https://www.openwall.com/lkrg
Twitter: @Openwall
Twitter: @Adam_pi3

IN A NUTSHELL

/USR/BIN/WHOAMI

Private contact:

http://pi3.com.pl
pi3@pi3.com.pl
Twitter: @Adam_pi3

- Adam 'pi3' Zabrocki

- NVIDIA (currently)
- Microsoft
- European Organization for Nuclear Research (CERN)
- Hispasec Sistemas
- Wroclaw Centre for Networking and Supercomputing
- Cigital

- Bughunting (Hyper-V, KVM vGPU, Linux kernel, OpenSSH, gcc SSP/ProPolice, Apache, xpdf, more…) – CVEs
- Phrack magazine (Scraps of notes on remote stack overflow exploitation)
- The ERESI Reverse Engineering Software Interface

ACKNOWLEDGMENT

Alexander Peslyak (Александр Песляк) a.k.a. Solar Designer

The following people also had impact on LKRG:

- Mariusz Zaborski – code cleanups (and hopefully more in the future)
- Ilya Matveychikov – bypass techniques, which shaped up protections
- Michael Larabel (Phoronix) – benchmarks, which led to optimizations
- Patrick Schleizer (Whonix) – packaging with DKMS for Debian-compatibles
- Everyone who supported the project on Patreon

Special thanks to the following people for the constructive criticism and brainstorming in the past stages of the project development:

- Rafał "n3rgal" Wojtczuk
- Brad "spender" Spengler
- PaX Team… I mean "pipacs"

WHAT IS LKRG?

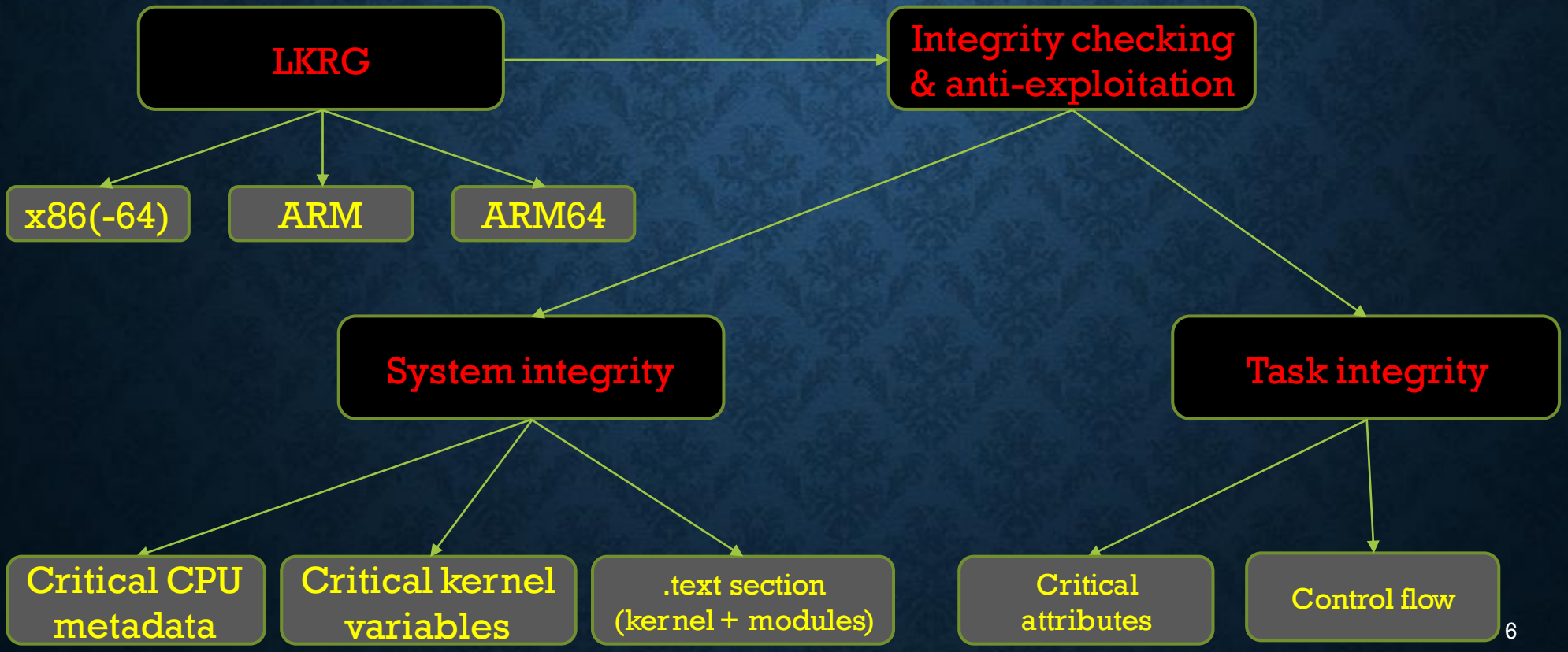
- LKRG – Linux Kernel Runtime Guard (self-explanatory ;p)

WHAT IS LKRG?

- LKRG – Linux Kernel Runtime Guard (self-explanatory ;p)
- Open Source project under GPLv2 License

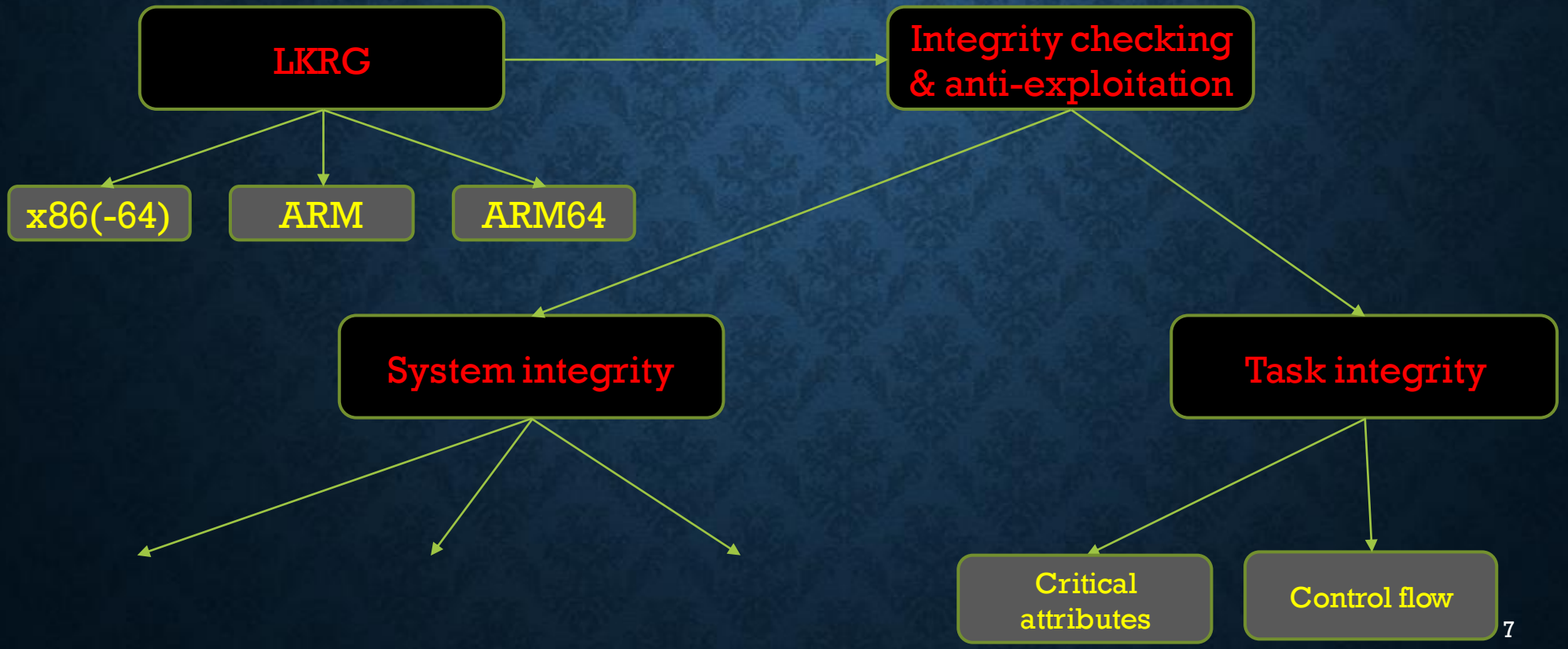
WHAT IS LKRG?

- LKRG – Linux Kernel Runtime Guard (self-explanatory ;p)
- Open Source project under GPLv2 License



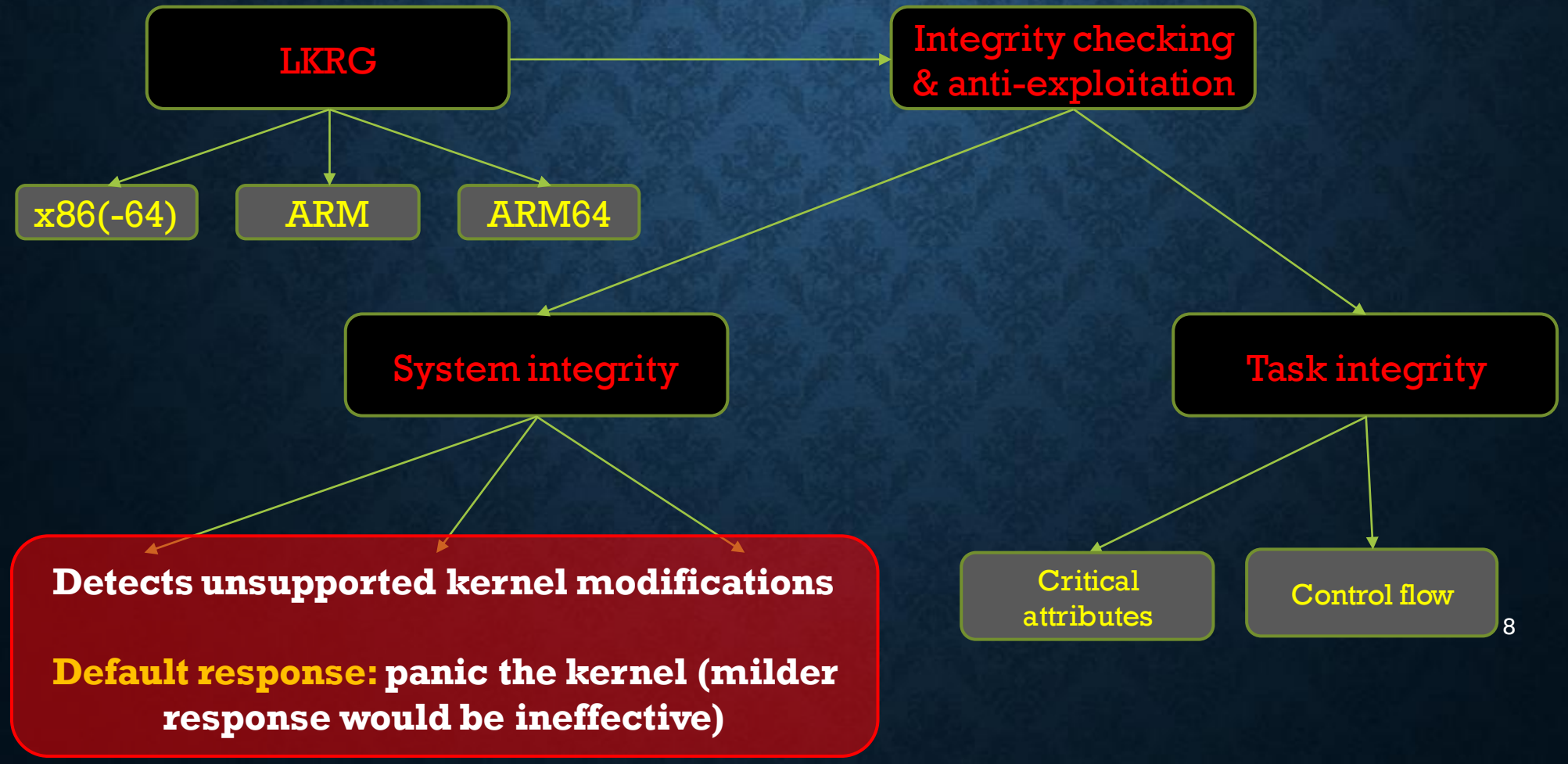
WHAT IS LKRG?

- LKRG – Linux Kernel Runtime Guard (self-explanatory ;p)
- Open Source project under GPLv2 License



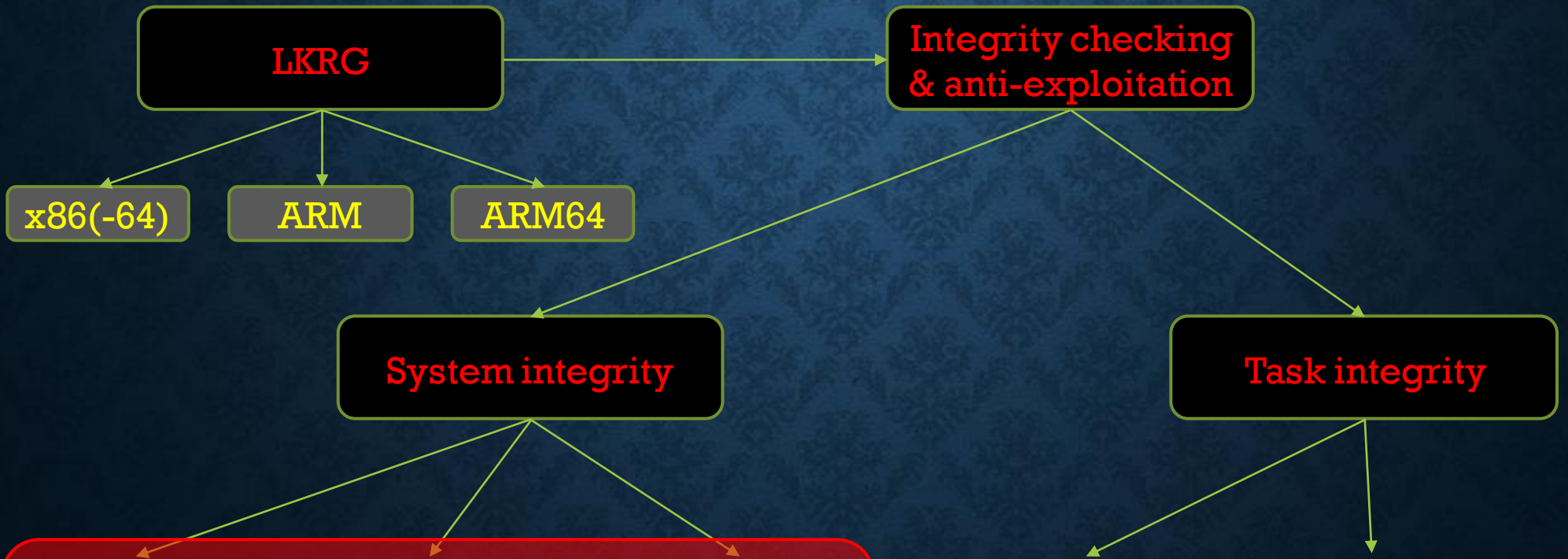
WHAT IS LKRG?

- LKRG – Linux Kernel Runtime Guard (self-explanatory ;p)
- Open Source project under GPLv2 License



WHAT IS LKRG?

- LKRG – Linux Kernel Runtime Guard (self-explanatory ;p)
- Open Source project under GPLv2 License

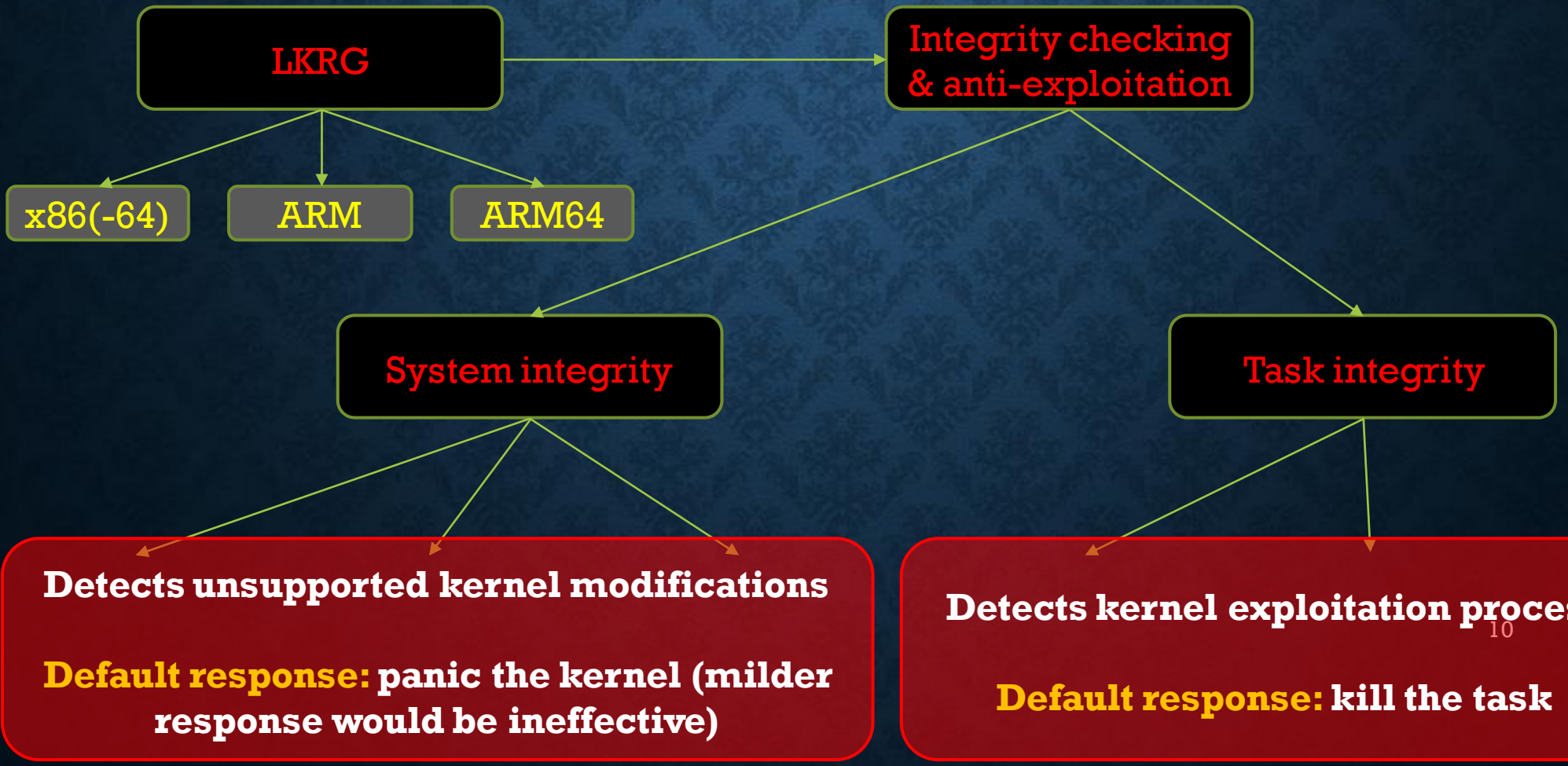


Detects unsupported kernel modifications

Default response: panic the kernel (milder response would be ineffective)

WHAT IS LKRG?

- LKRG – Linux Kernel Runtime Guard (self-explanatory ;p)
- Open Source project under GPLv2 License



WHAT IS LKRG?

- Officially, LKRG is distributed as source code:
  - https://www.openwall.com/lkrg/
  - https://github.com/openwall/lkrg

WHAT IS LKRG?

- Officially, LKRG is distributed as source code:
  - https://www.openwall.com/lkrg/
  - https://github.com/openwall/lkrg
- LKRG as a package:
  - ALT Linux
  - Arch Linux (aur)
  - Astra Linux
  - Debian and Ubuntu (reusing the Whonix/Kicksecure package)
  - Gentoo Linux (Pentoo Overlay)
  - Whonix and Kicksecure
  - + a few other less known

WHAT IS LKRG?

- Officially, LKRG is distributed as source code:
  - https://www.openwall.com/lkrg/
  - https://github.com/openwall/lkrg
- LKRG as a package:
  - ALT Linux
  - Arch Linux (aur)
  - Astra Linux
  - Debian and Ubuntu (reusing the Whonix/Kicksecure package)
  - Gentoo Linux (Pentoo Overlay)
  - Whonix and Kicksecure
  - + a few other less known
- LKRG-aware exploitation frameworks:
  - Metasploit bails out
  - Exploit-suggester bails out

ANTI-EXPLOITATION

- The aim of it is to detect kernel exploitation process by detecting specific data corruption in the kernel

ANTI-EXPLOITATION

- The aim of it is to detect kernel exploitation process by detecting specific data corruption in the kernel

ANTI-EXPLOITATION

- The aim of it is to detect kernel exploitation process by detecting specific data corruption in the kernel

- Illegal Elevation of Privileges (EoP):
  - Token / pointer swapping
  - Illegal call to commit_creds()
  - Overwriting the cred / real_cred structures

- Sandbox escapes (e.g. Chrome sandbox):
  - Overwriting seccomp configuration
  - Overwriting seccomp rules

- Various namespace escapes
- Various container escapes (e.g. Docker / Kubernetes / etc.)

- Illegal changes of:
  - CPU state e.g. SMAP / SMEP / WP / MSR
  - Any part of the kernel or modules

ANTI-EXPLOITATION

- The aim of it is to detect kernel exploitation process by detecting specific data corruption in the kernel

Critical process' and system's attributes

- Illegal Elevation of Privileges (EoP):
  - Token / pointer swapping
  - Illegal call to commit_creds()
  - Overwriting the cred / real_cred structures
- Sandbox escapes (e.g. Chrome sandbox):
  - Overwriting seccomp configuration
  - Overwriting seccomp rules
- Various namespace escapes
- Various container escapes (e.g. Docker / Kubernetes / etc.)
- Illegal changes of:
  - CPU state e.g. SMAP / SMEP / WP / MSR
  - Any part of the kernel or modules

Poor man's Control Flow Integrity (pCFI)

- It might detect (and block)
  - Return-Oriented-Programming (ROP)
  - Stack-pivoting attacks
- It might detect illegal control flow:
  - From non .text section pages
  - From dynamically generated executable pages
  - From pages not belonging to the kernel (e.g. user-mode pages)
  - When attacker bypasses SMEP protection

ANTI-EXPLOITATION

- Examples
  - Detection of calls into kernel APIs from non-code pages (CVE-2017-1000112)

```
Dec 15 12:41:10 pi3-ubuntu kernel: [p_lkrg] <Exploit Detection> Not valid call - pCFI violation: process[poc | 3250] !!!
Dec 15 12:41:10 pi3-ubuntu kernel: [p_lkrg] <Exploit Detection> Frame[1] nr_entries[13]: [0x40115b]. Full Stack:
Dec 15 12:41:10 pi3-ubuntu kernel: --- . ---
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff82c62da1>] __mark_inode_dirty+0x1/0x370
Dec 15 12:41:10 pi3-ubuntu kernel:  [<000000000040115b>] 0x40115b
Dec 15 12:41:10 pi3-ubuntu kernel:  [<00000000004011ce>] 0x4011ce
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff8316a164>] skb_release_all+0x24/0x30
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff8316a1c2>] kfree_skb+0x32/0x90
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff831c5e93>] __ip_flush_pending_frames.isra.40+0x43/0x90
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff831c74ac>] ip_flush_pending_frames+0x1c/0x20
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff831f102b>] udp_sendmsg+0x3eb/0xa80
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff831fe5e5>] inet_sendmsg+0x65/0xa0
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff83160a58>] sock_sendmsg+0x38/0x50
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff83161061>] SYSC_sendto+0x101/0x190
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff83161bbe>] SyS_sendto+0xe/0x10
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff8329a876>] entry_SYSCALL_64_fastpath+0x1e/0xa8
Dec 15 12:41:10 pi3-ubuntu kernel: --- END ---
Dec 15 12:41:10 pi3-ubuntu kernel: [p_lkrg] <Exploit Detection> Trying to kill process[poc | 3250]!
Dec 15 12:41:10 pi3-ubuntu kernel: [p_lkrg] <Exploit Detection> Path's inode[6951377] mode[0x89fd] will be isolated!
```

ANTI-EXPLOITATION

- Examples
  - Detection of calls into kernel APIs from non-code pages (CVE-2017-1000112)

```
Dec 15 12:41:10 pi3-ubuntu kernel: [p_lkrg] <Exploit Detection> Not valid call - pCFI violation: process[poc | 3250] !!!
Dec 15 12:41:10 pi3-ubuntu kernel: [p_lkrg] <Exploit Detection> Frame[1] nr_entries[13]: [0x40115b]. Full Stack:
Dec 15 12:41:10 pi3-ubuntu kernel: --- . ---
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff82c62da1>] __mark_inode_dirty+0x1/0x370
Dec 15 12:41:10 pi3-ubuntu kernel:  [<000000000040115b>] 0x40115b
Dec 15 12:41:10 pi3-ubuntu kernel:  [<00000000004011ce>] 0x4011ce
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff8316a164>] skb_release_all+0x24/0x30
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff8316a1c2>] kfree_skb+0x32/0x90
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff831c5e93>] __ip_flush_pending_frames.isra.40+0x43/0x90
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff831c74ac>] ip_flush_pending_frames+0x1c/0x20
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff831f102b>] udp_sendmsg+0x3eb/0xa80
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff831fe5e5>] inet_sendmsg+0x65/0xa0
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff83160a58>] sock_sendmsg+0x38/0x50
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff83161061>] SYSC_sendto+0x101/0x190
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff83161bbe>] SyS_sendto+0xe/0x10
Dec 15 12:41:10 pi3-ubuntu kernel:  [<ffffffff8329a876>] entry_SYSCALL_64_fastpath+0x1e/0xa8
Dec 15 12:41:10 pi3-ubuntu kernel: --- END ---
Dec 15 12:41:10 pi3-ubuntu kernel: [p_lkrg] <Exploit Detection> Trying to kill process[poc | 3250]!
Dec 15 12:41:10 pi3-ubuntu kernel: [p_lkrg] <Exploit Detection> Path's inode[6951377] mode[0x89fd] will be isolated!
```

ANTI-EXPLOITATION

- Examples
  - Detection of calls into kernel APIs from non-code pages (CVE-2017-1000112)

```
pi3@pi3-ubuntu:~/z_confidence/z/2$ ./poc
[.] starting
[.] checking distro and kernel versions
[.] kernel version '4.8.0-53-generic' detected
[~] done, versions looks good
[.] checking SMEP and SMAP
[~] done, looks good
[.] setting up namespace sandbox
[~] done, namespace sandbox set up
[.] KASLR bypass enabled, getting kernel addr
[~] done, kernel text:   ffffffff82a00000
[.] commit_creds:        ffffffff82aa5d00
[.] prepare_kernel_cred: ffffffff82aa60f0
[.] SMEP bypass enabled, mmapping fake stack
stack => 0x82a17c55
[~] done, fake stack mmapped
[.] executing payload ffffffff82a17c55
Killed
pi3@pi3-ubuntu:~/z_confidence/z/2$ ls -al /tmp/shell
----------. 1 nobody nogroup 8720 Dec 15 12:41 /tmp/shell
pi3@pi3-ubuntu:~/z_confidence/z/2$
```

ANTI-EXPLOITATION

- Examples
  - Detection of calls into kernel APIs from non-code pages (CVE-2017-1000112)

```
pi3@pi3-ubuntu:~/z_confidence/z/2$ ./poc
[.] starting
[.] checking distro and kernel versions
[.] kernel version '4.8.0-53-generic' detected
[~] done, versions looks good
[.] checking SMEP and SMAP
[~] done, looks good
[.] setting up namespace sandbox
[~] done, namespace sandbox set up
[.] KASLR bypass enabled, getting kernel addr
[~] done, kernel text:   ffffffff82a00000
[.] commit_creds:        ffffffff82aa5d00
[.] prepare_kernel_cred: ffffffff82aa60f0
[.] SMEP bypass enabled, mmapping fake stack
stack => 0x82a17c55
[~] done, fake stack mmapped
[.] executing payload ffffffff82a17c55
Killed
pi3@pi3-ubuntu:~/z_confidence/z/2$ ls -al /tmp/shell
----------. 1 nobody nogroup 8720 Dec 15 12:41 /tmp/shell
pi3@pi3-ubuntu:~/z_confidence/z/2$
```

ANTI-EXPLOITATION

- Examples
  - Detection of calls into kernel APIs from non-code pages (CVE-2017-1000112)

```
pi3@pi3-ubuntu:~/z_confidence/z/2$ ./poc
[.] starting
[.] checking distro and kernel versions
[.] kernel version '4.8.0-53-generic' detected
[~] done, versions looks good
[.] checking SMEP and SMAP
[~] done, looks good
[.] setting up namespace sandbox
[~] done, namespace sandbox set up
[.] KASLR bypass enabled, getting kernel addr
[~] done, kernel text:   ffffffff82a00000
[.] commit_creds:        ffffffff82aa5d00
[.] prepare_kernel_cred: ffffffff82aa60f0
[.] SMEP bypass enabled, mmapping fake stack
stack => 0x82a17c55
[~] done, fake stack mmapped
[.] executing payload ffffffff82a17c55
Killed
pi3@pi3-ubuntu:~/z_confidence/z/2$ ls -al /tmp/shell
----------. 1 nobody nogroup 8720 Dec 15 12:41 /tmp/shell
pi3@pi3-ubuntu:~/z_confidence/z/2$
```

ANTI-EXPLOITATION

- Examples
  - ROP detection

```
                (stack) legit ret
                (stack) legit ret
                (stack) legit ret
                (stack) legit ret doing stack pivoting for ROP
[detection race] (heap) legit ret (ROP chunk 1 - discover original stack)
[detection race] (heap) ...
[detection race] (heap) legit ret (ROP chunk N - discover original stack)
[detection race] (heap) legit ret (ROP chunk 1 - copy phase 2 ROP to the original stack)
[detection race] (heap) ...
[detection race] (heap) legit ret (ROP chunk N - copy phase 2 ROP to the original stack)
[detection race] (heap) legit ret (ROP - reverse stack pivoting)
                (stack) legit ret (ROP phase 2 - chunk 1)
                (stack) ...
                (stack) legit ret (ROP phase 2 - chunk N)
```

```
[p_lkrg] <Exploit Detection> Stack pointer corruption (ROP?) - pCFI violation: process[poc | 2127] !!!
[p_lkrg] <Exploit Detection> Trying to kill process[poc | 2127]!
[p_lkrg] <Exploit Detection> process [poc | 2127] has invalid base for stack pointer! [base:0xffff8cdba4f980
[p_lkrg] <Exploit Detection> process [poc | 2127] has invalid stack pointer (stack size mismatch)! [base:0xf
```

ANTI-EXPLOITATION

- Examples
  - ROP detection

```
                (stack) legit ret
                (stack) legit ret
                (stack) legit ret
                (stack) legit ret doing stack pivoting for ROP
[detection race] (heap) legit ret (ROP chunk 1 - discover original stack)
[detection race] (heap) ...
[detection race] (heap) legit ret (ROP chunk N - discover original stack)
[detection race] (heap) legit ret (ROP chunk 1 - copy phase 2 ROP to the original stack)
[detection race] (heap) ...
[detection race] (heap) legit ret (ROP chunk N - copy phase 2 ROP to the original stack)
[detection race] (heap) legit ret (ROP - reverse stack pivoting)
                (stack) legit ret (ROP phase 2 - chunk 1)
                (stack) ...
                (stack) legit ret (ROP phase 2 - chunk N)
```

```
[p_lkrg] <Exploit Detection> Stack pointer corruption (ROP?) - pCFI violation: process[poc | 2127] !!!
[p_lkrg] <Exploit Detection> Trying to kill process[poc | 2127]!
[p_lkrg] <Exploit Detection> process [poc | 2127] has invalid base for stack pointer! [base:0xffff8cdba4f980
[p_lkrg] <Exploit Detection> process [poc | 2127] has invalid stack pointer (stack size mismatch)! [base:0xf
```

ANTI-EXPLOITATION

- Examples - metasploit

```
msf5 exploit(linux/local/bpf_sign_extension_priv_esc) > run

[*] Started reverse TCP handler on 192.168.51.128:4444
[-] Exploit aborted due to failure: not-vulnerable: Target is not vulnerable. Set ForceExploit to override.
[*] Exploit completed, but no session was created.
msf5 exploit(linux/local/bpf_sign_extension_priv_esc) > run

[*] Started reverse TCP handler on 192.168.51.128:4444
[-] Exploit aborted due to failure: not-vulnerable: Target is not vulnerable. Set ForceExploit to override.
[*] Exploit completed, but no session was created.
msf5 exploit(linux/local/bpf_sign_extension_priv_esc) > set ForceExploit
ForceExploit => false
msf5 exploit(linux/local/bpf_sign_extension_priv_esc) > set ForceExploit true
ForceExploit => true
msf5 exploit(linux/local/bpf_sign_extension_priv_esc) > run

[*] Started reverse TCP handler on 192.168.51.128:4444
[!] Target does not appear to be vulnerable
[*] Writing '/tmp/.0wtvC0F9cZ.c' (10867 bytes) ...
[*] Writing '/tmp/.vc0sg' (250 bytes) ...
[*] Launching exploit ...
[*] Cleaning up /tmp/.vc0sg and /tmp/.0wtvC0F9cZ ...
[*] Exploit completed, but no session was created.
msf5 exploit(linux/local/bpf_sign_extension_priv_esc) >

Active sessions
===============

  Id  Name  Type                     Information
  --  ----  ----                     -----------
  1         meterpreter x86/linux  no-user @ ubuntu (uid=1000, gid=1000, euid=1000, egid=1000) @ 192.168.51.128
 (192.168.51.128)

msf5 exploit(linux/local/bpf_sign_extension_priv_esc) > set ForceExploit false
ForceExploit => false
msf5 exploit(linux/local/bpf_sign_extension_priv_esc) > run

[*] Started reverse TCP handler on 192.168.51.128:4444
[-] Exploit aborted due to failure: not-vulnerable: Target is not vulnerable. Set ForceExploit to override.
[*] Exploit completed, but no session was created.
msf5 exploit(linux/local/bpf_sign_extension_priv_esc) > run

[*] Started reverse TCP handler on 192.168.51.128:4444
[*] Writing '/tmp/.LDyR1.c' (10867 bytes) ...
[*] Writing '/tmp/.SXlPE6' (250 bytes) ...
[*] Launching exploit ...
[*] Sending stage (3012516 bytes) to 192.168.51.128
[*] Meterpreter session 4 opened (192.168.51.128:4444 -> 192.168.51.128:33896) at 2020-06-24 13:52:47 -0700
[*] Cleaning up /tmp/.SXlPE6 and /tmp/.LDyR1 ...

meterpreter > getuid
Server username: no-user @ ubuntu (uid=0, gid=0, euid=0, egid=0)
meterpreter >
```

Terminal: Metasploit

```
root@ubuntu:~/lkrg-main# insmod output/p_lkrg.ko
root@ubuntu:~/lkrg-main# tail -f /var/log/kern.log
Jun 24 13:50:03 ubuntu kernel: [ 6002.544092] [p_lkrg] LKRG unloaded!
Jun 24 13:50:28 ubuntu kernel: [ 6027.115585] [p_lkrg] Loading LKRG...
Jun 24 13:50:28 ubuntu kernel: [ 6027.115847] [p_lkrg] System does NOT support SMAP. LKRG can't enforce SMAP validation :(
Jun 24 13:50:28 ubuntu kernel: [ 6027.124086] Freezing user space processes ... (elapsed 0.001 seconds) done.
Jun 24 13:50:28 ubuntu kernel: [ 6027.125633] Double checking all user space processes after OOM killer disable... (elapsed 0.000 sec
Jun 24 13:50:28 ubuntu kernel: [ 6027.125751] [p_lkrg] 5/26 UMH paths are allowed...
Jun 24 13:50:28 ubuntu kernel: [ 6027.274605] [p_lkrg] [kretprobe] register_kretprobe() for <ovl_create_or_link> failed! [err=-2]
Jun 24 13:50:28 ubuntu kernel: [ 6027.274700] [p_lkrg] Can't hook 'ovl_create_or_link' function. This is expected if you are not using
Jun 24 13:50:28 ubuntu kernel: [ 6027.388190] [p_lkrg] LKRG initialized successfully!
Jun 24 13:50:28 ubuntu kernel: [ 6027.388243] Restarting tasks ... done.

Jun 24 13:51:52 ubuntu kernel: [ 6110.913840] [p_lkrg] <Exploit Detection> process[6262 | .0wtvC0F9cZ] has different UID! 1000 vs 0
Jun 24 13:51:52 ubuntu kernel: [ 6110.914408] [p_lkrg] <Exploit Detection> process[6262 | .0wtvC0F9cZ] has different UID! 1000 vs 0
Jun 24 13:51:52 ubuntu kernel: [ 6110.914911] [p_lkrg] <Exploit Detection> Trying to kill process[.0wtvC0F9cZ | 6262]!
^C
root@ubuntu:~/lkrg-main# rmmod p_lkrg
root@ubuntu:~/lkrg-main#
```

Terminal: LKRG

ANTI-EXPLOITATION

- Limitations – Bypassable by design (for now) – difficult to protect from the same trust level
  - Fly under LKRG's radar:
    - ✓ Overwrite critical metadata not guarded by LKRG
    - ✓ Try to win races
    - ✓ Move attack to userspace
  - Attack (disable) LKRG and continue normal work:
    - ✓ Try to win races (corrupting LKRG's database)
    - ✓ Attack LKRG's internal synchronization / locking
    - ✓ Find all LKRG's running contexts and disable them + block a new one
  - Directly attack the userspace via kernel (e.g. DirtyCOW)

SYSTEM INTEGRITY

- Calculate hash from the critical [meta]data – SipHash
- Guarded regions:
  - Critical (V)CPU/core data – Inter-Processor-Interrupt (IPI) is sent to the individual core in all (V)CPUs to exclusively run LKRG's guard function (IDT/MSR/CRx/etc.)
    - LKRG keeps information about how many (V)CPU/cores are „online" / „offline" / „possible"
  - Entire Linux kernel .text section
    - This covers almost entire Linux kernel itself, like syscall tables, all procedures, all function, all IRQ handlers, etc.
  - Entire Linux kernel .rodata section
  - Entire Linux kernel exception table
  - Critical global system variables, like:
    - selinux_enabled
    - selinux_enforcing / selinux_state
    - Supervisor Mode Execution Protection (SMEP) and Supervisor Mode Access Prevention (SMAP)
    - CR4.WP
  - All dynamically loaded modules AND their order in the internal structures
  - Optionally, it is possible to enable guard of the entire IOMMU table

SYSTEM INTEGRITY

- Calculate hash from the critical [meta]data – SipHash
- Guarded regions:
  - Critical (V)CPU/core data – Inter-Processor-Interrupt (IPI) is sent to the individual core in all (V)CPUs to exclusively run LKRG's guard function (IDT/MSR/CRx/etc.)
    - LKRG keeps information about how many (V)CPU/cores are „online" / „offline" / „possible"
  - Entire Linux kernel .text section
    - This covers almost entire Linux kernel itself, like syscall tables, all procedures, all function, all IRQ handlers, etc.
  - Entire Linux kernel .rodata section
  - Entire Linux kernel exception table
  - Critical global system variables, like:
    - selinux_enabled
    - selinux_enforcing / selinux_state

    ← SELinux escape

    Detects SMAP / SMEP bypasses →

    - Supervisor Mode Execution Protection (SMEP) and Supervisor Mode Access Prevention (SMAP)
    - CR4.WP ← Often changed by rootkits
  - All dynamically loaded modules AND their order in the internal structures
  - Optionally, it is possible to enable guard of the entire IOMMU table

COMMUNICATION CHANNEL

- Through the sysctl interface:

```
root@pi3-ubuntu:~/p_lkrg-main# sysctl -a | grep lkrg
lkrg.block_modules = 0
lkrg.heartbeat = 0
lkrg.hide = 0
lkrg.interval = 15
lkrg.kint_enforce = 2
lkrg.kint_validate = 3
lkrg.log_level = 3
lkrg.msr_validate = 1
lkrg.pcfi_enforce = 1
lkrg.pcfi_validate = 2
lkrg.pint_enforce = 1
lkrg.pint_validate = 3
lkrg.profile_enforce = 2
lkrg.profile_validate = 9
lkrg.smap_enforce = 2
lkrg.smap_validate = 1
lkrg.smep_enforce = 2
lkrg.smep_validate = 1
lkrg.trigger = 0
lkrg.umh_enforce = 1
lkrg.umh_validate = 1
```

PERFORMANCE AND SCALABILITY

❖ LKRG with default protections enabled:

CPU: Intel Xeon E-2176G @ 4.70GHz (6 Cores / 12 Threads)
OS: Ubuntu 18.04

- The newest version (0.8) has overhead around ~2.5%
  All details are available in PERFORMANCE file
- Performance impact was also comprehensively evaluated by Phoronix:
  https://www.phoronix.com/scan.php?page=article&item=lkrg-08-linux&num=1

❖ Scalability:

- We do NOT expect a significant increase in LKRG's overhead with a higher number of concurrently running processes. LKRG's process tracking database uses a hash table of RB trees with per-hash-bucket read/write-locks.

Q&A?

https://www.openwall.com/lkrg