Writeup by Bonfee

Chall 3

Description

```
You managed to retrieve the attacker's payload. Your job is now to
understand what it does and to see if it can allow you to penetrate their
infrastructure. Good luck.
```

We are given an ELF binary, compiled in `https://github.com/ponylang/ponyc`

Details

```
1. The infrastructure server has only one CPU core.
2. We are given the libraries to LD_PRELOAD the elf binary
```

Webserver structure

```
root/
├── exploits
│   └── ed53fa7f6e3a109b1756f03baadfe44f2a991ffe08934798ef7ec101a7412399
│       ├──
d0a85535263f9485ff6274be19a73e87d390438d489107133e2d26c1ec7423b3
│       └── infos.txt
└── payload.bin
```

Authentication

All endpoints seem to be authenticated with:
`Kim-Jong-Un:DoYouLikeMyCTF?`
->
`Authorization: Basic S2ltLUpvbmctVW46RG9Zb3VMaWtlTXlDVEY/`

Notes

When called with `--listen`, `payload.bin` will act as a webserver.
When called normally it acts as an `agent`.

When the `agent` starts it will:

1. Make a post request to `/enroll/:hash`, where `:hash` is the sha256 of `infos.txt`, to identify the agent. This post request includes in the body the base64 of `infos.txt`.

2. The agent will look for a folder under `/tmp/0dayz/`, and for each file under that directory it will make a post request to `/upload/:agent_hash/:hash`, where `:hash` is the sha256 of the `serialized file content`. And the body of the post request contains the file content.

Upload file serialization

When the agent uploads a file it serializes the content in a data structure.
For example, if we have `/tmp/0dayz/file1` with this content:

```
$ xxd /tmp/0dayz/file1
00000000: 4242 4242 4242 4242 4141 4141 4141 4141  BBBBBBBBAAAAAAAA
```

The http upload request will contain:

```
$ echo -ne
CwAAAAAAAAAQAAAAAAAAABEAAAAAAAAAIAAAAAAAAABCQkJCQkJCQkFBQUFBQUFBAA== |
base64 -d | xxd
00000000: 0b00 0000 0000 0000 1000 0000 0000 0000  ................
00000010: 1100 0000 0000 0000 2000 0000 0000 0000  ........ .......
00000020: 4242 4242 4242 4242 4141 4141 4141 4141  BBBBBBBBAAAAAAAA
00000030: 00
```

The format

The format seems to be:

```
struct data {
    size_t type_id = 0x0b;
    size_t size    = 0x10;
    size_t alloc   = 0x11;
    size_t b       = 0x20;
    unsigned char data[];
};
```

Info leak

We can leak data from the remote server by sending a corrupted msg and then downloading the file:

```
struct data {
    size_t type_id = 0x0b;
    size_t size    = 0x10000;
    size_t alloc   = 0x08;
    size_t b       = 0x20;
    unsigned char data[] = "AAAAAAAA";
};
```

Deserialization

By sending a message with a modified `type_id` we can make the binary deserialize an arbitrary `pony_type_t`.

```
pony_deserialise()
> pony_deserialise_offset()
> > t = desc_table[id];  // <--- t is pony_type_t*
```

`pony_type_t` contains a couple of function pointers, which will get called shortly after `pony_deserialise_offset` (`recurse()`).

Exploit

- Leak libraries by sending a `0x80` sized message
- Send a big message containing the spray of `[ptr to adjacent pony_type_t] + pony_type_t`. The `pony_type_t` structs contain modified func ptrs (with a stack-pivoting gadget)
- Then send a message with `type_id` equals to the average distance between `desc_table` and the sprayed data we just sent.
- If the spray succeded `t = desc_table[id];` will get one of our fake `pony_type_t`
- Our function pointer gets called, in `rsi` we have a pointer to the content of the message sent (the one with the fake `type_id`)
- Pivot stack to the content of the message, the ropchain `mmap`s a rwx region, `memcpy` the rest of the message and execute shellcode

So the last message is :

```
type_id <--> distance between desc_table and our spray
data:
 > Stack pivot gadget
 > ropchain: RWX mmap + memcpy
 > shellcode: popen() + write() to socket
```

Flag

`HXN{1f329793ed7d4b9b178de07eb257cfed}`