CSCI 49381

RandCompile - Removing Forensic Gadgets from the Linux Kernel to Combat Its Analysis

Tamar Abelson, Mackenzie Eng, Karwai Kang, David Xiao

Introduction

- **Forensic Gadgets –** structural information artifacts; predictable patterns in OS memory (structures, code, strings) that forensic tools exploit to analyze a system's state.
  - e.g. `task_struct`, symbol tables, ABI constraints, order of fields, pointer graphs
- **The Problem** – New memory forensics tools can reconstruct Linux kernel data from a memory dump without debug symbols.
  - e.g. LogicMem, AutoProfile, Katana
- **Threat Scenario –** A malicious cloud hypervisor (Virtual Machine Monitor) could use these tools to spy on guest VMs – listing processes, reading kernel logs, etc., unbeknownst to the VM owner.
- **Why RandCompile –** Existing defenses (e.g., KASLR) randomize addresses, but don't remove these forensic artifacts. RandCompile introduces compile-time randomization to eliminate or encrypt those predictable markers, foiling automated analysis.

RandCompile Key Features

String & Pointer Encryption

Hide obvious constants (e.g. "swapper/0" process name) and pointer values by XOR-encrypting them in memory. Prevents easy identification of the init task or traversal of linked lists.

Expanded Data Randomization

Use structure layout randomization on many more kernel structs and break fixed layouts (e.g., don't keep next/prev pointers adjacent). Makes kernel data layout unpredictable.

Externalized printk Strings

Remove human-readable format strings from kernel binary; replace with unique IDs. Kernel log in memory becomes gibberish to an outsider, stopping log-based analysis. (Admins can reconstruct logs offline.)

Parameter Order Randomization

Randomly shuffle function call arguments in the compiled code. This breaks forensic tools' assumptions that, say, the first register = first struct field.

Bogus Parameters

Insert fake function arguments (with dummy memory reads) into functions with few parameters. Adds noise and fake "fields" so analysis tools can't tell real data from fake.

Experimental Goals

Main Questions:

Can memory analysis tools like Katana
or HyperLink reconstruct kernel structures from hardened images?

Does RandCompile introduce noticeable performance overhead?

Approach:

1. Build multiple kernel variants with/without hardening
2. Use forensic tools to analyze memory state
3. Run microbenchmarks to evaluate runtime impact

Kernel Variants Overview

Variant	Description
base	Clean vanilla Linux 5.15.63
base_ftrace	FTRACE + debug symbols enabled
nobogus	RandCompile patch, no obfuscation
bogusmem	Obfuscates memory regions
bogusargs	Corrupts pointer-based arguments
forensic_hardening	All RandCompile protections enabled

Building the Kernels

- Started with unmodified 5.15.63 vanilla kernel source
- Copied and patched source for each variant: base, base_ftrace, bogusmem, bogusargs, nobogus, forensic_hardening
- Buildroot was used to generate a bootable initrd
- Kernels built using script (`build-kernel.sh`) with randomized struct offsets and flags
- All builds tested in QEMU to ensure bootability before dumping

Build Kernel Script

```bash
#!/bin/bash
set -eux

CURDIR=$PWD

TARGETS="config_base config_base_ftrace config_bogusmem config_bogusargs config_nobogus config_forensic_hardening"

function config_base() {
    cp "../config-base" .config

    # Disable Ftrace & Kprobes: Apparantly the patch points at the start of the
    # kernel functions do not get NOPed out correctly
    scripts/config -d CONFIG_KPROBES
    scripts/config -d CONFIG_FTRACE
    scripts/config -d CONFIG_FORENSIC_HARDENING
}

function config_base_ftrace() {
    config_base
    scripts/config -e CONFIG_KPROBES
    scripts/config -e CONFIG_FTRACE
    scripts/config -e CONFIG_FUNCTION_TRACER
    scripts/config -e CONFIG_DYNAMIC_FTRACE
    scripts/config -e CONFIG_FUNCTION_PROFILER
    scripts/config -e CONFIG_FUNCTION_GRAPH_TRACER
    scripts/config -e CONFIG_FTRACE_SYSCALLS
}

function config_forensic_hardening {
    config_base
    scripts/config -e CONFIG_FORENSIC_HARDENING
}

function config_base_ftrace_no_graph() {
    config_base
    scripts/config -e CONFIG_KPROBES
    scripts/config -e CONFIG_FTRACE
    scripts/config -e CONFIG_FUNCTION_TRACER
    scripts/config -e CONFIG_DYNAMIC_FTRACE
    scripts/config -e CONFIG_FUNCTION_PROFILER
    scripts/config -d CONFIG_FUNCTION_GRAPH_TRACER
    scripts/config -e CONFIG_FTRACE_SYSCALLS
}

function config_base_ftrace_with_graph() {
    config_base
    scripts/config -e CONFIG_KPROBES
    scripts/config -e CONFIG_FTRACE
    scripts/config -e CONFIG_FUNCTION_TRACER
    scripts/config -e CONFIG_DYNAMIC_FTRACE
    scripts/config -e CONFIG_FUNCTION_PROFILER
    scripts/config -e CONFIG_FUNCTION_GRAPH_TRACER
    scripts/config -e CONFIG_FTRACE_SYSCALLS
}

function config_bogusmem() {
    config_forensic_hardening
    scripts/config -e CONFIG_RANDFUN
    scripts/config -e CONFIG_RANDFUN_USE_RANDSTRUCT_ATTRS
    scripts/config -e CONFIG_RANDFUN_NO_RANDOMIZE_ABI_FILE
    scripts/config -e CONFIG_RANDFUN_GENERATE_BOGUS_ARGS
    scripts/config -e CONFIG_RANDFUN_GENERATE_BOGUS_MEMOFFSETS
    scripts/config -d CONFIG_RANDFUN_CALLEELIST_FILE
}

function config_bogusargs() {
    config_forensic_hardening
    scripts/config -e CONFIG_RANDFUN
    scripts/config -e CONFIG_RANDFUN_USE_RANDSTRUCT_ATTRS
    scripts/config -e CONFIG_RANDFUN_NO_RANDOMIZE_ABI_FILE
    scripts/config -e CONFIG_RANDFUN_GENERATE_BOGUS_ARGS
    scripts/config -d CONFIG_RANDFUN_GENERATE_BOGUS_MEMOFFSETS
    scripts/config -d CONFIG_RANDFUN_CALLEELIST_FILE
}

function config_nobogus() {
    config_forensic_hardening
    scripts/config -e CONFIG_RANDFUN
    scripts/config -e CONFIG_RANDFUN_USE_RANDSTRUCT_ATTRS
    scripts/config -e CONFIG_RANDFUN_NO_RANDOMIZE_ABI_FILE
    scripts/config -d CONFIG_RANDFUN_GENERATE_BOGUS_ARGS
    scripts/config -d CONFIG_RANDFUN_GENERATE_BOGUS_MEMOFFSETS
    scripts/config -d CONFIG_RANDFUN_CALLEELIST_FILE
}
function build_kernel() {
    #FILENAME=$(basename $f)
    #FILENAME_CLN="${FILENAME//.config-/}"
    FILENAME=$1
    FILENAME_CLN="${FILENAME//config_/}"
    echo $FILENAME
    echo $FILENAME_CLN

    if [[ "$FILENAME_CLN" == "base" || "$FILENAME_CLN" == "base_ftrace" ]]; then
        pushd source-unpatched
    else
        pushd "source"
    fi
    # Configure kernel
    $1
    #cp $f .config
    make oldconfig
    # ./rebuild.sh &> buildlog
    make clean
    make CONFIG_SECTION_MISMATCH_WARN_ONLY=y -j$(nproc) &> $CURDIR/kernels/$FILENAME_CLN.buildlog
    cp vmlinux $CURDIR/kernels/$FILENAME_CLN.vmlinux
    cp arch/x86/boot/bzImage $CURDIR/kernels/$FILENAME_CLN.bzImage
    cp System.map $CURDIR/kernels/$FILENAME_CLN.systemmap
    popd
}

function build_rootfs() {
    curl https://buildroot.org/downloads/buildroot-2023.02.3.tar.gz -o - | tar -xz
    cd buildroot-2023*
    cp ../config-buildroot .config
    make olddefconfig
    make
    cp output/images/rootfs.cpio.gz ../kernels/
}

if [ -z "${1+x}" ] ; then
    for f in $TARGETS; do
        build_kernel $f
    done
else
    if [[ "$1" == "build-initrd" ]]; then
        build_rootfs
    else
        build_kernel $1
    fi
fi
```

Katana Setup

- Tool: Katana
- Analyzes raw memory dumps (.core)
- Reconstructs task_struct and pointer chains using static heuristics
- Requires:
  - .core dump
  - Symbol mappings
  - Layout inference

Workflow:

1. Memory dumps (.core files) obtained from QEMU VMs using dump-guest-memory
2. kallsyms_finder.py scans memory for embedded symbol strings (like .kallsyms section)
3. recover-offsets-from-dump.sh matches known kernel fields to memory structures
4. Generates layout files: <variant>.core-layout-processed
5. Katana plugins (list_procs.py, list_modules.py, etc.) use layout to identify structures
6. CR3 values and init_task pointers confirmed via GDB when not auto-recoverable

Boot QEMU using following commands:
I.e: base kernel

```
cd ~/randcompile

qemu-system-x86_64 \
  -nographic \
  -s \
  -cpu qemu64 \
  -m 1G \
  -kernel kernels/base_ftrace.bzImage \
  -initrd kernels/rootfs.cpio.gz \
  -append "console=ttyS0 nokaslr"
```

Once in, dump the VM's memory

```
[    1.790245] Run /init as init process
[    1.827073] mount (69) used greatest stack depth: 14688 bytes left
[    1.934156] mount (74) used greatest stack depth: 14400 bytes left
[    1.943787] tsc: Refined TSC clocksource calibration: 4514.866 MHz
[    1.944081] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x411443c2542, max_idle_ns:
[    1.944408] clocksource: Switched to clocksource tsc
Starting syslogd: OK
Starting klogd: OK
Running sysctl: [    2.264354] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042/s
OK
Saving 256 bits of non-creditable seed for next boot
Starting network: [    2.508608] ip (106) used greatest stack depth: 13960 bytes left
[    2.546620] ip (107) used greatest stack depth: 12296 bytes left
OK
[    2.698548] clocksource: timekeeping watchdog on CPU0: Marking clocksource 'tsc' as unstable be
large:
[    2.699354] clocksource:                       'hpet' wd_nsec: 499219650 wd_now: f0842a7 wd_las
fff
[    2.699681] clocksource:                       'tsc' cs_nsec: 498744554 cs_now: 361f7f506 cs_la
ffffffffffff
[    2.700038] clocksource:                       'tsc' is current clocksource.
[    2.700281] tsc: Marking TSC unstable due to clocksource watchdog
[    2.700917] TSC found unstable after boot, most likely due to broken BIOS. Use 'tsc=unstable'.
[    2.701141] sched_clock: Marking unstable (2671608528, 29278453)<-(2703331163, -2441604)
[    2.702406] clocksource: Not enough CPUs to check clocksource 'tsc'.
[    2.702926] clocksource: Switched to clocksource hpet

Welcome to Buildroot
buildroot login: QEMU 8.2.2 monitor - type 'help' for more information
(qemu) dump-guest-memory base_ftrace.core
```

Then recover symbol mapping and layout inferences with:

```
shuk@Shukishi:~/randcompile$ docker run --rm -v $PWD:/mnt -it randcompile-katana ¥
hon3 kal>   python3 kallsyms_finder.py /mnt/base_ftrace.core
[+] Version string: Linux version 5.15.63 (root@7ed07a17f9fb) (gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0, GNU ld (GNU Bi
nutils for Ubuntu) 2.38) #2 SMP Tue May 13 21:48:47 UTC 2025
[+] Found kallsyms_token_table at file offset 0x0253aaf0
[+] Found kallsyms_token_index at file offset 0x0253ae70
[+] Found kallsyms_markers at file offset 0x0253a2e8
[+] Found kallsyms_names at file offset 0x0238a9b8
[+] Found kallsyms_num_syms at file offset 0x0238a9b0
[i] Negative offsets overall: 99.7917 %
[i] Null addresses overall: 0.00152027 %
[+] Found kallsyms_offsets at file offset 0x0230a218
shuk@Shukishi:~/randcompile$
```

```
shuk@Shukishi:~/randcompile$ docker run --rm -v $PWD:/mnt -it randcompile-katana ¥
luation/>   evaluation/recover-offsets-from-dump.sh ¥
 /mnt/ba>   /mnt/base_ftrace.core ¥
>   db/fields.v5.15.5-def.txt ¥
>   db/structinfo.v5.15.5-def.json
Number of mappings: 65711
Skipping address ffff88800009b000
Skipping address ffffc9000000b000
Skipping address ffffc9000001d000
Skipping address ffffc90000180000
Skipping address fffffffff5fb000
Skipping address fffffffff5fc000
Picked up _JAVA_OPTIONS: -Xmx12g
Picked up _JAVA_OPTIONS: -Xmx12g
Picked up _JAVA_OPTIONS: -Xmx12g
openjdk version "17.0.15" 2025-04-15
OpenJDK Runtime Environment (build 17.0.15+6-Debian-1deb12u1)
OpenJDK 64-Bit Server VM (build 17.0.15+6-Debian-1deb12u1, mixed mode)
INFO  Using log config file: jar:file:/ghidra_10.0.4_PUBLIC/Ghidra/Framework/Generic/lib/Generic.jar!/generic.log4j.xml
(LoggingInitialization)
INFO  Using log file: /root/.ghidra/.ghidra_10.0.4_PUBLIC/application.log (LoggingInitialization)
INFO  Loading user preferences: /root/.ghidra/.ghidra_10.0.4_PUBLIC/preferences (Preferences)
```

BASE Kernels Results

```
Extracted offsets:
    task_struct->tasks.next: 0x3e8
    task_struct->state:      0x18
    task_struct->pid:        0x4f0
    task_struct->comm:       0x6c8
    task_struct->mm:         0x438
    mm_struct->pgd:          0x50

Attributes found, skipping for now

    task_struct->cred:       0x6b8
    cred->uid                0x04

0xffffffff82614580
PID: 0 (swapper/0        ) State: 0x0 MM 0x0 UID 0x00
Task struct @ 0xffffffff82614580 CR3 ([31m0x0000000000000000[39m)
0xffff888003088000
PID: 1 (init             ) State: 0x1 MM 0xffff888003e0c00 UID 0x00
Task struct @ 0xffff888003088000 CR3 (0xffff888003af2000          0x3af2000)
0xffff888003088d80
PID: 2 (kthreadd         ) State: 0x1 MM 0x0 UID 0x00
Task struct @ 0xffff888003088d80 CR3 ([31m0x0000000000000000[39m)
0xffff888003089b00
PID: 3 (rcu_gp           ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff888003089b00 CR3 ([31m0x0000000000000000[39m)
0xffff88800308a880
PID: 4 (rcu_par_gp       ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800308a880 CR3 ([31m0x0000000000000000[39m)
0xffff88800308b600
PID: 5 (netns            ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800308b600 CR3 ([31m0x0000000000000000[39m)
0xffff88800308c380
PID: 6 (kworker/0:0      ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800308c380 CR3 ([31m0x0000000000000000[39m)
0xffff88800308d100
PID: 7 (kworker/0:0H     ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800308d100 CR3 ([31m0x0000000000000000[39m)
0xffff88800308de80
PID: 8 (kworker/u2:0     ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800308de80 CR3 ([31m0x0000000000000000[39m)
0xffff88800308ec00
PID: 9 (kworker/0:1H     ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800308ec00 CR3 ([31m0x0000000000000000[39m)
0xffff888003098000
PID: 10 (mm_percpu_wq     ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff888003098000 CR3 ([31m0x0000000000000000[39m)
0xffff888003098d80
PID: 11 (ksoftirqd/0      ) State: 0x1 MM 0x0 UID 0x00
Task struct @ 0xffff888003098d80 CR3 ([31m0x0000000000000000[39m)
0xffff888003099b00
PID: 12 (rcu_sched        ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff888003099b00 CR3 ([31m0x0000000000000000[39m)
0xffff88800309a880
PID: 13 (migration/0      ) State: 0x1 MM 0x0 UID 0x00
Task struct @ 0xffff88800309a880 CR3 ([31m0x0000000000000000[39m)
0xffff88800309b600
PID: 14 (cpuhp/0          ) State: 0x1 MM 0x0 UID 0x00
Task struct @ 0xffff88800309b600 CR3 ([31m0x0000000000000000[39m)
0xffff88800309c380
PID: 15 (kdevtmpfs        ) State: 0x1 MM 0x0 UID 0x00
Task struct @ 0xffff88800309c380 CR3 ([31m0x0000000000000000[39m)
0xffff88800309d100
PID: 16 (inet_frag_wq     ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800309d100 CR3 ([31m0x0000000000000000[39m)
0xffff88800309de80
PID: 17 (kauditd          ) State: 0x1 MM 0x0 UID 0x00
Task struct @ 0xffff88800309de80 CR3 ([31m0x0000000000000000[39m)
0xffff88800309ec00
PID: 18 (oom_reaper       ) State: 0x1 MM 0x0 UID 0x00
Task struct @ 0xffff88800309ec00 CR3 ([31m0x0000000000000000[39m)
0xffff88800030c8000
PID: 19 (writeback        ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff8880030c8000 CR3 ([31m0x0000000000000000[39m)
0xffff8880030c8d80
PID: 20 (kcompactd0       ) State: 0x1 MM 0x0 UID 0x00
Task struct @ 0xffff8880030c8d80 CR3 ([31m0x0000000000000000[39m)
0xffff88800320c380
PID: 42 (kblockd          ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800320c380 CR3 ([31m0x0000000000000000[39m)
0xffff88800320d100
PID: 43 (blkcg_punt_bio   ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800320d100 CR3 ([31m0x0000000000000000[39m)
0xffff88800320b600
PID: 44 (kworker/0:1      ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800320b600 CR3 ([31m0x0000000000000000[39m)
0xffff88800320a880
PID: 45 (ata_sff          ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800320a880 CR3 ([31m0x0000000000000000[39m)
0xffff888003209b00
PID: 46 (md               ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff888003209b00 CR3 ([31m0x0000000000000000[39m)
0xffff888003208d80
PID: 47 (rpciod           ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff888003208d80 CR3 ([31m0x0000000000000000[39m)
0xffff888003208000
PID: 48 (kworker/u3:0     ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff888003208000 CR3 ([31m0x0000000000000000[39m)
0xffff88800320de80
PID: 49 (xprtiod          ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800320de80 CR3 ([31m0x0000000000000000[39m)
0xffff88800320ec00
PID: 50 (cfg80211         ) State: 0x402 MM 0x0 UID 0x00
Task struct @ 0xffff88800320ec00 CR3 ([31m0x0000000000000000[39m)
0xffff888003206c00
```

Hardened Kernel Result

```
Extracted offsets:
    task_struct->tasks.next: 0x3e8
    task_struct->state:      0x18
    task_struct->pid:        0x4f0
    task_struct->comm:       0x08
    task_struct->mm:         0x438
    mm_struct->pgd:          0x50

Attributes found, skipping for now

    task_struct->cred:       0x6b8
    cred->uid                0x04

0xffffffff82614580
PID: 0 (                 ) State: 0x0 MM 0x0 UID 0x00
Task struct @ 0xffffffff82614580 CR3 ([31m0x0000000000000000[39m)
0xe0ade7347b67045e
Traceback (most recent call last):
  File "/katana/list_procs.py", line 88, in <module>
    mm = pointer(view.get_virt(cur + mm_off, view.pointer_size))
                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/katana/elfview.py", line 298, in get_virt
    raise NotMapped("NotMapped: Virtual Address - 0x{:016x}".format(addr))
elfview.NotMapped: NotMapped: Virtual Address - 0xe0ade7347b670896
```

Hardened kernel structure did not match expected structure for Katana, therefore broke Katana's analysis abilities.

Katana Results

Kernel Variant	Katana Output	Notes
base	✅ Full process list	Baseline
base_ftrace	✅ Full process list	With debugging enabled
nobogus	✅ Full process list	Works like basemodel
bogusargs	❌ Crashed	Garbage offset detection
bogusmem	❌ Crashed	Traversal failure
forensic_hardening	❌ Crashed	All protections effective

Hyperlink Workflow

QEMU launched with -s -S for remote debugging

Attached with GDB to paused VM state

Loaded vmlinux for the matching build

Retrieved &init_task.tasks and passed address to hyperlink-ps

Base kernel showed successful process walk via pointer chains

Hardened variants showed breakage: circular chains, invalid pages, or unreachable init_task

Demonstrates real-time forensic evasion

Boot into QEMU with

```
qemu-system-x86_64 \
  -s \
  -m 1G \
  -kernel kernels/base_ftrace.bzImage \
  -initrd kernels/rootfs.cpio.gz \
  -append "console=ttyS0 nokaslr" \
  -nographic
```

Then on separate Terminal, attach onto GDB with:

```
gdb
(gdb) file kernels/base_ftrace.vmlinux
(gdb) target remote :1234
(gdb) source hyperlink-gdb/hyperlink.py
(gdb) p &init_task.tasks
(gdb) hyperlink-ps <the_address>
```

Example Output:

```
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file kernels/base_ftrace.vmlinux
Reading symbols from kernels/base_ftrace.vmlinux...
(gdb) target remote :1234
Remote debugging using :1234
Enable debuginfod for this session? (y or [n])
amd_e400_idle () at arch/x86/kernel/process.c:780

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
warning: 780    arch/x86/kernel/process.c: No such file or directory
(gdb) source hyperlink-gdb/hyperlink.py
(gdb) p &init_task.tasks
$1 = (struct list_head *) 0xffffffff82814d30 <init_task+1008>
(gdb) hyperlink-ps 0xffffffff82814d30
Looking for kernel pointers around 0xffffffff82814d30
-d30 0xffffffff82814000: ffff82f9e000007f -> <invalid addr>
-d10 0xffffffff82814020: ffff8280400001ff -> <invalid addr>
-c50 0xffffffff828140e0: ffffffff828140e0 -> ffffffff828140e0 [circular len=0]
-c48 0xffffffff828140e8: ffffffff828140e0 -> ffffffff828140e0 [circular len=0]
-c30 0xffffffff82814100: ffffffff8255ad4a -> 54002f3d454d4f48
-c28 0xffffffff82814108: ffffffff8255ad51 -> 6e696c3d4d524554
-b10 0xffffffff82814220: ffffffff8255ad24 -> 622f0074696e692f
-b08 0xffffffff82814228: ffff88803f68074e -> 726c73616b6f6e
-a00 0xffffffff82814330: ffffffff8255ad24 -> 622f0074696e692f
-970 0xffffffff828143c0: ffffffff8255ad5c -> 6c6c616374696e69
-968 0xffffffff828143c8: ffffffff82583852 -> 705f5f00636e7566
```

HyperLink Results

Base Kernel Results

```
(gdb) file kernels/base_ftrace.vmlinux
Reading symbols from kernels/base_ftrace.vmlinux...
(gdb) target remote :1234
Remote debugging using :1234
Enable debuginfod for this session? (y or [n])
amd_e400_idle () at arch/x86/kernel/process.c:780

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
warning: 780    arch/x86/kernel/process.c: No such file or directory
(gdb) source hyperlink-gdb/hyperlink.py
(gdb) p &init_task.tasks
$1 = (struct list_head *) 0xffffffff82814d30 <init_task+1008>
(gdb) hyperlink-ps 0xffffffff82814d30
Looking for kernel pointers around 0xffffffff82814d30
-d30 0xffffffff82814000: ffff82f9e000007f -> <invalid addr>
-d10 0xffffffff82814020: ffff8280400001ff -> <invalid addr>
-c50 0xffffffff828140e0: ffffffff828140e0 -> ffffffff828140e0 [circular len=0]
-c48 0xffffffff828140e8: ffffffff828140e0 -> ffffffff828140e0 [circular len=0]
-c30 0xffffffff82814100: ffffffff8255ad4a -> 54002f3d454d4f48
-c28 0xffffffff82814108: ffffffff8255ad51 -> 6e696c3d4d524554
-b10 0xffffffff82814220: ffffffff8255ad24 -> 622f0074696e692f
-b08 0xffffffff82814228: ffff88803f68074e -> 726c73616b6f6e
-a00 0xffffffff82814330: ffffffff8255ad24 -> 622f0074696e692f
-970 0xffffffff828143c0: ffffffff8255ad5c -> 6c6c616374696e69
-968 0xffffffff828143c8: ffffffff82583852 -> 705f5f00636e7566
-950 0xffffffff828143e0: ffffffff8255fb70 -> 6d5f736900746e69
-948 0xffffffff828143e8: ffffffff8258332d -> 36785f5f00746572
-910 0xffffffff82814420: ffffffff8255ad5c -> 6c6c616374696e69
-908 0xffffffff82814428: ffffffff82583852 -> 705f5f00636e7566
-8d0 0xffffffff82814460: ffffffff8255ad67 -> 6c5f617461645f5f
-8c8 0xffffffff82814468: ffffffff825bd4f7 -> 7325006c6576656c
-890 0xffffffff828144a0: ffffffff81000aa0 -> fb894853d6894855
-888 0xffffffff828144a8: ffffffff8115cea0 -> 20d8878b4855
-880 0xffffffff828144b0: ffffffff8115cea0 -> 20d8878b4855
-878 0xffffffff828144b8: ffffffff8115cea0 -> 20d8878b4855
-870 0xffffffff828144c0: ffffffff81000a50 -> fb894853d6894855
-868 0xffffffff828144c8: ffffffff8115cea0 -> 20d8878b4855
-860 0xffffffff828144d0: ffffffff8115cea0 -> 20d8878b4855
-858 0xffffffff828144d8: ffffffff8115cea0 -> 20d8878b4855
-850 0xffffffff828144e0: ffffffff81000a00 -> fb894853d6894855
-848 0xffffffff828144e8: ffffffff8115cea0 -> 20d8878b4855
-840 0xffffffff828144f0: ffffffff8115cea0 -> 20d8878b4855
--Type <RET> for more, q to quit, c to continue without paging--
```

Hardened Kernel Results

```
(gdb) file kernels/forensic_hardening.vmlinux
Reading symbols from kernels/forensic_hardening.vmlinux...
(gdb) target remote :1234
Remote debugging using :1234
Enable debuginfod for this session? (y or [n])
amd_e400_idle () at arch/x86/kernel/process.c:780

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
warning: 780    arch/x86/kernel/process.c: No such file or directory
(gdb) source hyperlink-gdb/hyperlink.py
(gdb) p &init_task.tasks
There is no member named tasks.
(gdb) p &init_task.tasks
There is no member named tasks.
(gdb) p &init_task.tasks
There is no member named tasks.
(gdb) p &init_task.tasks
There is no member named tasks.
(gdb) _
```

On Base Kernels:

- Successfully recovered process list
- Pointer chains interpreted correctly
- init_task.tasks visible and accessible

On Hardened Kernels(forensic_hardening, bogusmem, etc)

- GDB: "No member named tasks"
- HyperLink failed to dereference task list
- Kernel obfuscation broke runtime inspection

Conclusion: RandCompile also defeats runtime forensic tools.

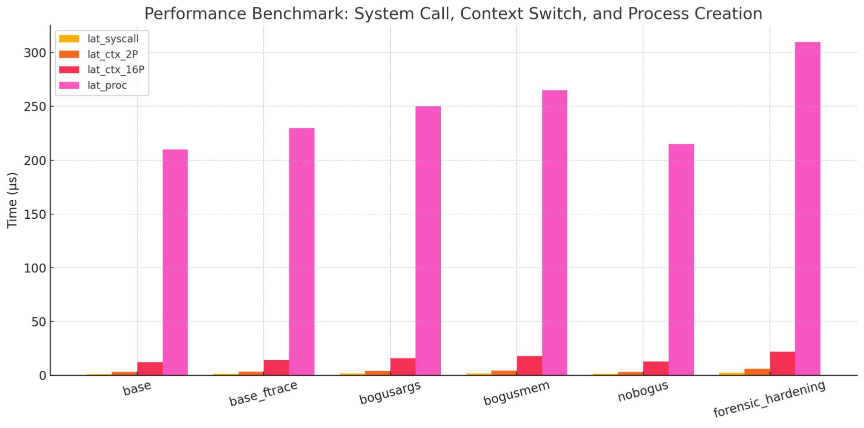
LogicMem Failure

- LogicMem did not work even on the execution of the base kernel. Expected layout inputs (e.g., core-layout, core-kallsym, and core-cr3) were present. However, `LogicMem` relies on specific structure assumptions (task_struct, mm_struct).
- RandCompile introduces randomized field padding and obfuscates init_task, breaking assumptions. Matching the authors of RandCompile, we also failed to extract meaningful results.
- The root cause of this failure is due to LogicMem operating in a very rigid ruleset, which are invalidated by SLR (Structure Layout Randomization).
- Furthermore, input format of LogicMem is not documented by authors of the tool, and a code analysis did not yield any results.

Performance Impact

- Benchmarks run: `syscall` latency, context switch, fork+exec
- forensic_hardening shows measurable overhead vs. base

Variant	lat_syscall (µs)	lat_ctx_2P (µs)	lat_ctx_16P (µs)	lat_proc (µs)
base	1.26	3.10	12.32	212.8
base_ftrace	1.40	3.40	14.27	234.0
bogusargs	1.68	4.15	16.04	251.3
bogusmem	1.74	4.46	17.84	265.0
nobogus	1.25	3.20	13.08	217.5
forensic_hardening	2.43	6.23	22.05	313.4



Conclusion

Key Takeaway: RandCompile successfully hardens the Linux kernel against memory forensic analysis. With forensic gadgets removed, even powerful introspection tools can't easily spy on a running VM. This enhances privacy and security for cloud users.

Minimal Trade-off: The security gains come at little cost – negligible performance impact and no need to modify running system behavior (changes are at compile-time).

Possible Improvements:

1. **Cross-Platform Application:** Explore applying RandCompile's approach beyond Linux – e.g., to other OS kernels (Windows, *BSD) or hypervisors – to protect against similar memory analysis on those platforms.
2. **Combine with Hardware Security:** Use RandCompile alongside hardware encryption like AMD SEV. Even if memory is encrypted, RandCompile provides a safety net against any weaknesses by obfuscating the content itself
3. **Fine-tuning:** Investigate any edge cases, ensure stability with more kernel modules, and refine randomization (e.g., more sophisticated encryption or diversification techniques) as needed.

Impact: This project means to show a novel defense-in-depth strategy. By preemptively removing the "hooks" that attackers rely on, we can stay a step ahead of forensic analysis techniques and protect sensitive systems moving forward.

REFERENCES

1. F. Franzen, A. C. Wilhelmer, and J. Grossklags, “RandCompile: Removing Forensic Gadgets from the Linux Kernel to Combat its Analysis,” Proc. ACSAC 2023, pp. 14–27, Dec. 2023. DOI: 10.1145/3627106.3627197.
2. F. Franzen, T. Holl, M. Andreas, J. Kirsch, and J. Grossklags, “Katana: Robust, Automated, Binary-Only Forensic Analysis of Linux Memory Snapshots,” Proc. RAID 2022, pp. 299–318, Oct. 2022. DOI: 10.1007/978-3-031-16919-1_14.
3. IBM Developer, “KASLR support (Linux on Z),” IBM Knowledge Center, 2021. [Online]. Available: https://www.ibm.com/docs/en/linux-on-systems?topic=shutdown-kaslr
4. Z. Qi, Y. Qu, and H. Yin, “LogicMem: Automatic Profile Generation for Binary-Only Memory Forensics via Logic Inference,” in Proc. NDSS 2022, Apr. 2022.
5. F. Pagani and D. Balzarotti, “AutoProfile: Towards Automated Profile Generation for Memory Analysis,” ACM Trans. Privacy and Security, vol. 25, no. 1, pp. 1–26, Feb. 2022. https://doi.org/10.1145/3485471
6. Career Technology Cyber Security, “Volatility (memory forensics framework overview),” Medium, Jul. 2023. [Online]. Available: https://medium.com/@careertechnologymiraroad/volatility-978e32316616.