WEAPONIZING HYPERVISORS TO FIGHT & BEAT CAR & MEDICAL DEVICE ATTACKS

Ali Islam – CEO Numen Inc

Dan Regalado – DanuX – CTO Numen Inc
AGENDA

Basic Concepts

Embedded Environment

Demo – Attacks & Use cases

Q&A
Hypervisors & Strong Trending

**INTEL® DEVELOPMENT PLATFORM FOR IN-VEHICLE EXPERIENCES**

<table>
<thead>
<tr>
<th>Infotainment</th>
<th>Cluster</th>
<th>ADAS visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMI</td>
<td>HMI</td>
<td>HMI</td>
</tr>
<tr>
<td>Frameworks and middleware</td>
<td>Frameworks and middleware</td>
<td>Frameworks and middleware</td>
</tr>
<tr>
<td>Android* or Linux* and BSP</td>
<td>RTOS and BSP</td>
<td>RTOS and BSP</td>
</tr>
<tr>
<td>SoC drivers</td>
<td>SoC drivers</td>
<td>SoC drivers</td>
</tr>
</tbody>
</table>

**Developer tools**

To speed development, Intel and its partners provide third-party OS support and a comprehensive set of developer tools, including:

- The Intel® C++ Compiler, Intel® VTune™ Amplifier for Systems, and Intel® Graphics Performance Analyzer
- Reference stacks, including an IVI middleware and automotive boot loader
- Reference OS support for Linux*, Android*, Green Hill Integrity*, QNX CAR Platform for Infotainment*, and Wind River Helix*
- Hypervisors for multi-operating systems from QNX, Green Hill, and Mentor Graphics
- Performance-tuning tools for the Intel® architecture-based CPU and GPU and complete hardware development vehicles

**Figure 2.** The Intel® development platform for in-vehicle experiences drives the consolidation of systems in the vehicle.

By 2020, the two companies look to install an infotainment solution in millions of vehicles throughout China.
Agent-less vs Agent (AV)

• Sophisticated Invisibility (VMI) - Cat and mouse game
• No messing up the actual device functionality
• Helps with regulations and certifications
Let's start the Journey

Setting up the Environment on a Zynq UltraScale+ MPSoC ZCU 102
Zynq UltraScale+ MPSoC ZCU 102

- DDR4 – 4 GB
- Quad-core Cortex A-53
- Dual-core Cortex R5F

Booting the board with JTAG

• Using Xilinx System Debugger CLI (xsdb) which reads a tcl file

```bash
prec run (fsbl uboot b131 pmu) {
    targets -set -filter (name =~ "PMU" || name =~ "PSU")
    # According to AR 67871
    mvr 0x1f120000 0x14000000
    mvr 0x77e0000 0x1ff
    mask_write 0x1b0000 0x501 0x0
    after 1000
    targets -set -filter (name =~ "MicroBlaze PMU")
    dow $pmu
can
    after 1000
download and run FSBL
    targets -set -filter (name =~ "Cortex-A53 #0")
downloading FSBL
dow $fsbl
can
    after 1000
    stop
    # Set the GEMS to be NS
    mvr 0x1f240000 0x492
    mvr 0x1f240000 0x492
    # Unmask SWI interrupts.
    mvr 0x7ff90001 0x1f
dow $uboot
dow b131
can
}
```

PMUFW – Setup clock and platform management
FSBL – First Stage Bootloader – Initializes U-Boot
U-Boot – Boots the Hypervisor, Kernel and rootfs
Bl31 – ARM Trusted Firmware

```
set fsbl /home/builder/ZCU102_BSP_2018.3/images/zynqmp_fsbl.elf
set uboot /home/builder/ZCU102_BSP_2018.3/images/u-boot.elf
set b131 /home/builder/ZCU102_BSP_2018.3/images/b131.elf
set pmu /home/builder/ZCU102_BSP_2018.3/images/pmuufw.elf
```

X-Boot 2018.01 (Dec 06 2018 - 09:36:05 +0000) Xilinx ZynqMP ZCU102 rev1.0

If 0: ready
0DRAM: Hit any key to stop autoboost: 0
ZynqMP> £
Preparing Device Tree Blob (DTB) `xen.dtb` file (dts below):

```plaintext
chosen {
  bootargs = "earlycon console=ttyS0,115200 clk_ignore_unused";
  stdout-path = "serial10:115200n8";
  @address-cells = <0x7>;
  @size-cells = <0x1>;
  xen,xen-bootargs = "console=ttb0 tttb=serial0 dom0_mem=1G bootscrub=0 dom0_max_vpus=2 sched=credit alphabet=1";
  xen,dom0-bootargs = "console=hvc0 earlycon=xen earlyprintk=xen maxpus=1 clk_ignore_unused root=/dev/mmcblk0p3 rw rootwait";

  dom0 {
    compatible = "xen-linux-zimage", "xen, multiboot-module";
    reg = <0x0 0x80000 0x3100000>;
  }
};
```

Preparing the hypervisor:

```
# mkimage -A arm64 -T kernel -a 0x14000000 -e 0x14000000 -C none -d xen-zcu102-zynqmp xen.ub
ZynqMP> tftp 0x1380000 xen.dtb
ZynqMP> tftp 0x800000 Image-2018.3
ZynqMP> tftp 0x140000 xen.ub
ZynqMP> bootm 0x14000000 - 0x1380000
```

```plaintext
root@zcu102:~# uname -a
Linux zcu102 4.14.0-xilinx-v2018.3 #1 SMP Thu Dec 6 10:01:26 UTC 2018 aarch64 GNU/Linux
```
Building the rootfs

- PetaLinux: Xilinx-based and therefore not universal
- Yocto: Universal but builds a Busybox limited rootfs
  - Real pain to compile new libraries
- Debootstrap: Way to go, Debian-based FileSystem 😊
Dev environment

- You do not want to make changes directly on the board
- Schroot to the rescue
  - Chroot into the rootfs but from a mounting point via QEMU
Let’s get the damn ARM Syscalls out!
VMI & Semantic Gap

Understand meaning using OS specific knowledge

linux_name = 0x4f0;
linux_tasks = 0x280;
linux_mm = 0x2d0;
linux_pid = 0x334;
linux_pgd = 0x40;

https://notes.shichao.io/lkd/ch3/
Kernel Symbol Value Example

status_t vmi_read_ksym(vmi_instance_t vmi, const char *sym, size_t count, void *buf, size_t *bytes_read);

http://libvmi.com/docs/gcode-intro.html
Single Stepping

- Hardware Breakpoints
- Software Breakpoints - CPU assisted
- Software breakpoints – No CPU Assistance

Extended Page table(s)

EPT pointer (EPTP) is stored in the Virtual Machine Control Structure (VMCS) - A per VM data struct in the memory and managed by VMM.
Multiple p2m Translations

Virtual to VM PA

VM Virtual Address

VM Physical Address

EPTP 1

EPTP 2

Machine Physical Address

Virtual to VM PA

Extend Page Table Entry (epte) struct from Xen code

```
#include <asm/1387.h>
#include <asm/hvm/support.h>
#include <asm/hvm/trace.h>
#include <asm/hvm/vmcs.h>

typedef union {
    struct {
        u64 r : 1, /* bit 0 - Read permission */
        w : 1, /* bit 1 - Write permission */
        x : 1, /* bit 2 - Execute permission */
        emt : 3, /* bits 5:3 - EPT Memory type */
        ipat : 1, /* bit 6 - Ignore PAT memory type */
        sp : 1, /* bit 7 - Is this a superpage? */
    }
} epte;
```
Multiple p2m Translations (continued)

VM Virtual Address → Virtual to VM PA

VM Physical Address

EPT 1

Machine Physical Address (MPA)

EPT 2

MPA in second memory copy

VM PA to Machine PA
Single Stepping on ARM

Default Memory View

0xD4000003 (BP)
Instruction 2
Instruction 3
Instruction n

Single Stepping View

Instruction 1
0xD4000003 (BP)
Instruction 3
Instruction n

BP = Breakpoint = SMC
Hooking and Syscall Monitoring on ARM

Singlestep
Make sure to singlestep in order to execute the original functionality

Clean
After you are done, make sure to remove all hooks and exit VMI. Otherwise the VM might crash or become unstable

Add & Register Hook
vmi_register_event() & Write to memory 0xD4000003 (SMC) at the start of each API function.

Callback
Do you analysis when the control gets to your registered callback.

vmi_destroy();
Syscalls Monitoring in ARM

(ARM-Syscalls.mp4)
Attacks and Detection scenarios
Memory corruption attack

Benign activity

Malicious activity

Exit gracefully

Shell spawn at the end

Easy sequence-based detection
Shellcode execution delay

- Syscall monitoring cannot be on all the time
- Not using syscall (sleep) to delay execution
- Traditional AV challenge
Solution approach

- Create a “triggered memory view” hooking only suspicious syscalls: execve, connect, clone, etc all the time
- As soon as the shellcode spawns, full hooking on that process is enabled!
Malware hypervisor-aware

- The malware is able to read kernel memory and identify SMC hooks
  - Stops running or wipes the system!
- Even in some conditions is able to remove the hooks!
  - Worst scenario, detection bypass!
Stealthiness using memory views
Policy Enforcement – Network Use Case

Once you have a good handle on Virtual Machine Introspection, there are many possibilities.

1) Traverse a task list and see if there is any socket handle for a particular task struct
   1.1) A socket is a special type of file. So check if there is any additional file handle

2) Hook the network related APIs (e.g. connect).
   2.1) More active approach vs the passive one in step 1.
Our patent pending Numen Adaptive Monitoring (NAM) is a combination of different techniques to achieve exceptional performance.
Remediation

- Its not easy to remediate from outside without putting any agent inside. Let’s say kill a process.

- How about manipulating with one of the frequently called APIs?

- Maybe make one of the string parameter NULL?

- Just a basic way. There can be other more mature ways.
PRACTICAL RECOMMENDATIONS FOR END TO END SYSTEM

- Software Breakpoints
- Efficient Single Stepping Mechanism
- Event Mechanism
- Efficient translations caching
- Multiple mappings support for p2m (physical to machine)
- Memory page permissions management
Releasing tool to the public

- Tool to perform syscall monitoring for ARM & Intel 😊
- All files needed to setup a working environment:
  - Booting the board: zynqmp_fsbl.elf, u-boot.elf, bl31.elf, pmufw.elf
  - Test: ARM64-based malware and exploit samples.
- Dropbox link: xxxxxxxxxxxxxxxxxxxxxxxxxxx
Takeaways

• “Smart” Hypervisors on ARM are needed, not only for isolation

• ARM Syscall Hooking is great achievement but just the beginning, the detection strategies is what makes the difference

• Switching between memory views for detection strategies is a new way to detect maliciousness from VMI
Special Thanks

• Stefano Stabellini: For his great help on Xen troubleshooting

• Matt Leinhos: For his great features on ARM/Intel VMI

• For those 3 of you guys, you know who you are 😊

Without you, no way to complete this effort
Q & A

@Ali_Islam_Khan
@danuxx