Kernel W^X Improvements In OpenBSD

Hackfest 2015

Mike Larkin
mlarkin@openbsd.org
@mlarkin2012

About Myself

- Started hacking on OpenBSD in 2008
  - ACPI
  - S3 (suspend to RAM)
  - S4 (suspend to disk)
  - vmm
- Late last year, I started taking a look at improving W^X in OpenBSD's kernel …

About Myself

- Started hacking on OpenBSD in 2008
  - ACPI
  - S3 (suspend to RAM)
  - S4 (suspend to disk)
  - vmm
- Late last year, I started taking a look at improving W^X in OpenBSD's kernel …
  - It was supposed to be a one month effort...

About Myself

- I'm not a “security guy”
- Improving W^X was an effort in improving *correctness*, not security
- Improve correctness, and sometimes you get security improvements for free

W^X – What Is It?

- W^X is a *memory protection policy*
  - Memory should not be simultaneously writable and executable
- How is that policy *enforced*?
  - The OS drives processor hardware enforcement features
  - Both OS and CPU involved
- Both usermode (eg, processes) and kernel mappings can be protected

W^X And OpenBSD

- OpenBSD has supported W^X in usermode for a long long time
  - More than 15 years
- Implemented with *page table permissions* on hardware architectures that support it
  - R/W/X bits or “R/W and NX bit”

W^X And OpenBSD

- The i386 platform historically did not have hardware “no execute” capability
  - Added later, requires PAE paging and a late-model Pentium 4 or better
- Kernel mode W^X protection in OpenBSD came later

W^X In The OpenBSD Kernel

- In Oct 2014, I was not a W^X hacker
  - Then I casually read this commit:

> /sys/arch/amd64/amd64/pmap.c
>
> revision 1.75
> date: 2014/10/18 17:28:34; author: kettenis; state: Exp; lines +2/-2;
> Make sure the direct map isn't executable on hardware that allows us to do so.
> Enforcing W^X in the kernel like this mitigates at least some ret2dir attacks.

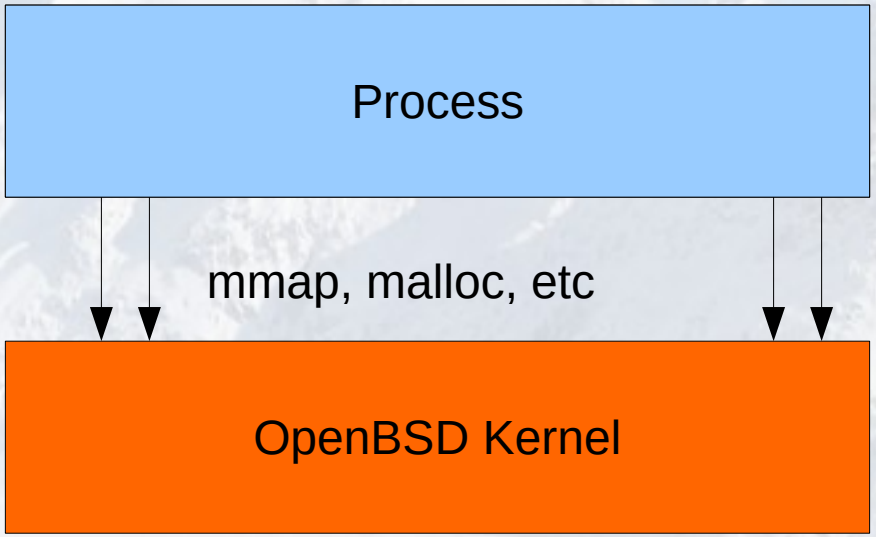
- I then wondered what other areas were not protected

W^X In The OpenBSD Kernel

- Looking at the protection bits in the kernel, I found many areas with incorrect protection
- Slowly, we started fixing things
  - amd64 was more or less done by Jan/Feb 2015
  - i386 . Ugh.

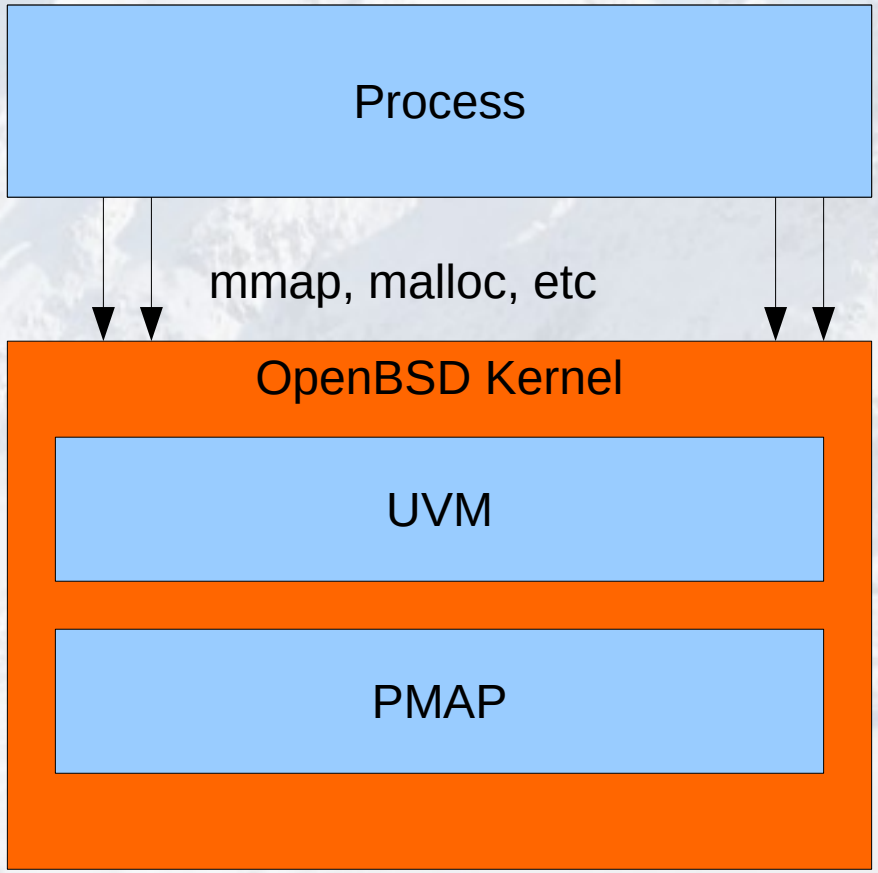
How OpenBSD Manages Memory

- When a process issues a malloc / mmap call ...



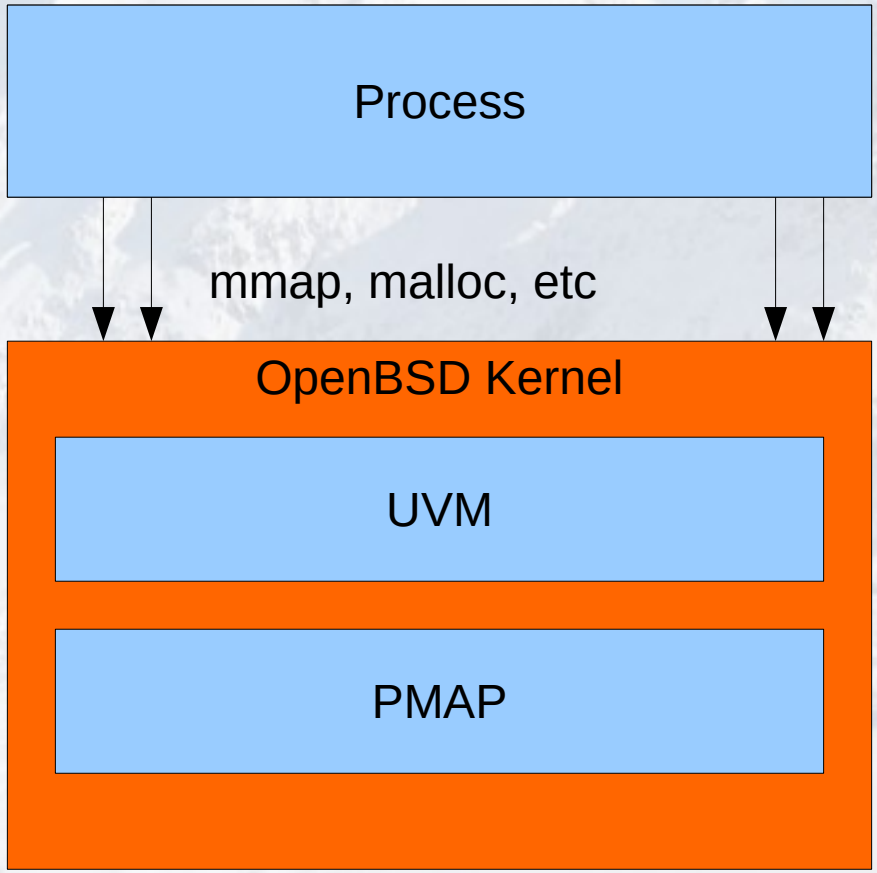
How OpenBSD Manages Memory

- When a process issues a malloc / mmap call ...
- Multiple layers of the OpenBSD kernel cooperate to manage the memory allocated to the process



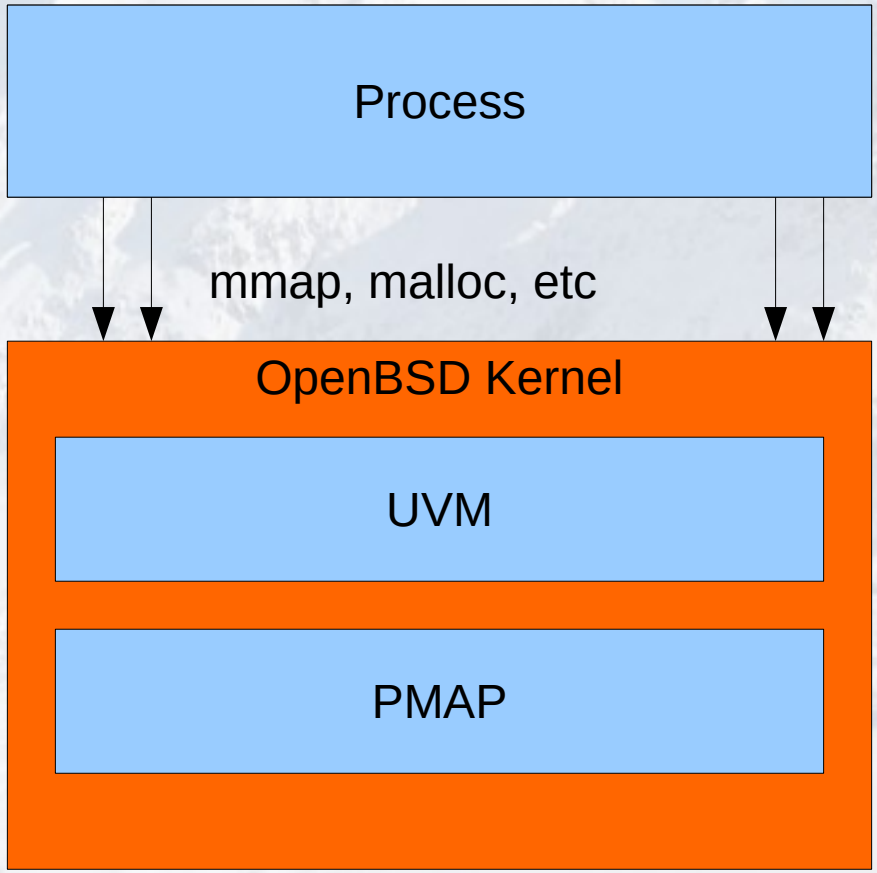
How OpenBSD Manages Memory

- The UVM layer is a machine independent (MI) memory manager
- Handles where memory is allocated, process memory maps, file-backed mmaps, etc.



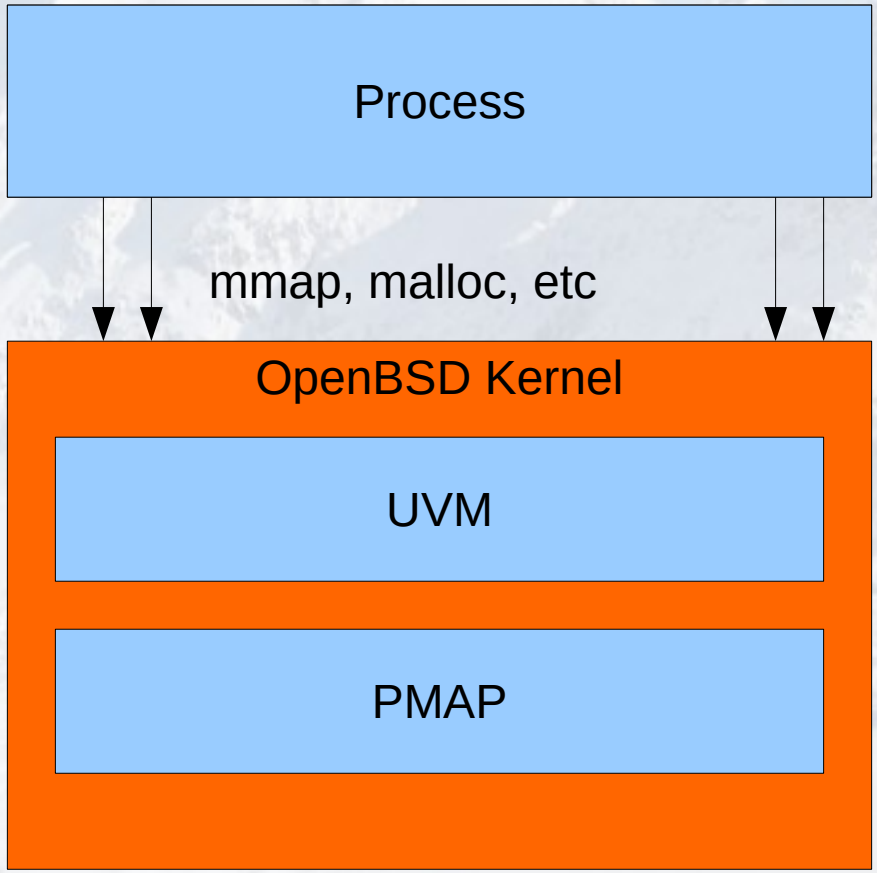
How OpenBSD Manages Memory

- The pmap layer is a machine dependent (MD) module
  - Different for each architecture
- Manages page tables at the hardware level



How OpenBSD Manages Memory

- As a memory protection policy, W^X is enforced at ***both*** layers in OpenBSD
  - UVM won't let you ask for W and X
  - pmap always encodes proper permissions



How OpenBSD Manages Memory

- For example, in /sys/uvm/uvm_map.c:

```
if (map == kernel_map &&
    (prot & (PROT_WRITE | PROT_EXEC)) == (PROT_WRITE | PROT_EXEC))
        panic("uvm_map: kernel map W^X violation requested");
```

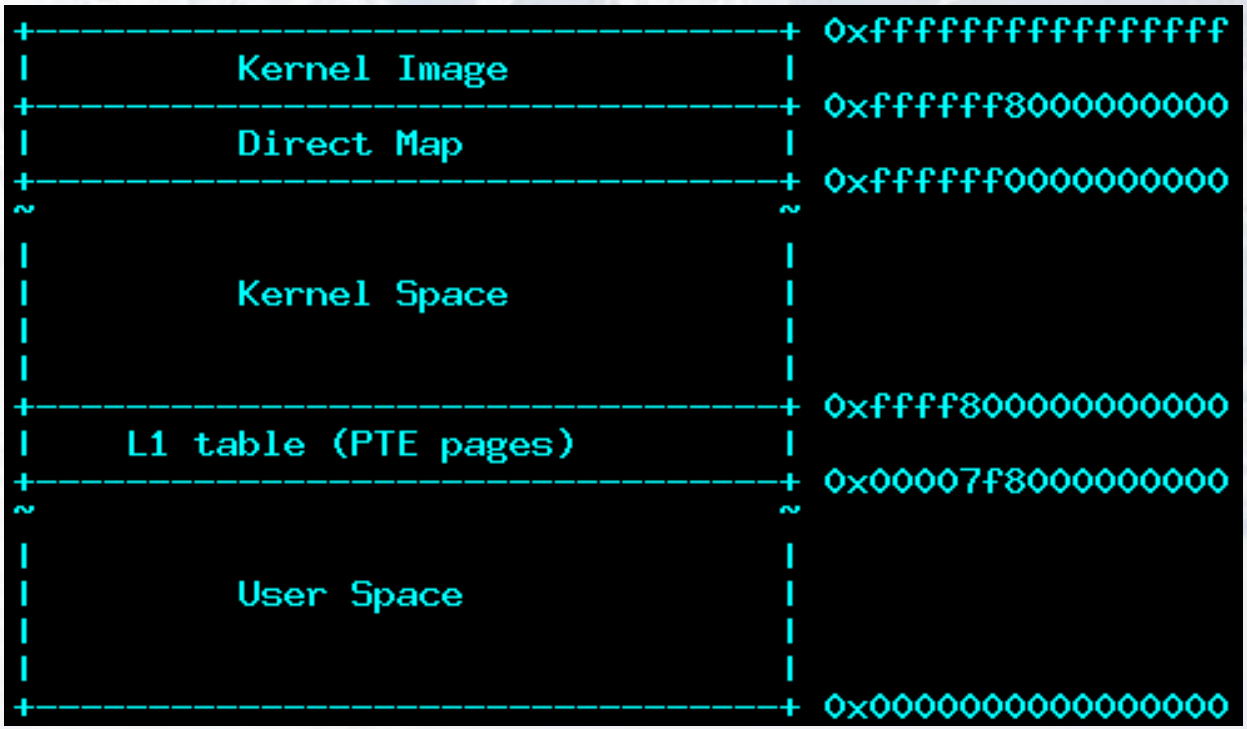
- No fuss, we just panic the machine.

Fixing Kernel W^X

- We have all the pieces in place now to enforce W^X
  - UVM enforcing sane requests
  - pmap code to enforce proper page permissions
  - Hardware that enforces the permissions
- So all we need to do now is identify all the different areas that need different permissions, and set everything up

Fixing amd64

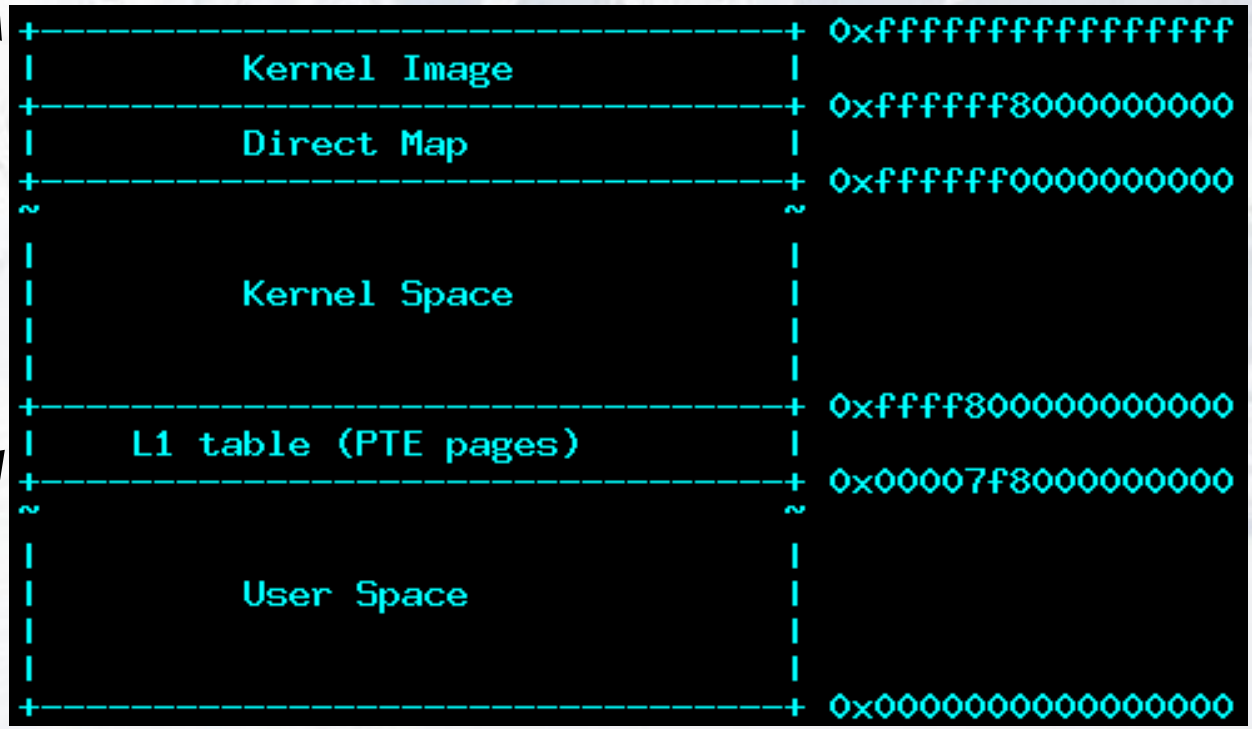
- Like most OSes, the virtual address (VA) space on OpenBSD amd64 is split into various regions



Fixing amd64

- Like most OSes, the virtual address (VA) space on OpenBSD amd64 is split into various regions

For this talk, I'm focusing on this area



Fixing amd64

- As earlier shown, the first commit to fix W^X in amd64 was the fix for the direct map region
  - That only leaves 3 more regions, how hard could that be?

Fixing amd64

- As earlier shown, the first commit to fix W^X in amd64 was the fix for the direct map region
  - That only leaves 3 more regions, how hard could that be?
  - If only it was that easy ...

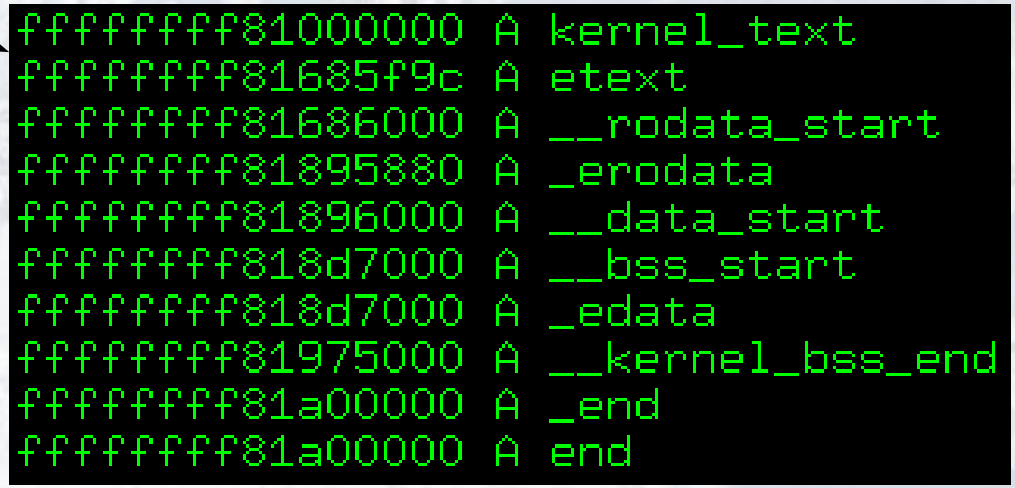
Fixing amd64

- The kernel area itself is subdivided
- Can't apply same (RW or RX) permissions to everything

```
ffffffff81000000 A kernel_text
ffffffff81685f9c A etext
ffffffff81686000 A __rodata_start
ffffffff81895880 A _erodata
ffffffff81896000 A __data_start
ffffffff818d7000 A __bss_start
ffffffff818d7000 A _edata
ffffffff81975000 A __kernel_bss_end
ffffffff81a00000 A _end
ffffffff81a00000 A end
```

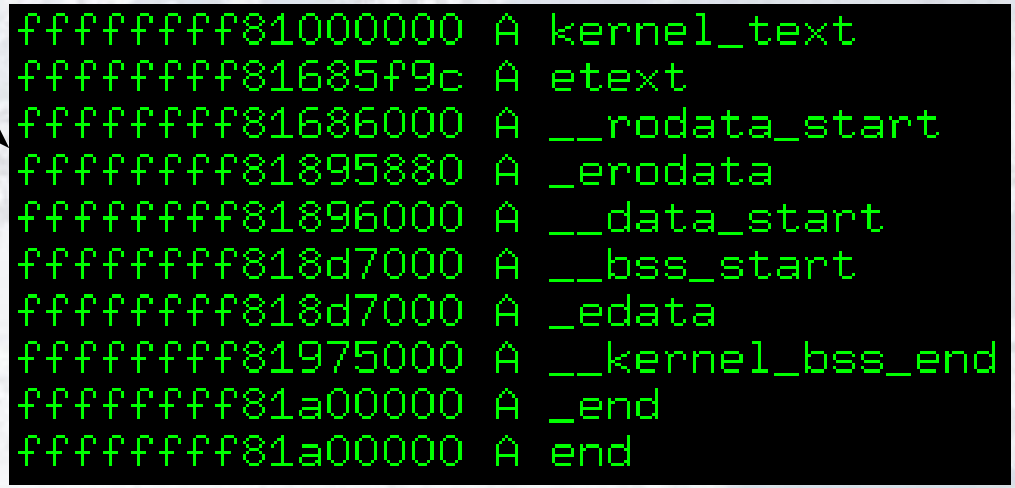
Fixing amd64

- Kernel text gets RX



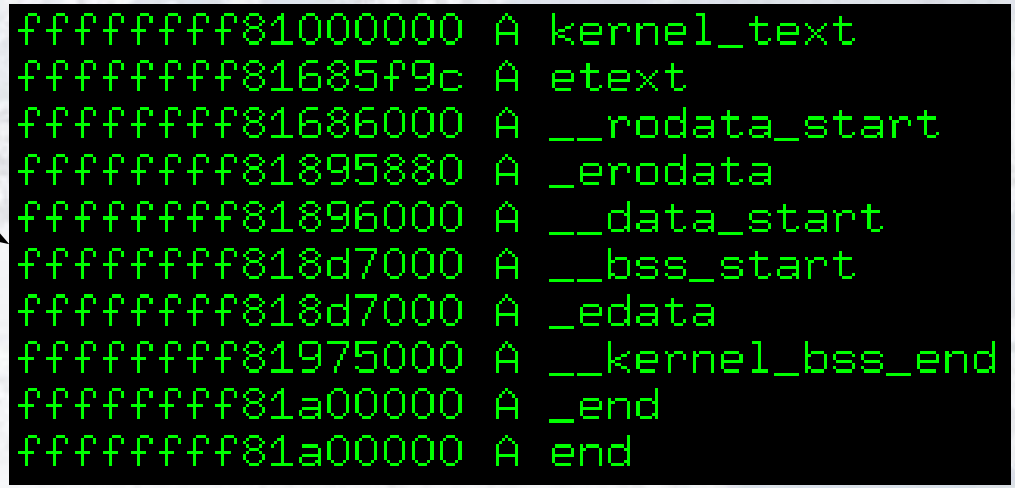
Fixing amd64

- Kernel text gets RX
- RO data gets R



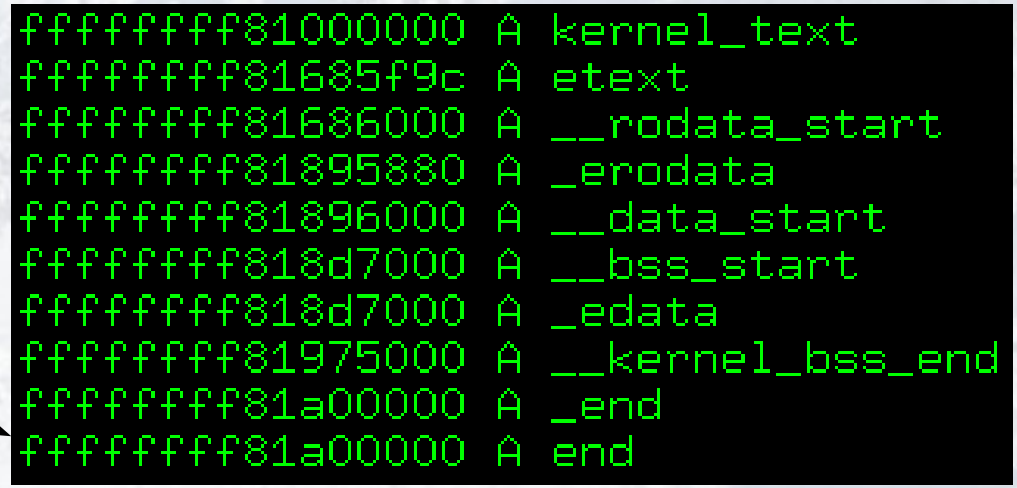
Fixing amd64

- Kernel text gets RX
- RO data gets R
- Data gets RW



Fixing amd64

- Kernel text gets RX
- RO data gets R
- Data gets RW
- Padding at the end gets R



Fixing amd64

- Before this, everything had X permissions, and some of the subdivisions didn't exist
  - That means data was RWX!
- Slowly, I fixed all this over the course of several months
  - Subdivide, apply permissions, repeat

Fixing amd64

- I fixed a few other things while I had the hood open
- ACPI resume trampoline
- MP spinup trampoline
  - Each trampoline was split into code and data/stack pages, with RX / RW perms.
  - Previously the trampolines were RWX

Fixing amd64

- Page tables
  - Page tables are now all NX
- APIC page
  - APIC page was executable before, now it isn't
- And of course if we missed something, we'll fix it when it becomes known

Verifying The Fixes

- How do you know if you did it right?
- A few ways …
  - Fix permissions, then intentionally try to break them somewhere
    - Should panic or die
  - Dump all the page permissions and look

Verifying The Fixes

- Tools like qemu and bochs can directly inspect the page table structure
  - In qemu, “info tlb” shows this information
  - In bochs, “page” shows this information
- For example:

```
ffffffff81478000: 0000000001478000 XG-DA---W
```

Verifying The Fixes

- Tools like qemu and bochs can directly inspect the page table structure
  - In qemu, “info tlb” shows this information
  - In bochs, “page” shows this information
- For example:

```
ffffffff81478000: 0000000001478000 XG-DA---W
```

- Permissions here, W = write,

  X = **no** execute

Fixing i386

- Someone challenged me over a beer to fix i386 next
  - I should have refused the beer

Fixing i386

- The memory map on i386 is similar to amd64
  - Smaller
  - No direct map
  - 3 level page table instead of 4
- Benefits from all of UVM's protections
  - Since UVM is machine independent

Fixing i386

- Our i386 pmap was very ancient
  - **NO** support for “NX” bit
- That meant every single page was executable

Fixing i386

- The first effort in fixing i386 was fixing its pmap
  - PAE page table
  - Has room for NX bit (if the hardware supports it)
- That took several months …
  - Existing i386 PAE code was 10+ years old
  - Full of bugs

Fixing i386

- Legacy machines complicate things
  - Some i386 machines don't support PAE
  - Some i386 machines don't support NX
- We have to leave the “old” pmap and the “new” pmap available, and decide at boot which to use

Fixing i386

- I flipped the switch to enable PAE on April 24th

```
revision 1.94
date: 2015/04/24 19:53:43;  author: mlarkin;  state: Exp;  lines: +1 -3;

Enable PAE mode for those CPUs that support it. This allows us to use the
NX bit for userland and kernel W^X. Unlike the previous c.2008 PAE
experiment, this does not provide > 4GB phys ram on i386 - PAE is solely
being used for NX capability this time. If you need > 4GB phys, use amd64.

Userland W^X was committed yesterday by kettenis@, and we will shortly
start reworking the kernel like we did for amd64 a few months back to get
kernel W^X.
```

Fixing i386

- Now that we had a way to enforce our W^X policy in hardware, a similar effort was made to subdivide and protect the kernel
- Second time (first was amd64) went much faster
  - But I got distracted by something called vmm ...

Fixing i386

- After enough urging by Theo, I spent a few days “finishing” i386 and committed the rest in August:

```
revision 1.161
date: 2015/08/25 04:57:31;  author: mlarkin;  state: Exp;  lines: +5 -5;

Enforce kernel w^x policy by properly setting NX (as needed) for
kernel text, PTEs, .rodata, data, bss and the symbol regions. This has
been in snaps for a while with no reported fallout.

The APTE space and MP/ACPI trampolines will be fixed next.
```

Finishing i386

- Alas, bug reports soon started appearing
  - Weird boot issues
  - Hangs
  - Reboots
- Unlike amd64, i386 still uses the machine BIOS for various things, and it wasn't protected right
  - Yuck.

Finishing i386

- Unfortunately, we needed to relax some of our page permissions in a region called the ISA hole
  - Sits after 640KB physical memory
  - Contains BIOS ROMs and other goo
- On amd64, we map this whole region NX
- On i386, it needs X permissions

Current Status

- amd64 is complete from what I can tell
  - Userland / kernel W^X
  - If someone finds a wrong mapping, I'd love to know about that
- I386 is mostly complete
  - Userland / kernel W^X
    - Left in old “line in the sand” mode for now
  - A few lingering BIOS bugs
  - Trampolines need to be split

Wrapping Up

- This was supposed to be a 1 month effort
  - “How hard could it possibly be?”
- I viewed it as a *correctness* fix, not a security fix
- After all the pages had proper W^X permissions, how many violators did we find in OpenBSD code on amd64?

Wrapping Up

- This was supposed to be a 1 month effort
  - “How hard could it possibly be?”
- I viewed it as a *correctness* fix, not a security fix
- After all the pages had proper W^X permissions, how many violators did we find in OpenBSD code on amd64?
  - ZERO.

Wrapping Up

- Keep in mind...
  - Nothing is a silver bullet
  - It's a cost analysis, and the cost is really low on this one.

Thanks For Listening

- Thanks Hackfest!
- Any questions?
- I'm mlarkin@openbsd.org if you have questions later