rise

RESEARCH GROUP

Kernel Intrusion Detection System

Rodrigo Rubira Branco

rodrigo@kernelhacking.com
rodrigo@risesecurity.org

Monica's Team!! Brazilian famous H.Q. story

Cebolinha, você come acelga?
Sim Mônica, a celga, a mulda, a sulda, e até a palapléugica

Amazon Forest – Yeah, Brazilian country!

Soccer – Brazilian team, the world most – we can lose, but we are the best in the world!

Em pé: Lúcio, Edmilson, Roque Júnior, Gilberto Silva, Marcos, Kaká, Vampeta, Anderson Polga, Dida, Rogério Ceni e Belletti. Agachados: Ronaldinho Gaúcho, Ronaldo, Roberto Carlos, Kléberson, Rivaldo, Cafu, Júnior, Ricardinho, Luizão, Edilson, Denilson e Juninho Paulista.

Samba, Carnaval, Brazilian's woman!

Agenda:

- The lifecycle of security (evading/identifying/correcting)
- Why will this presentation cover kernel-specific features?
- A little talk about compiler security enhancements (canary protection, code inspection)
- The mostly commons security resources for the linux kernel and how it works
- When does it fail? Kernel vulnerabilities/Kernel Attacks
    - FreeBSD 5.x 0day overview
- The proposal: Security Integrity Checks at Kernel Level
- A little talk about hooking the ELF loader to md5-sum the executed binaries
- StMichael, what is it?
- StMichael Test Cases
- Acknowledges

The lifecycle of security (evading/identifying/correcting)

- We try don't to put programming errors
- We try to identify the errors
- We try to correct that errors

But, bugs continuing to appear, security flaws continuing to be explored... what is missing?

An approach that improves the security, and turn the bug... ah... unexploitable?

Why will this presentation cover kernel-specific features?

- Actual and efficient security systems running in kernel mode

- To defeat that kind of systems, we must analyze kernel security

- We need to protect the kernel, to protect security systems

- Hum, it's funny cover the kernel ;)

A little talk about compiler security enhancements (canary protection, code inspection)

- Only detects/prevents the “classical” exploitation forms:
  * Format strings in standard implementations (or warnings against all other implementations)
  * Try to prevent overflows by inserting canary (can be easily defeated, as showed many times)

- Need to have improvements from the kernel mode:
  * No-exec stack, random allocation address (evade ret-into-lib)
  * Syscall trace
  * Be creative... let's analyze the existent ones!

The mostly commons security resources for the linux kernel and how it works

We will analyze the pros/cons of these security systems:

StJude
LIDS
PaX
SeLinux

These are the mostly common security systems used in Linux environments.

We don't plan to recommend or break any of that systems, only spot the improvements and problems existent today.

StJude

- Model to detect unauthorized/improper root transitions
- Originally developed by Timothy Lawless (presented at Defcon) and now maintained by Rodrigo Rubira Branco (me)
- All the privileged process are associated with a restriction list
- Each restriction list is based on a global rule-based that describes the valid transitions for an application in a defined context
- If a process creates a child, it will inherit the restriction list from its parent
- All executions are tested against the restriction list
- StJude uses another module, called StMichael (developed and maintained by the same people as Stjude) to protect the kernel mode

LIDS

- Trusted Domain Enforcement (TDE)

- Trusted Path Execution (TPE)

- Network enhancements
    * Socket restriction
    * Packet labels and netfilter interaction
    * Portscan/Sniffer detector

PaX

- KERNEXEC
* Introduces non-exec data into the kernel level
* Read-only kernel internal structures (feature exist in the 2.6.17.4 kernels)

- RANDKSTACK
* Introduce randomness into the kernel stack address of a task
* Not really useful when many tasks are involved nor when a task is ptraced (some tools use ptraced childs)

All that PaX features don't prevent kernel changes (so, an attacker maybe can't elevate privileges, but, change the kernel itself – introducing a backdoor, for example)

SeLinux

- SeLinux is not LSM! LSM is used by SeLinux to improve the system security

- A lot of discussion exist about SeLinux, specially when talking about pathnames into kernel mode

  * You can handle newly created files (like .procmailrc) using a usermode daemon (fedora implements restorecond to set the correct permissions/label into a newly created file)

- Another question is about the threat model, which leaves kernel exploits out of discussion

LSM

- Ok, because SeLinux uses the LSM framework, we will explain how the LSM framework works for the purpose of this presentation:

* security_operations structure that contains pointers to functions that will be called by the internal hooks

* dummy implementation that does nothing and will call the loaded module hooks (stackable) -> First problem... the stackable module support depends entirely on the modules, it will inherit a lot of complexity into the code (kernel bugs)

* all symbols are exported, so, anyone can use it in a backdoor (some samples...)! - for injection code, see phalanx in the references

LSM – Dumb Module Fragment

```
int myinode_rename(struct inode *old_dir, struct dentry *old_dentry,
                        struct inode *new_dir, struct dentry *new_dentry) {
     printk("\n dumb rename \n");

     return 0;
}

static struct security_operations my_security_ops = {
     .inode_rename      = myinode_rename,
};

register_security (&my_security_ops);
```

Kernel 2.6 – Backdoor Fragment

```
static int
execute(const char *string)
{


    ...

    if ((ret = call_usermodehelper(argv[0], argv, envp, 1)) != 0) {
       printk(KERN_ERR "Failed to run \"%s\": %i\n",
              string, ret);
    }
    return ret;

}
```

OBS: call_usermodehelper replaces the exec_usermodehelper showed in the phrack article (see references)

Kernel 2.6 – Backdoor Fragment

```
/* create a socket */
      if ( (err = sock_create(AF_INET, SOCK_DGRAM, IPPROTO_UDP, &kthread->sock)) < 0)
              printk(KERN_INFO MODULE_NAME": Could not create a datagram socket, error = %d\n", -ENXIO);
              goto out;
      }


      if ( (err = kthread->sock->ops->bind(kthread->sock, (struct sockaddr *)&kthread->addr,
        sizeof(struct sockaddr))) < 0)
              printk(KERN_INFO MODULE_NAME": Could not bind or connect to socket, error = %d\n", -err);
              goto close_and_out;
      }

      /* main loop */
      for (;;)
      {
              memset(&buf, 0, bufsize+1);
              size = ksocket_receive(kthread->sock, &kthread->addr, buf, bufsize);
```

OBS: See the references for a complete UDP Client/Server in kernel mode

Kernel 2.6 – Backdoor Fragment

```
static struct workqueue_struct *my_workqueue;

static struct work_struct Task;
static DECLARE_WORK(Task, intrpt_routine, NULL);

static void intrpt_routine(void *irrelevant)
{

     /* do the scheduled action here */

     if (!die)
          queue_delayed_work(my_workqueue, &Task, HZ);

}


my_workqueue = create_workqueue(MY_WORK_QUEUE_NAME);
queue_delayed_work(my_workqueue, &Task, 100);
```

OBS: StMichael uses this kind of schedule, it has been taken from the LKMPG Chapter 11 (see references)

Kernel 2.6 – Backdoor

- Putting all things together, so you have:

* UDP Client/Server -> You can use that to receive and respond to backdoor commands

* LSM registered functions (or hooks) -> Can intercept commands, hide things, and do interesting things (will be revised later)

* Execution from the kernel mode -> Can execute commands requested by the user

* Schedule tasks -> Permits scheduling the backdoor to run again (maybe to begin a new connection - connback), after a period of time

Yeah, only using public available sources!!

When does it fail? Kernel vulnerabilities/Kernel Attacks

- Kernel bugs have been exploited many times in the last years...

- PaX is the only security feature that covers kernel exploits

- When compromise the kernel, an attacker can easily defeat many of the existing security tools...

- Let's analyze a real kernel security flaw (yeah, Defcon is cool!)

FreeBSD 5.x 0day Overview

- Integer overflow vulnerability

- Useful for our sample because no security feature can prevent this kind of vulnerability from being exploited (PaX turns it really difficult, but don't prevent changes in the kernel integrity, only put some parts as read-only)

- Why show a FreeBSD and not a Linux vulnerability? Just for fun!

- Let's understand this flaw!

The result

The proposal: Security Integrity Checks at Kernel Level

- We can use many kernel features to offer a security integrity check in the kernel-level

- Specially for Defcon, I have first ported the StMichael to the 2.6 kernel branch (try it, help with patches)

- StMichael uses LSM only for checking the integrity of the [mod]_{register|unregister}_security functions and registering itself to avoid any other modules to be registered (but, don't cover the other hooks, it continues to be exported)...

A little talk about hooking the ELF loader to md5-sum the executed binaries

- Presented by Richard Johnson at Toorcon 2004

```
int
_load_binary (struct linux_binprm *linux_binprm, struct pt_regs *regs)
{

     ...
}
```

The parameter regs isn't used...

A little talk about hooking the ELF loader to md5-sum the executed binaries

```
int my_bprm_set_security (struct linux_binprm *bprm)
{

    if ( ! md5verify_sum(bprm->filename) )
    {
        printk("\n hey hey hey\n");
        return -1;
    }

    return 0;
}
```

StMichael, what is it?

- A Kernel Module developed to protect the kernel integrity

- Checks some kernel parts and look for changes

- Protects StJude (or any other kernel-security feature)

- Detects all existing public backdoors (and offers some security features – already covered by all others patches presented)

- Can't replace PaX, a good choice to complement it!

- **Show me the code!**

StMichael Test Cases

- Replacing kernel code
- Loading a module that hides itself
- Changing the printk
- Using the timer handler to put hackings...
- Trying to defeat the module itself (attacking StMichael)

Acknowledges

- RISE Security

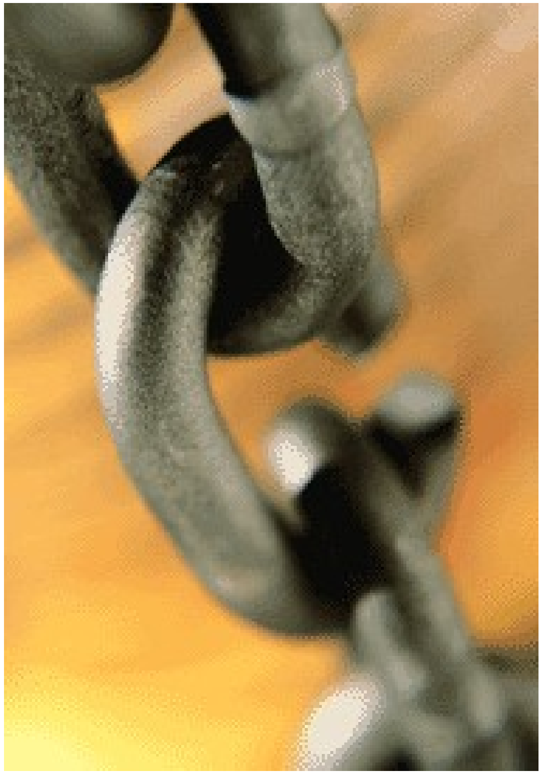
- Defcon Organization

- Pipacs (PaX creator)

- Timothy Lawless (StMichael/StJude creator)

- Your patience!

- Without friends, we are nothing, let's drink!



References

- PaX/GrSecurity: http://pax.grsecurity.net/docs/
- Selinux: http://www.nsa.gov/selinux/
- StJude/StMichael: http://www.sf.net/projects/stjude
- Lids: http://www.lids.org
- LSM discussions: http://www.kernelhacking.com/rodrigo/lsm
- Kernel UDP Client/Server: http://www.kernelnewbies.org/Simple_UDP_Server
- Izik's Userland Scheduler Paper:
http://www.uninformed.org/?v=3&a=6&t=pdf
- Hooking the ELF Loader:
http://labs.idefense.com/speaking/hooking_the_linux_ELF_loader.pdf
- Phalanx: http://packetstormsecurity.org/UNIX/penetration/rootkits/phalanx-b6.tar.bz2
- http://www.phrack.org/phrack/61/p61-0x0e_Kernel_Rootkit_Experiences.txt
- Chapter 11 (Scheduling Tasks): http://lkmpg.cvs.sourceforge.net/lkmpg/2.4/
- This Presentation/Samples: http://www.kernelhacking.com/rodrigo/defcon

All that references has last seen in: 07/03/2006.

END! Really is?

DOUBTS ?

Rodrigo Rubira Branco

rodrigo@kernelhacking.com
rodrigo@risesecurity.org