Linux security, one year later...

Nicolas Bareil

EADS Innovation Works
Suresnes, France

IT Defense

EADS

About

This talk

Describes what happened in 2010:

- New vulnerability classes
- New protections
- New ideas

This talk is **not**

- A rant (on the contrary)
- A long (and boring) list of vulnerabilities

EADS

The next hour...

Both points of view are analyzed:

1. Attacker side
2. Defensive

EADS

Plan

NULL pointer dereference

The vulnerability class of 2009

```
sock->ops->send_page(sock, ppos, pipe, len, flags);
```

What happens when **sock->opts == NULL**?

- Just a DoS in userspace (except for VM)
- Arbitrary code execution in kernel space

Dispersed by Julien Tinnes, Tavis Ormandy and Brad Spengler.

NULL pointer dereference

The vulnerability class of 2009

```
sock->ops->send_page(sock, ppos, pipe, len, flags);
```

What happens when **sock->opts == NULL**?

- Just a DoS in userspace (except for VM)
- Arbitrary code execution in kernel space

Dispersed by Julien Tinnes, Tavis Ormandy and Brad Spengler.

Principle

a process can map the first memory page (0–4096)

no segregation between kernel and user memory

When the kernel dereferences a NULL pointer, it will use the userspace pages if mapped.

Exploiting

Function pointer dereference

```
sock->ops->send_page(sock, ppos, pipe, len, flags);
```

Just drop off your shellcode at address **offsetof**(**sock->ops**, **send_page**)

Read/Write dereference

```
pipe = file->f_path.dentry->d_inode->i_pipe;
```

Fake a structure that will feed interesting values in order to control the execution path.

Proactive measure

The Right Way (tm)

Not having the bug in the first place is obviously the best. But we know some will slip through anyway. How to avoid that those bugs become exploitable privilege escalation vulnerabilities ?

- heavyweight/complex but effective : PaX UDEREF
- lightweight/simple: `mmap_min_addr`, adopted by mainstream

mmap_min_addr

Forbid processes to map pages below a limit:

Configured with /proc/sys/vm/mmap_min_addr

Very simple but with some shortcomings

mmap_min_addr

Forbid processes to map pages below a limit:

Configured with `/proc/sys/vm/mmap_min_addr`

Very simple but with some shortcomings

Mouse and cat

Game started, security researchers found several ways to bypass it:

- Places where the security check is missing,
- Special-cases disabling checks
- Side effects

At least 6 ways were found in 2009...

Mouse and cat continues...

In 2010, two ways were published:

- CVE-2010-4258: `set_fs()` override[1]
- CVE-2010-4346: Memory mapping instantiation[2]

[1] `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4258`
[2] `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4346`

Kernel's own hack

Kernel memory is mapped into all processes memory. Thanks to MMU, the process (ring 3) cannot access to kernel memory. When processing a system call, the kernel may have to write data to addresses provided by the process. The kernel checks that these addresses really belong to process' memory and not to kernel's memory. That prevents this kind of thing:

```
read(fd, 0xc1000000, 1)
```

access_ok()

Compares the pointer to a frontier (PAGE_OFFSET):

- Below is the user space
- Above is the kernel

When kernel cheats...

Sometimes, the kernel needs to use syscalls for its own usage, so the check shall not be made...

Hack spotted!

To prevent code duplications, a dirty trick is used: modifying the value of the frontier.
It makes `access_ok()` always returning true.

The frontier value modification is very limited in time!

. . . bad things happen: CVE-2010-4258

Objective

Trigger a NULL pointer dereference in this temporary context.

Nelson Elhage found that when an assertion failure is encountered (with a `BUG()` or an `Oops`), the kernel calls `do_exit()` on the triggering process.

Gotcha! Now find a pointer access!

. . . bad things happen: CVE-2010-4258

Objective

Trigger a NULL pointer dereference in this temporary context.

Nelson Elhage found that when an assertion failure is encountered (with a `BUG()` or an `Oops`), the kernel calls `do_exit()` on the triggering process.

Gotcha! Now find a pointer access!

Exploiting do_exit()

man set_tid_address:

When clear_child_tid is set, and the process exits, and the process was sharing memory with other processes or threads, then 0 is written at this address...

`BUG() -> do_exit() -> clear_child_tid -> access_ok()`

Kernel normally checks that the given address belongs to the parent... with `access_ok()` in the temporary context: attacker can write a 0 anywhere in virtual memory.

CVE-2010-4346: install_special_mapping

At execve() time, kernel maps ELF sections to memory. Tavis Ormandy found a way to map the VDSO page one page below the mmap_min_addr limit.

On RHEL, mmap_min_addr == 4096
$\Rightarrow$ VDSO mapped at 0x00000000

```
struct {
    short a;
    char b;
    int c;
} s;

s.a = X;
s.b = Y;
s.c = Z;

copy_to_user(to, &s, sizeof s);
```

- Padding byte between `.b` and `.c`
- Leaked to user land
- A process can keep hitting this code path in order to reveal sensible material eventually

Naive fix

Obvious fix

Put a `memset(&s, '\0', sizeof s)` before initializing the structure.

What is not so obvious...

C99 ignores totally padding issues, so the compiler is free to optimize code and can make the following assumptions:

- Considering the `memset()` call as a "dead store" as every structure's member are initialized
- Later, when assigning `.b`, compilers can overflow in the padding if needed

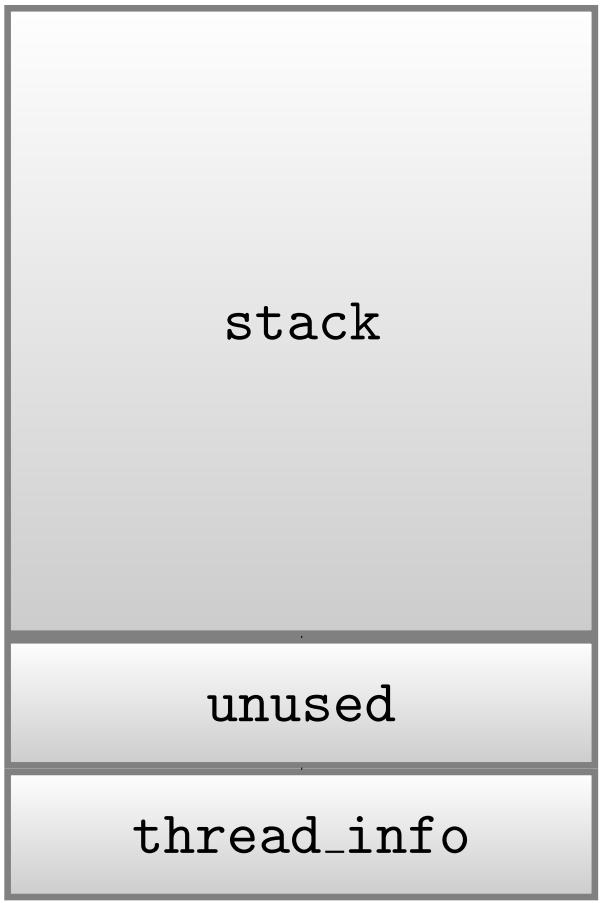
Kernel relies on compiler side-effects

Current GCC behavior is not intentional and could change in the future.

Possible solutions:

- CERT proposed the normalization of `memset_s()`, which would never be subject to "*dead store removal*" optimization.
- Explicitly define the padding bytes and mark the structure with the `__packed__` attribute.

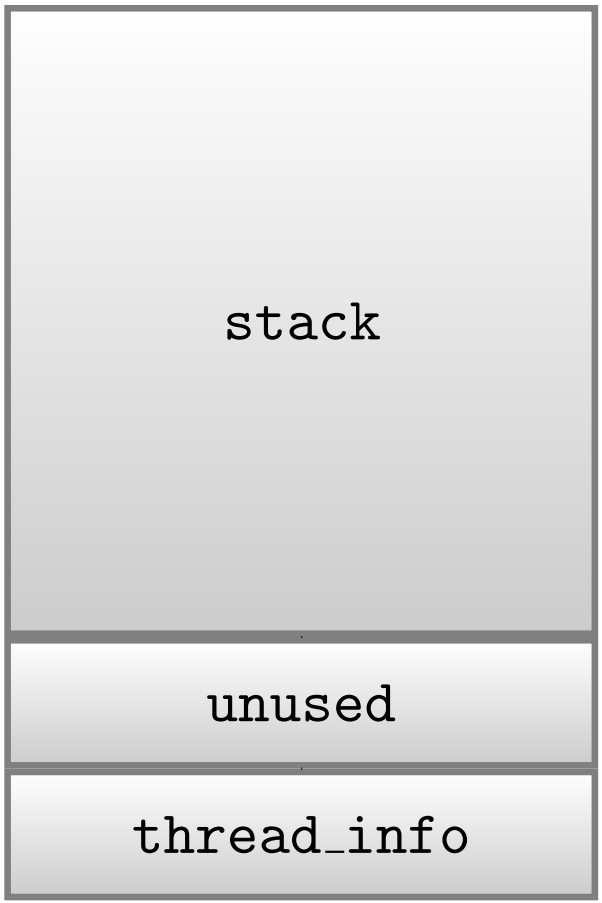
Memory layout



- Stack grows down
- Kernel tasks have a limited stack size: 2 pages max
- Limit is "conventional": no guard page

Expansion leads to expands on `thread_info` structure.

Stack expansion: CVE-2010-3848



Objective

Find a function where stack size is controled by attacker somehow.

Nelson Elhage found this behavior in `econet_sendmsg()`

Stack expansion in econet_sendmsg()

```
static int econet_sendmsg(struct kiocb *iocb, struct socket *sock,
                          struct msghdr *msg, size_t len)
{
        struct sock *sk = sock->sk;
        struct sockaddr_ec *saddr=(struct sockaddr_ec *)msg->msg_n
        struct net_device *dev;
        ...
        struct msghdr udpmsg;
        struct iovec iov[msg->msg_iovlen+1];
        struct aunhdr ah;
```

`iov` local variable is sized dynamically by a user-controlled length.

Plan

EADS

Too much information

`/proc`, `/dev`, `/sys` and `/debug` are full of pseudo-files which are gold mines to an attacker.

- Addresses
- Processes (PID, memory mapping, environment [not so long ago], signals, statistics, etc.)
- Internal statistics

Theses files are world-readable. . . and even world-writable for some

CVE-2010-4347: Embarrassing

Fuzzer discovered that `/sys/.../acpi/custom_method` was world-writable.

Any[3] user could upload custom methods to ACPI tables! Oops.

[3] `/debugfs` needs to be mounted

Information leak: addresses

Impact

Memory corruption vulnerabilities require to know at least one address to jump or write into.
⇒ Bruteforcing is not an option in kernel land.

Not needed! Every symbols are available:

- `/proc/kallsyms` lists functions addresses
- `/proc/modules` for modules address
- ...

```
grep -El '0x[0-9A-Fa-f]{8}' /proc -r 2> /dev/null
```

Information leak: addresses

Impact

Memory corruption vulnerabilities require to know at least one address to jump or write into.
⇒ Bruteforcing is not an option in kernel land.

Not needed! Every symbols are available:

- `/proc/kallsyms` lists functions addresses
- `/proc/modules` for modules address
- ...

```
grep -El '0x[0-9A-Fa-f]{8}' /proc -r 2> /dev/null
```

Limiting leaks: easy?

Rule #1: Never break user space!

Kernel must deal with broken legacy program. . . and have to live with it.

Rule #2: System must be debuggable

Developers sometimes have to work on "one-shot bug report", they can't ask the reporter to add `printk()`.

Limiting leaks: proposed solutions

- Change permissions
- Replace addresses with arbitrary values
- XOR displayed addresses with a secret value

Limiting leaks: proposed solutions

- ~~Change permissions~~: breaks rule #1
  klogd segfaults if it cannot open `/proc/kallsyms`
- Replace addresses with arbitrary values
- XOR displayed addresses with a secret value

Limiting leaks: proposed solutions

- ~~Change permissions~~: breaks rule #1
  klogd segfaults if it cannot open `/proc/kallsyms`
- ~~Replace addresses with arbitrary values~~: breaks rule #2
- XOR displayed addresses with a secret value

Limiting leaks: proposed solutions

- ~~Change permissions~~: breaks rule #1
  klogd segfaults if it cannot open `/proc/kallsyms`
- ~~Replace addresses with arbitrary values~~: breaks rule #2
- ~~XOR displayed addresses with a secret value~~: silly

Limiting leaks: proposed solutions

- ~~Change permissions~~: breaks rule #1
  klogd segfaults if it cannot open `/proc/kallsyms`
- ~~Replace addresses with arbitrary values~~: breaks rule #2
- ~~XOR displayed addresses with a secret value~~: silly

Retained solution: compromise

Use a special `printk()` specifier displaying dummy addresses if reader not privileged.
⇒ Introduction of the new capability `CAP_SYSLOG`

Memory usage

Currently, kernel does not use pages permissions for his own usage:

Section	Permissions
.data	READ, WRITE, EXECUTE
constants	READ, WRITE, EXECUTE
.text	READ, WRITE, EXECUTE

This is like userspace in the 80's

Hardening memory pages

Obviously, pages should be updated to be

Section	Permissions
`.data`	`READ, WRITE`
constants	`READ`
`.text`	`READ, EXECUTE`

Work in progress

1. Really set the physical page permissions (when NX available)
2. Declare the maximum of variables[4] as `const`
3. Hide `set_kernel_text()` entry points

[4] especially function pointers

Universal kernel

Vendor world

One kernel for all users: every features need to be present.
To avoid bloating the memory, everything is compiled in dynamically loaded modules.

Autoloading

Module loading is transparent for user: requesting a feature makes the kernel load the needed module.
$\Rightarrow$ Cool for attackers: ask for a SCTP socket and it's ready to be exploited :)

Universal kernel

Vendor world

One kernel for all users: every features need to be present.
To avoid bloating the memory, everything is compiled in dynamically loaded modules.

Autoloading

Module loading is transparent for user: requesting a feature makes the kernel load the needed module.
$\Rightarrow$ Cool for attackers: ask for a SCTP socket and it's ready to be exploited :)

Auto-loading

Mitigation

- Distributions disable auto-loading for some features really. . . insecure: X.25, SCTP, etc.
- A change was proposed: only privileged users could trigger auto-loading. But it was rejected for fear of breaking some legacy users.

Auto-loading

Mitigation

- Distributions disable auto-loading for some features really. . . insecure: X.25, SCTP, etc.
- A change was proposed: only privileged users could trigger auto-loading. But it was rejected for fear of breaking some legacy users.

UDEREF

PaX feature

Prevents NULL pointer dereference by putting kernel and user memory in two distinct segments.

Disclaimer

> *before everything, let's get out one thing that i'll probably repeat every now and then: UDEREF on amd64 isn't and will never be the same as on i386. it's just the way it is, it cannot be 'fixed'*
>
> pageexec, April, 9^{th}, 2010

UDEREF

PaX feature

Prevents NULL pointer dereference by putting kernel and user memory in two distinct segments.

Disclaimer

> *before everything, let's get out one thing that i'll probably repeat every now and then: UDEREF on amd64 isn't and will never be the same as on i386. it's just the way it is, it cannot be 'fixed'*
>
> pageexec, April, 9^{th}, 2010

UDEREF on AMD64

Without segmentation...

When switching to kernel mode, PaX moves the process memory at another address and changes permissions to deny any access.

Shortcomings

- This is "just" a shift of the problem
  - Instead of dereferencing a NULL pointer, attackers needs to dereference a specific address
  - But at this point, game is over anyway...
- Impact on performances: kernel transitions takes a hit

Plan

"Linux security module" design fail

The current security callbacks are absolutely nonsensical random crap slapped all around the kernel. It increases our security complexity and has thus the opposite effect - it makes us less secure.
Did no-one think of merging the capabilities checks and the security subsystem callbacks in some easy-to-use manner, which makes the default security policy apparent at first sight?

Ingo Molnar, November 30^{th}, 2010[5]

[5] http://thread.gmane.org/gmane.linux.kernel/1069948

Capability system is a mess

Quite frankly, the Linux capability system is largely a mess, with big bundled capacities that don't make much sense and are hideously inconvenient with the capability system used in user space (groups).

H. Peter Anvin, November 29th, 2010[6]

[6] http://permalink.gmane.org/gmane.linux.kernel.lsm/12196

-stable branch

>> I realise it wasn't ready for stable as Linus only pulled
>> it in 2.6.37-rc3, but surely that means this neither of
>> the changes should have gone into 2.6.32.26.
>
> Why didn't you respond to the review??

I don't actually read those review emails, there are too many of them.

Avi Kivity, KVM Maintainer, November 26^{th}, 2010[7]

[7] http://article.gmane.org/gmane.linux.kernel/1068374

Hopes

- Raise the cost of exploiting kernel vulnerabilities
  - We need more proactive measures!
- Wishlist:
  - Rethink LSM architecture
    - Pathname or label based?
  - Stackable LSM?

Thanks!

Full article on `http://justanothergeek.chdir.org/`

EADS