, the algorithm is input-dependent.
- This allows the attacker to gain information from timing.

```python
def check_password(s):
    if len(s) != PASSWORD_LEN:
        return False

    for i in range(PASSWORD_LEN):
        if s[i] != PASSWORD[i]:
            return False

    return True
```

Mitigation

- Do not exit early.
- Remove branches if possible.
- More generally, aim for constant-time code.

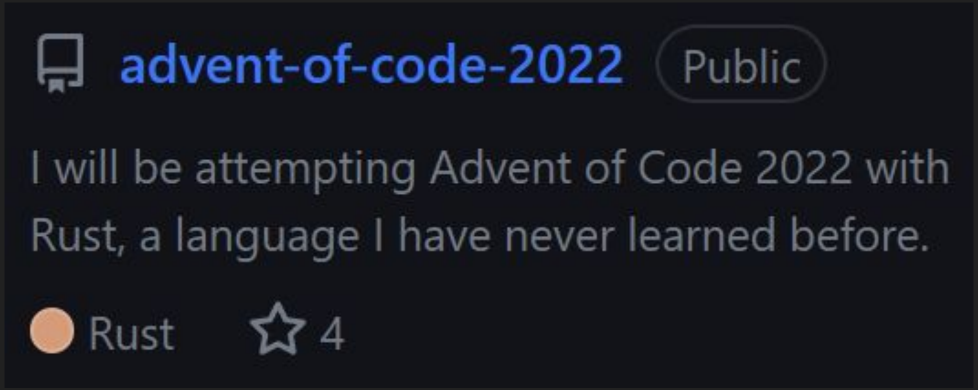
```python
def check_password(s):
    if len(s) != PASSWORD_LEN:
        return False

    for i in range(PASSWORD_LEN):
        if s[i] != PASSWORD[i]:
            return False

    return True
```

```python
def check_password_safe(s):
    failed = len(s) != PASSWORD_LEN

    for i in range(PASSWORD_LEN):
        failed |= s[i] == PASSWORD[i]

    return not failed
```

Case Study #1: Symfony UriSigner

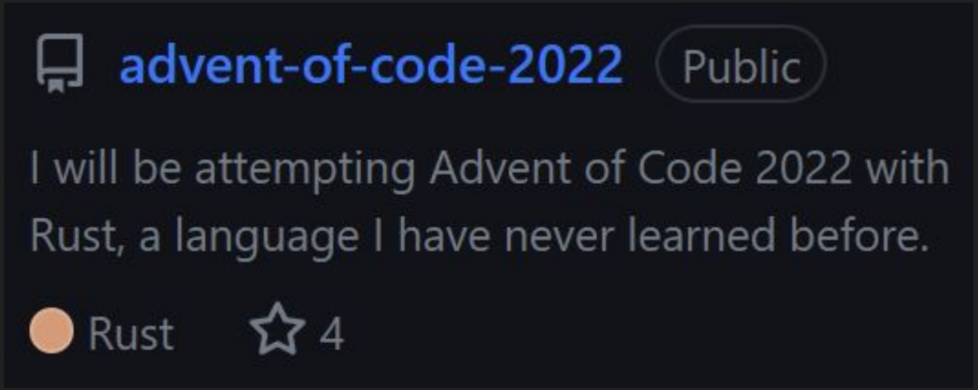
Symfony

- A **very** popular PHP framework for web applications.



Symfony

- A **very** popular PHP framework for web applications.



Symfony Code review 👏👏

- The **`UriSigner`** class performs checks on the built URI.
- In the end, a hash comparison is performed.
- === is used, which is not constant-time.
- Critical vulnerability since **`UriSigner`** lies in the **`http-kernel`** module.

```
class UriSigner
    public function check($uri)

        return $this->computeHash($this->buildUrl($url, $params)) === $hash;
    }
```

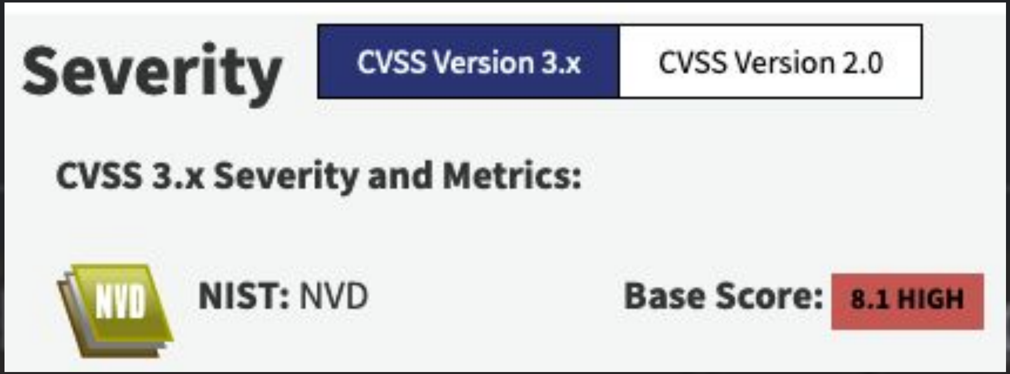
Why is === not constant-time?

- Digging through PHP's source code, === is implemented in C.
- The C function **`strcmp`** is used.
- We can look at the source code in GCC:

```c
int
memcmp (const void *str1, const void *str2, size_t count)
{
  register const unsigned char *s1 = (const unsigned char*)str1;
  register const unsigned char *s2 = (const unsigned char*)str2;

  while (count-- > 0)
    {
      if (*s1++ != *s2++)
          return s1[-1] < s2[-1] ? -1 : 1;
    }
  return 0;
}
```

Symfony Code review 👏👏

- Vulnerability reported to the National Vulnerability Database
- Assigned as CVE-2019-18887.
- **Severity: 8.1 / 10**



Fixing Symfony’s code 👏👏

- Solution: Use constant-time string comparison.
- PHP docs mentions `hash_equals`.

hash_equals

(PHP 5 >= 5.6.0, PHP 7, PHP 8)

hash_equals — Timing attack safe string comparison

```c
    if (ZSTR_LEN(a) != ZSTR_LEN(b)) {
        return -1;
    }

    /* This is security sensitive code. Do not op
    while (i < ZSTR_LEN(a)) {
        r |= ua[i] ^ ub[i];
        ++i;
    }
```

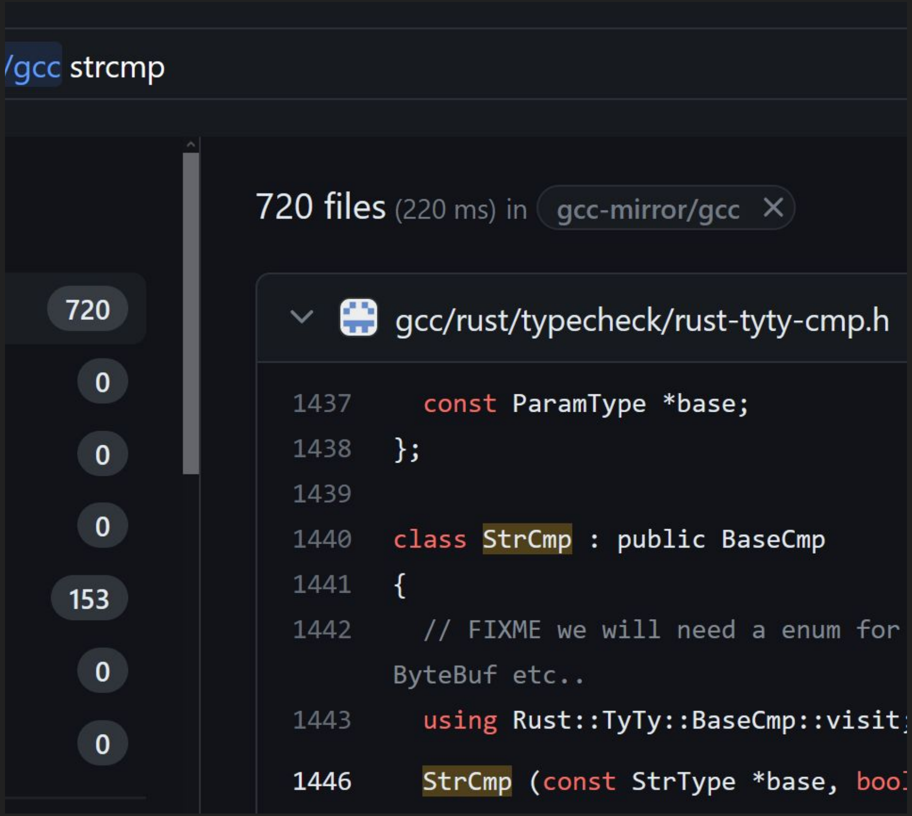
Implementation of `hash_equals`

No early return!

```php
class UriSigner
    public function check(string $uri): bool
        return hash_equals($this->computeHash($this->buildUrl($url, $params)), $hash);
    }
```

Vulnerable code! 😱

- Wait… if GCC's **memcmp** is unsafe
- Does that mean all codes using it are unsafe?
- Or all codes using === are unsafe?
- That's a lot of them…



Vulnerable code? 😮‍💨

- Wait… if GCC's **`memcmp`** is unsafe
- Does that mean all codes using it are unsafe?
- Or all codes using === are unsafe?
- That's a lot of them… No. Most of them are safe.
- There are many other factors:
    - Attack surface
    - Which parameters the attacker can control
    - …

Where is crypto?

Case Study #2: RSA Decryption

What is RSA? 🤔

- Standard encryption protocol.
- Commonly used in real world cryptography.

```
Your identification has been saved
Your public key has been saved in /
The key fingerprint is:
SHA256:JGWm/DZpoiH+QknFhcji7ssTKtez
The key's randomart image is:
+---[RSA 3072]----+
| . o o. +        |
|. o +. =         |
|.. .  + .       .|
| ..    + .     o |
|..... . S     o  |
| +o. o + . . o . |
|o.o..     . . =  |
|+oo.o .  .   ..*.|
|.+o.o+ ....  .o=E|
+----[SHA256]-----+
```

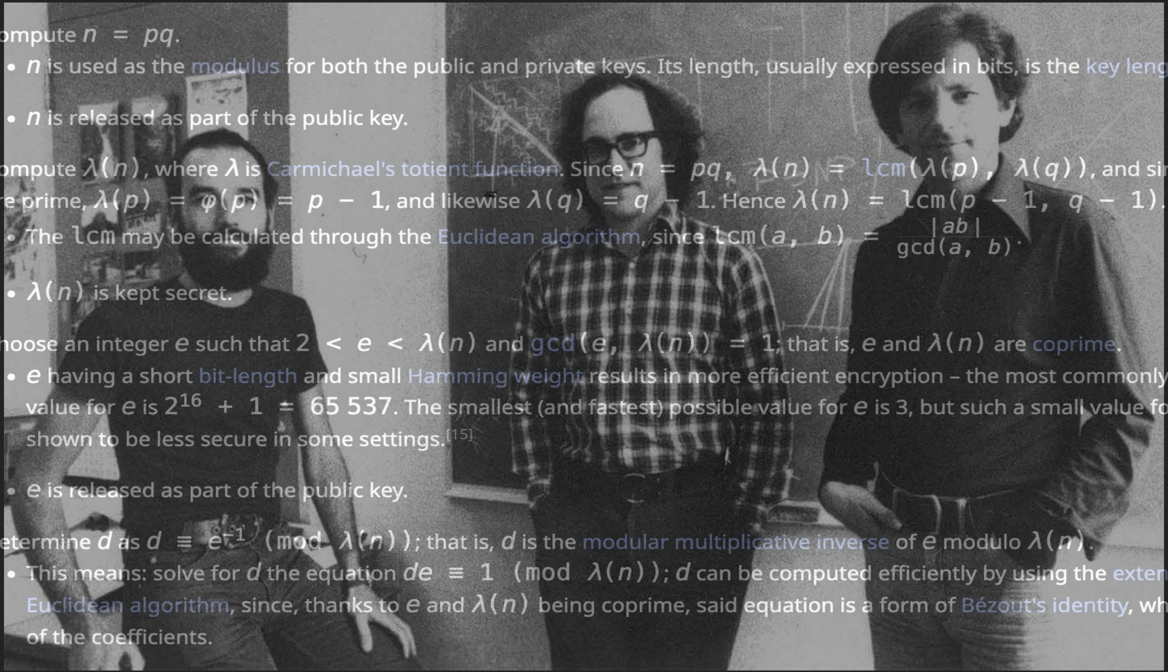
What is RSA? 🤔

RSA

Key generation

The keys for the RSA algorithm are generated in the following way:

1. Choose two large prime numbers p and q.
   - To make factoring harder, p and q should be chosen at random, be both large and have a large difference.[1] For choosing them the standard method is to choose random integers and use a primality test until two primes are found.
   - p and q should be kept secret.



2. Compute $n = pq$.
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
   - n is released as part of the public key.
3. Compute $\lambda(n)$, where λ is Carmichael's totient function. Since $n = pq$, $\lambda(n) = \operatorname{lcm}(\lambda(p), \lambda(q))$, and since p and q are prime, $\lambda(p) = \varphi(p) = p - 1$, and likewise $\lambda(q) = q - 1$. Hence $\lambda(n) = \operatorname{lcm}(p - 1, q - 1)$.
   - The lcm may be calculated through the Euclidean algorithm, since $\operatorname{lcm}(a, b) = \frac{|ab|}{\gcd(a, b)}$.
   - $\lambda(n)$ is kept secret.
4. Choose an integer e such that $2 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; that is, e and $\lambda(n)$ are coprime.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – the most commonly chosen value for e is $2^{16} + 1 = 65\,537$. The smallest (and fastest) possible value for e is 3, but such a small value for e has been shown to be less secure in some settings.[15]
   - e is released as part of the public key.
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; that is, d is the modular multiplicative inverse of e modulo $\lambda(n)$.
   - This means: solve for d the equation $de \equiv 1 \pmod{\lambda(n)}$; d can be computed efficiently by using the extended Euclidean algorithm, since, thanks to e and $\lambda(n)$ being coprime, said equation is a form of Bézout's identity, where d is one of the coefficients.
   - d is kept secret as the *private key exponent*.

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the private (or decryption) exponent d, which must be kept secret. p, q, and $\lambda(n)$ must also be kept secret because they can be used to calculate d. In fact, they can all be discarded after d has been computed.[16]

Bottom Text

Code inspection 👀

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

d is private
n is public

- Very simple maths! Shown on the right.
- Implementation: `m = pow(c, d, n)`
- Code is run in e.g. connection signature verification
- Let's look at its source code.

Code inspection 👀

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

d is private
n is public

- Very simple maths! Shown on the right.
- Implementation: `m = pow(c, d, n)`
- Code is run in e.g. connection signature verification
- Let's look at its source code.
- … wait, but it's **built-in**!

```
In [1]: import inspect

In [2]: x = 13

In [3]: inspect.getsource(x.__pow__)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [3], in <cell line: 1>()
----> 1 inspect.getsource(x.__pow__)
```

Code inspection 👀

- `m = pow(c, d, n)`
- Let's look at its C source code.
- Code taken from CPython 3.11.

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

d is private, n is public

```
for (--i, bit >>= 1;;) {
    for (; bit != 0; bit >>= 1) {
        MULT(z, z, z);
        if (bi & bit) {
            MULT(z, a, z);
        }
    }
    if (--i < 0) {
        break;
    }
    bi = b->ob_digit[i];
    bit = (digit)1 << (PyLong_SHIFT-1);
}
```

Code inspection 👀

- `m = pow(c, d, n)`
- Let's look at its C source code.
- Code taken from CPython 3.11.

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

d is private, n is public

```
for (--i, bit >>= 1;;) {
    [redacted]
        if (bi & bit) {
            MULT(z, a, z);
        }
    [redacted]
}
```

Code inspection 👀

- `m = pow(c, d, n)`
- Let's look at its C source code.
- Code taken from CPython 3.11.
- Control c such that MULT takes more time depending on bit.

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

d is private, n is public

```
for (--i, bit >>= 1;;) {
    [redacted]
        if (bi & bit) {
            MULT(z, a, z);
        }
    [redacted]
}
```

c ← c * a

Depending on d

Timing attack 🕒 (Technical)

- The attack isn't exactly straightforward. Essentially, the branch is taken whenever **bi & bit** is true, which isn't actually up to the attacker's control. The vulnerability lies in the code within i.e. the MULT call, which essentially computes `z := z * a % c`. This line is faster when `z * a` is small, as that would mean that the modulo operation is not performed. More importantly, `c` is the public key modulus, which is known to the attacker! Therefore with some clever maths, the attacker could craft inputs that will satisfy the "hypothesis" where the kth bit of the secret key exponent is set or not. Then, the RSA decryption time will reveal which hypothesis holds. There is even more technicality due to the time taken for the other steps to potentially fluctuate with different inputs. However, with some careful analysis one can show that it is essentially random. There are even more potential problems due to caching, but those are overcome by sending different numbers satisfying the same hypothesis and performing statistical analysis on the timing information.

```c
/* pow(v, w, x) */
static PyObject *
long_pow(PyObject *v, PyObject *w, PyObject *x)
{
    PyLongObject *a, *b, *c; /* a,b,c = v,w,x */
```

```c
for (--i, bit >>= 1;;) {

        if (bi & bit) {
            MULT(z, a, z);
        }

}
```

Timing attack 🕒

- Exact same problem appear in **Balsn CTF 2021 (dlog)**.
- Check out our team captain’s write-up on the solution!
- Accessible at https://**mystiz.hk**/posts/2021/2021-11-27-balsn-dlog/
- (He is the best in the world at crypto, the exact opposite of me)

Impact

- Generalisable to other cryptography protocols
  - Mainly ECDSA, used in Bitcoin and Ethereum!!!
  - In that case, it suffices to leak the length of the secret (nonce)
- Not uncommon in critical software…

CVE-2020-17478 Detail

Crypt::Perl ECDSA

Base Score: 7.5 HIGH

CVE-2022-4304 Detail

OpenSSL RSA

Base Score: 5.9 MEDIUM

CVE-2023-0361 Detail

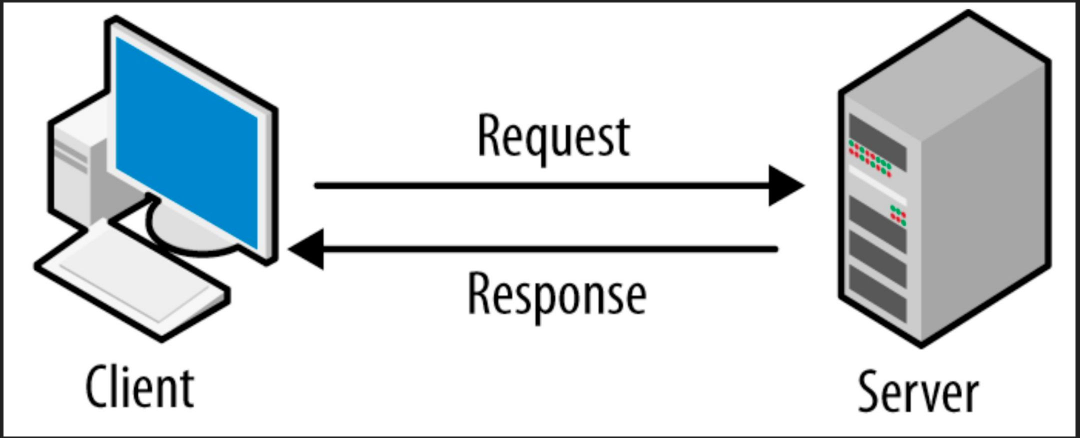
GnuTLS RSA

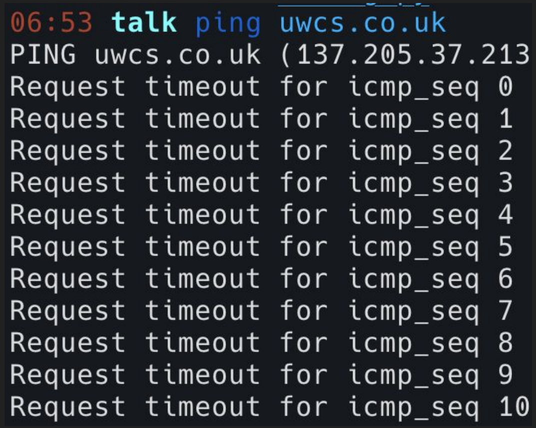
Base Score: 7.5 HIGH

Appendix: Timeless Timing Attacks

Schrödinger’s broken code 💀❓

- There is one final saving grace (for the server) - network delays.
- Our attack relies on receiving accurate timing information.
- What if your server connection is worse than Valorant KR pings? 😭
- Imagine a 30ms±250ms measurement 😰

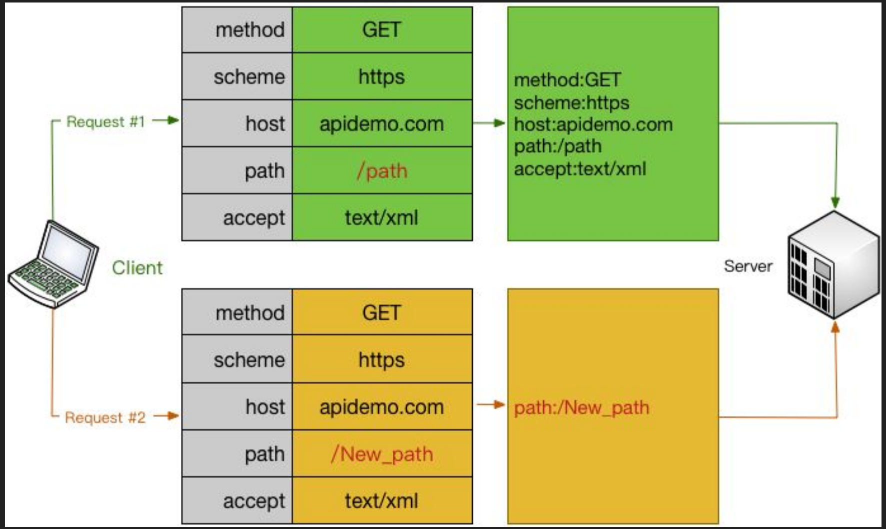




```
06:53 talk ping uwcs.co.uk
PING uwcs.co.uk (137.205.37.213
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
Request timeout for icmp_seq 4
Request timeout for icmp_seq 5
Request timeout for icmp_seq 6
Request timeout for icmp_seq 7
Request timeout for icmp_seq 8
Request timeout for icmp_seq 9
Request timeout for icmp_seq 10
```

“Timeless” 🕒❌ timing 🕒✅ attacks

- Background: HTTP/2.0 includes a new feature called Multiplexing.
- Allows for multiple requests to be sent over a single connection.
- Accelerates page loading speed, better streaming support, reduced latency, …



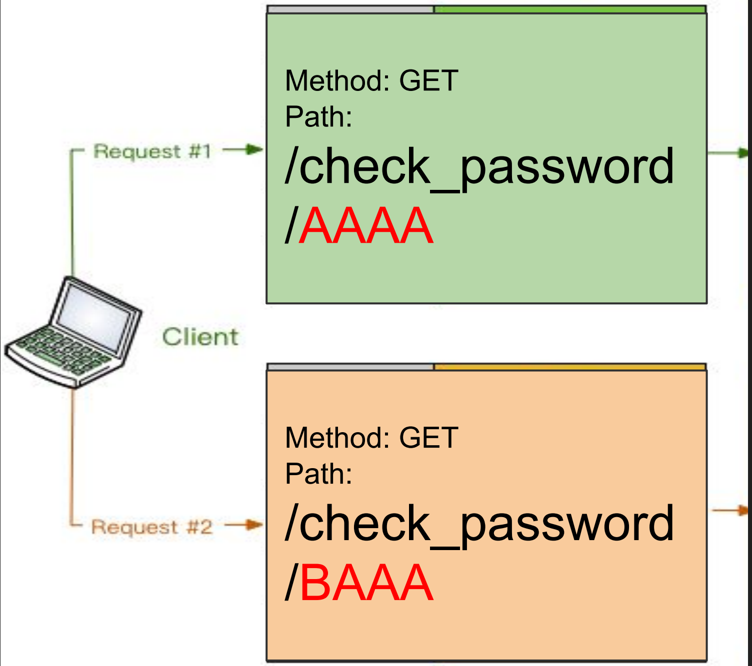
“Timeless” 🕒❌ timing 🕒✅ attacks

- Background: HTTP/2.0 includes a new feature called Multiplexing.
- Allows for multiple requests to be sent over a single connection.
- Accelerates page loading speed, better streaming support, reduced latency, …

- Attack proposed by **Tom Van Goethem, Christina Pöpper, Wouter Joosen, Mathy Vanhoef** in “Timeless Timing Attacks: Exploiting Concurrency to Leak Secrets over Remote Connections” in USENIX 2020.
- Makes use of Multiplexing so that the server receives two requests simultaneously.
- We can then observe the order of the return packets.

“Timeless” 🕒❌ timing 🕒✅ attacks

- We use multiplexing to pack two requests into a single request.
- There will be two responses. The slower one has a chance to be the correct guess.
- We do it for sufficiently many pairs of guesses, and after a while, the correct guess would arrive later the majority of the time.



“Timeless” 🕒❌ timing 🕒✅ attacks

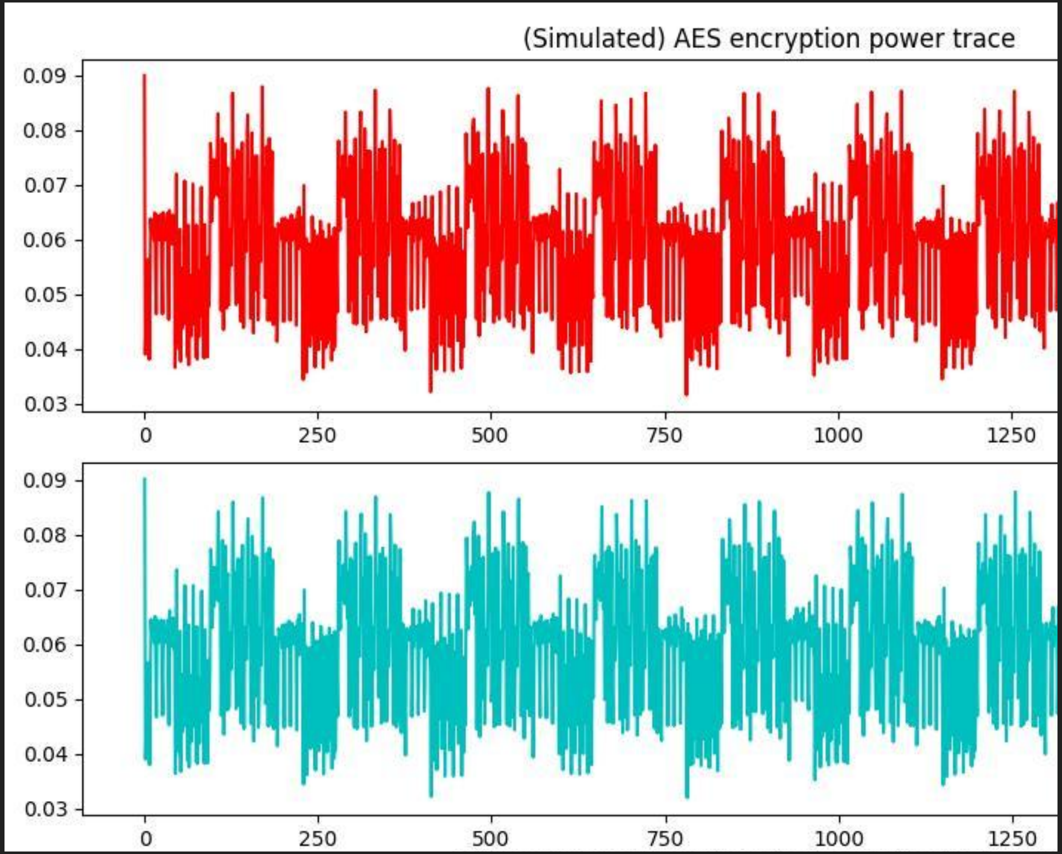
- Exact same problem appear in **WCTF 2020 (Spaceless Spacing)**.
- Check out Connor Nelson’s write-up on the solution!
- Accessible at https://github.com/ConnorNelson/spaceless-spacing
- Detailed explanation with a concise implementation.

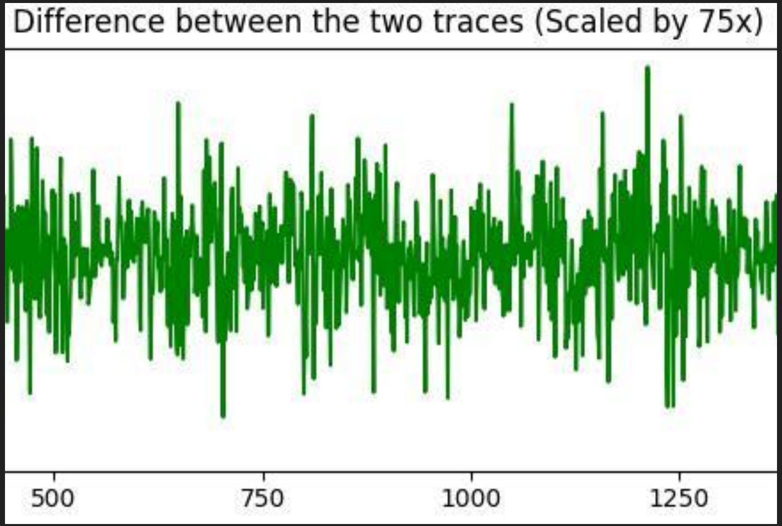
```
elif payload_len <= max_payload_len:
    stream1 = self.send_request(headers1, end_stream=False)
    stream2 = self.send_request(headers2, end_stream=False)
    self.h2conn.send_data(stream1, data1, end_stream=True)
    self.h2conn.send_data(stream2, data2, end_stream=True)
```

Conclusion

- Cryptography is hard.
- Something can be broken by “unconventional methods”.
- One must keep them in mind when implementing critical software.

- Other types of side channel attacks possible - power trace, cache, …





Conclusion?

- Being a software engineer is hard
- Warwick pls teach
- Might lose job
- Even ChatGPT fails

Random keywords to keep you thinking

- “AI is the future” - Joey
    - Applying neural networks for automated attacks?
- C++ / Rust / Python / JS
- LLVM Optimisation
- -O3?
- Vectorisation
- Extending timeless to HTTP/1.1?
- Replaying queries for better accuracy?

Search Results

There are **331** CVE Records that match your search.

Name	
CVE-2023-26557	io.finnet tss-lib before 2.0.0 can leak the lambda example leak is in crypto/paillier/paillier.go. (bnb-
CVE-2023-26556	io.finnet tss-lib before 2.0.0 can leak a secret key loop). One leak is in ecdsa/keygen/round_2.go. (b

tial entries and detec
0, Firefox < 101, an

Flask-AppBuilder is an
enumerate existing a

Thank you

Zulip is an open-source team collaboration tool.
comparator that did not run in constant time. Th
successful, this would allow the attacker to impe

A timing side-channel in the handling of RSA ClientKeyExchange
style attack. To achieve a successful decryption the attacker woul
attacker would be able to decrypt the application data exchanged