TD 4: Studying an exploit

We will study the description of an old exploit, and explain how it works.
The disclosure was provided by Dan Rosenberg on the *Full Disclosure* mailing list:
`https://seclists.org/fulldisclosure/2010/Dec/85`
It does not provide much explanation. A talk was given by Keegan McAllister on September 19, 2012 to explain in more details what this was all about:
`https://dept-info.labri.fr/~thibault/SecuSys/full-nelson.pdf`
The goal of this TD is to follow the explanations and understand them.

Slide 2

This slide says that it "affects Linux through 2.6.36", and later on it is said that the bulk of the issues (the `econet` driver) were removed in Linux 3.5.

- Find out e.g. from `https://www.kernel.org/pub/linux/kernel/` which dates these Linux versions were released, to get an idea when the exploit got fixed and the bogus driver eradicated.

The slide also says it combines three bugs. That's a very common thing: most often you need to leverage several bugs in order to get an exploit to work. Attackers will keep various bugs in mind, until they determine that a combination of some of them can produce something interesting.

slide 3: `clone()`'s `clear_child_tid`

`clone()` is the real system call behind the `fork()` and `pthread_create()` functions. It allows to control very precisely what the parent and the child process or thread will actually share (address space, files descriptors, pid, etc.)

- In `LANG= man 2 clone` , read about the `CLONE_CHILD_CLEARTID` flag[1].
- This flag is available since Linux 2.5.49. Check what date this was released, to get an idea when the exploit got in.
- The 2.5.x series was however a development series. The code got into production-release with version 2.6.0. Check the date of that release.
- Read the code snippet on slide 3. `mm_release` is called at thread termination, it uses `put_user` to write at the location specified by userland.

[1] The french translation is actually bogus

So far it looks good: `mm_release` takes care of using `put_user` to write the zero, that shouldn't be harmful.

slide 4: Nasty x86 quirks come back

"But sometimes the kernel disables these checks"

`KERNEL_DS` is the segment used by the kernel to access all memory, while `USER_DS` is the segment used by the user to access only the userland memory. At that time, `set_fs` was used to switch between the two for the `get/put_user` calls, by setting the x86 `fs` register[2].

- Read on `https://lwn.net/Articles/722267/` about the nastiness of `set_fs`.
- Read the description part of `man splice` to understand what it does.
- From that, explain why the kernel has to use `set_fs` so that `splice`'s call to `vfs_readv` can work.
- Draw a parallel between the LWN `splice` code snippet and the code shown on slide 4.
- See that the LWN text talks about CVE-2010-4258: if the kernel triggers an oops between the two `set_fs` calls, the second call will never be done! We will discuss that in more details in the next section.
- See in the LWN text that this is a more general problem, and it has been reported that one can return to userland with the second `set_fs` call not done, thus letting userland access all memory!
- See in the LWN text the proposed workaround: making sure that before we return to userland, we check the `fs` value. But then the discussion between security and speed entails... And thus `set_fs` is rather getting completely replaced.

For those interested, `https://lwn.net/Articles/832121/` talks about the actual eventual removal of `set_fs` uses.

slide 5: oops triggers the issue

We have seen last week what an oops looks like: the kernel did a bad thing (e.g. dereferencing a NULL pointer), so it prints a lot of debugging information in `dmesg`, and kills the thread that triggered the oops. Oh, wait...

On thread termination, `clear_child_tid` was supposed to be safely written to through `put_user`, but if an oops happens during e.g. a `splice` call, that write is completely unsafe, and userland can decide whatever address it wants!

- Explain the scenario with your own words (for now don't explain how we trigger the oops and the consequences, that's for next sections).

[2] Nowadays `set_fs` takes a precise address limit instead of setting `KERNEL_DS` / `USER_DS` in `fs`, but the principe is the same.

- This is CVE-2010-4258, read on `https://www.cvedetails.com/cve/CVE-2010-4258/` about it.

slide 6: looking for other bugs

So we have an interesting CVE, let's now look for other bugs that we can leverage.

slide 7: Econet!

As discussed last week, old protocol implementations are bugs nests. And they can get autoloaded...

slide 8: The comment which says it all

This is a good example of a bug that was noticed, but never really dealt with, and happened to have been exploitable... for years!

There is a saying that all bugs can be security bugs, and should thus be considered and acted on accordingly.

slide 9: thus the CVE

- This is CVE-2010-3849, read on `https://www.cvedetails.com/cve/CVE-2010-3849/` about it.
- Look at the source code on `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/net/econet/af_econet.c?id=c39508d6f118308355468314ff414n354`
- See that there is indeed a codepath that tries to deal with `saddr` being NULL.
- Explain why a static analyzer such as coverity would have easily been able to catch this bug.
- Also look at the "fix" `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/net?id=fa0e846494792e722d817b9d3d625a4ef4896c96`

slide 10: splicing the CVE

Let's now really look at the source code.

- Download the source at that time: `https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.36.tar.xz` , unpack it in `/local` (so it's *much* faster to extract than into your network-stored home)
- Run your favorite IDE on it with tags enabled, so as to be able to efficiently navigate between functions.

- The story begins in `fs/splice.c`, line 1706 with the `splice` system call implementation.
- Follow the codepath until the `do_splice_from` call.
- That one uses the `splice_write` method of the file.
- See in `net/socket.c` that for sockets it is `generic_splice_sendpage` that is used.
- See that that function passes to `splice_from_pipe` the `pipe_to_sendpage` actor.
- See that `pipe_to_sendpage` uses the `sendpage` method of the file..
- See in `net/socket.c` what that method is for sockets.
- Follow the codepath until calling the `sendpage` method of the protocol.
- See in `net/econet/af_econet.c` that this is `sock_no_sendpage` for those sockets.
- Follow the codepath until `kernel_sendmsg`, which does the infamous `set_fs` call, then `sock_sendmsg`.
- Follow the codepath until calling the `sendmsg` method of the protocol.
- See what that is for the econet sockets.
- See that we get to the `BUG` comment.

- Make a summarized picture of this succession of calls, showing which layers we have crossed between the VFS, the socket layer, and the actual econet driver.

slide 11: Accessing the econet network

Triggering the load of the econet driver and creating an econet socket is trivial:

```
socket(PF_ECONET, SOCK_DGRAM, 0);
```

But slide 11 says that to reach the `BUG` part one needs an interface with an econet address, which can be done with an `SIOCSIFADDR` `ioctl` call.

- See what in `econet_sendmsg` would indeed prevent us from reaching the `BUG`, and how using `SIOCSIFADDR` fixes that.
- See that `ec_dev_ioctl` and its caller `econet_ioctl` make no permission check at all.
- That's CVE-2010-3850, read `https://www.cvedetails.com/cve/CVE-2010-3850/` about it.

slide 12: All the pieces stick together

- Summarize the scenario so far with a drawing and your own words (for now don't explain the consequences, that's for next sections).

slide 13-14: a 4-byte write is more than plenty

So all in all, through the `clear_child_tid` pointer we can actually write 4 zero-bytes *wherever* we want in the kernel.

This exploit targets the top part of a kernel function pointer, here the `econet_ioctl` method of the econet protocol

- Explain why clearing the top part of a kernel function pointer can then be leveraged to execute userland code with kernel privileges.
- Explain the `target` computation on slide 14.
- Why is the code using `mmap+memcpy` to put a trampoline code somewhere?
- Why will it easy to get the `econet_ioctl` method of the econet protocol called?

slide 15: the actual trampoline

x86_64 likes to use EIP-based relative addressing when calling functions (so that it doesn't have to put a 64bit function pointer into the binary, only a 32bit or even 16bit offset), so we cannot just move the payload code around, whenever it makes calls to other functions, its relative addressing becomes bogus.

- Explain why it becomes bogus.
- Explain what the trampoline code does.

slide 16-17: Preparing the way

So we'll want to call `splice` on an econet socket, so we have to create two file descriptors. Not much to say here.

We also create the thread with the problematic `CLONE_CHILD_CLEARTID` flag.

slide 18-19: trigger the oops and exploit it

On slide 18 we have the code that actually triggers the oops, and on slide 19 the code that exploits it.

- Summarize what happens with a drawing and your own words.

slide 21: pwnd!

Nowadays: would not work

Nowadays, the SMEP feature would prevent such kind of exploit.

- See `https://wiki.osdev.org/Supervisor_Memory_Protection` for a simple description
- Explain how SMEP would prevent this exploit.