Kernel address space layout randomization

By **Jake Edge**
October 9, 2013

Linux Security Summit

Address-space layout randomization (ASLR) is a well-known technique to make exploits harder by placing various objects at random, rather than fixed, addresses. Linux has long had ASLR for user-space programs, but Kees Cook would like to see it applied to the kernel itself as well. He outlined the reasons why, along with how his patches work, in a Linux Security Summit talk. We looked at Cook's patches back in April, but things have changed since then; the code was based on the original proposal from Dan Rosenberg back in 2011.

Attacks

There is a classic structure to many attacks against the kernel, Cook said. An attacker needs to find a bug either by inspecting kernel code, noticing something in the patch stream, or following CVEs. The attacker can then use that bug to insert malicious code into the kernel address space by various means and redirect the kernel's execution to that that code. One of the easiest ways to get root privileges is to execute two simple functions as follows:

```
commit_creds(prepare_creds());
```

The existence of those function has made things "infinitely easier for an attacker", he said. Once the malicious code has been run, the exploit will then clean up after itself. For an example, he pointed to Rosenberg's RDS protocol local privilege escalation exploit.

These kinds of attacks rely on knowing where symbols of interest live in the kernel's address space. Those locations change between kernel versions and distribution builds, but are known (or can be figured out) for a

particular kernel. ASLR disrupts that process and adds another layer of difficulty to an attack.

ASLR in user space randomizes the location of various parts of an executable: stack, mmap region, heap, and the program text itself. Attacks have to rely on information leaks to get around ASLR. By exploiting some other bug (the leak), the attack can find where the code of interest has been loaded.

Randomizing the kernel's location

Cook's kernel ASLR (KASLR) currently only randomizes where the kernel code (i.e. text) is placed at boot time. KASLR "has to start somewhere", he said. In the future, randomizing additional regions is possible as well.

There are a number of benefits to KASLR. One side effect has been moving the interrupt descriptor table (IDT) away from the rest of the kernel to a location in read-only memory. The unprivileged `SIDT` instruction can be used to get the location of the IDT, which could formerly have been used to figure out where the kernel code was located. Now it can't be used that way because the IDT is elsewhere, but it is also protected from overwrite because it is read-only.

ASLR is a "statistical defense", because brute force methods can generally be used to overcome it. If there are 1000 locations where the item of interest could reside, brute force will find it once and fail 999 times. In user space, that failure will lead to a crash of the program, but that may not raise the kind of red flags that crashing 999 machines would. The latter is the likely outcome from a wrong brute force guess against KASLR.

On the other hand, KASLR is not compatible with hibernation (i.e. suspend to disk). That is a solvable problem, Cook said, but is not interesting to him. The amount of space available for the kernel text to move around in is another problem. The code must be 2M aligned because of page table restrictions, and the space available is 2G. In a "perfect world", that would give 1024 slots for the location. In the real world, it turns out to be a fair amount less.

There are also some steps that need to be taken to protect against information leaks that can be used to determine where the kernel was loaded. The `kptr_restrict` sysctl should be enabled so that kernel pointers are not leaked to user space. Similarly, `dmesg_restrict` should be used as dmesg often has addresses or other information that can be used. Also, log files (like `/var/log/messages`) should have permissions for root-only access.

The last source of leaks he mentioned is conceptually easy to fix, but has run into resistance from the network subsystem maintainers. The `INET_DIAG`

socket API uses the address of a kernel object as a handle. That address is opaque to user space, but it is a real kernel pointer, so it can be used to determine the kernel location. Changing it to some obfuscated value would fix the problem, but the network maintainers are not willing to do so, he said.

In a completely unconfined system, especially one with local untrusted users, KASLR is not going to be very useful, Cook said. But, on systems that use containers or have heavily contained processes, KASLR can help. For example, the renderer process in the Chrome browser is contained using the seccomp-BPF sandbox, which restricts an exploit to the point where it shouldn't be able to get the information needed. It is also useful to protect against attacks via remote services since there are "many fewer leaks" available remotely.

Implementation

KASLR has been added to Chrome OS, Cook said. It is in the Git tree for the distribution's kernel and will be rolled out in the stable release soon. He has a reputation for "bringing disruptive security changes to people who did not necessarily want them", he said with a smile, but KASLR was actually the "least problematic" of his proposed changes. Part of the reason for that is that "several other very smart people" have helped, including Rosenberg, other Google developers, and folks on the kernel mailing list.

Cook's patches change the boot process so that it determines the lowest safe address where the kernel could be placed. It then walks the e820 regions counting kernel-sized slots. From those, it chooses a slot randomly using the best random number source available. Depending on the system, that would be from the `RDRAND` instruction, the low bits from a `RDTSC` (time stamp counter), or bits from the timer I/O ports. After that, it decompresses the kernel, handles the relocation, and starts the kernel.

The patches are currently only for 64-bit x86, though Cook plans to look at ARM next. He knows a "lot less" about ARM, though, so he is hoping that he can "trick someone into helping me", he said.

The current layout of the kernel's virtual address space only leaves 512M for the kernel code—and 1.5G for modules. Since there is no need for that much module space, his patches reduce that to 1G, leaving 1G for the kernel, thus 512 possible slots (as it needs to be 2M aligned). The number of slots may increase when the modules' location is added to KASLR.

A demonstration of three virtual machines, with one running a "stock" kernel and two running the KASLR code, was up next. Looking at `/proc/kallsyms` and `/sys/kernel/debug/kernel_page_tables` on each

showed different addresses. Cook said that he was unable to find a measurable performance impact from KASLR.

The difference in addresses makes panics harder to decode, so the offset of the slot used to locate the kernel has been added to that output. He emphasized that information leaks are going to be more of a problem for KASLR-enabled systems, noting that it is somewhat similar to Secure Boot now making a distinction between root and kernel ring 0. For the most part, developers didn't care about kernel information leaks, but that needs to change.

There are some simple steps developers can take to avoid leaking kernel addresses, he said. Using the "`%pK`" format for printing addresses will show regular users 0, while root still sees the real address (if `kptr_restrict` is enabled, otherwise everyone sees the real addresses). The contents of dmesg need to be protected using `dmesg_restrict` and the kernel should not be using addresses as handles. All of those things will make KASLR an effective technique for thwarting exploits—at least in restricted environments.

[I would like to thank LWN subscribers for travel assistance to New Orleans for LSS.]

Kernel address space layout randomization
Posted Oct 10, 2013 17:03 UTC (Thu) by **kees** (subscriber, #27264) [Link]

Very minor correction: KASLR is working on both 32-bit and 64-bit x86 (with 256 and 512 possible random positions respectively), not just 64-bit. I do still want to make this work on ARM, though. :)

Kernel address space layout randomization
Posted Oct 13, 2013 17:26 UTC (Sun) by **rwmj** (guest, #5474) [Link]

I understand that huge pages are 2M, but why does that mean the kernel can only go at 2M boundaries? Isn't it possible to have ELF-style symbol relocations so the kernel can be moved to smaller offsets within the page (eg. the kernel would start at 2M page boundary + 1 byte)?

Anyway, here's hoping that once this is implemented, Ubuntu will make the vmlinuz files public readable again (and thus gain the much bigger security benefits of using libguestfs and VM sandboxes for user processes).

Reply to this comment

Kernel address space layout randomization
Posted Oct 17, 2013 22:08 UTC (Thu) by **heijo** (guest, #88363) [Link]

Does Ubuntu somehow randomly alter the vmlinuz files on disk?

If not, what's the point of making them unreadable, since the attacker can, very easily and automatically, determine that the distribution is Ubuntu, and download the files from their servers?

Reply to this comment

Kernel address space layout randomization
Posted Oct 18, 2013 0:07 UTC (Fri) by **spender** (guest, #23067) [Link]

They don't and of course you can absolutely automatically do what you just mentioned.

As for why:
http://lmgtfy.com/?q=cargo+cult+security

-Brad

Reply to this comment

Kernel address space layout randomization
Posted Oct 18, 2013 6:21 UTC (Fri) by **rwmj** (guest, #5474) [Link]

There's no security added by chmod go-r those files. However there is a lot of pain caused by it.

Reply to this comment

Kernel address space layout randomization
Posted Jan 24, 2014 15:59 UTC (Fri) by **deepfire** (guest, #26138) [Link]

> There's no security added by chmod go-r those files.
> However there is a lot of pain caused by it.

Really?

> Laying bare your code/data layouts for the local would-be-root's perusal is nothing?
>
> OTOH, if your logic goes along the lines of "there's so many other places you can get that information from, it just doesn't matter" -- I could agree.

Reply to this comment

Kernel address space layout randomization
Posted Oct 14, 2013 13:03 UTC (Mon) by **spender** (guest, #23067) [Link]

commit_creds(prepare_creds()) won't get you anything. The correct technique (which I first published) is commit_creds(prepare_kernel_cred(NULL))

Since it's relevant: http://forums.grsecurity.net/viewtopic.php?f=7&t=3367

I fully expect the warnings to be completely ignored and KASLR advertised as important security in future distro kernels, despite being completely useless.

-Brad

Reply to this comment