Live Hacking: Hardware-enforced Virtualization of a Linux Home Gateway

Michael Hohmuth, Adam Lackorzynski
Kernkonzept GmbH
Dresden, Germany
michael.hohmuth@kernkonzept.com

Cesare Garlati
prpl Foundation
Santa Clara, CA, USA
cesare@prplFoundation.com

***Abstract* — Trust and security are central to embedded computing as network devices - such as home gateways - have become the first line of defense for the IoT devices connected to the smart home. In this paper, we present a virtualization-based approach to securing home gateway while preserving functionality and performance.**

Keywords—home gateway; router; virtualization; security; live hacking; linux, hypervisor, microkernel, IoT, Internet

I. Introduction

Trust and security have never been more important to the embedded computing world, especially when it comes to network devices, such as home gateways, that are the first line of defense for the IoT devices connected to the smart home [4]. In 2017, a plethora of stories have confirmed that these devices are fundamentally broken from a security perspective.

At embedded world 2017, we hosted a successful live demonstration showing attendees how the prpl Foundation’s new approach to embedded computing security works in an industrial Internet scenario – that is, secure remote control of a robotic arm. We are back this year with a new demonstration designed to show the application of the new capabilities of the prplSecurity Framework 2.0 – as implemented in the open-source L4Re hypervisor – to a different real word scenario: a typical Linux-based Internet router, deployed as a home gateway, that connects home computers, smartphones, IoT devices and other smart devices to the Internet.

Linux is the dominant operating-system used for Internet routing devices. Optimized Linux distributions, like OpenWrt, add to the vanilla kernel a configuration system, additional applications including IP-telephony, network-printing services, VPN, media streaming and a browser-based administrative UI. Although optimized for minimal system footprint, many components of the resulting software stack are complex and inevitably enlarge the attack surface. The Linux kernel alone is comprised of millions of lines of code. And a large part of the code runs in privileged CPU mode or with elevated OS rights. This is a major security concern especially because many home-gateway vendors have shown marginal attention to securing devices in the field. Availability of security updates is sporadic and the patching process in not fully automated as it typically requires end-user intervention. As a result, home routers present a large attack surface and many exploitable vulnerabilities. This puts the security of personal data, smart-home applications and IoT devices at risk. Given the sheer number of connected devices, it also represents a great risk for the Internet infrastructure itself as shown by the recent DDOS attacks such as Mirai and similar.

This unsatisfying state of home-router security has led the telecom industry to look for solutions that guarantee availability, security, and remote patching of home routers independently from the Linux operating system itself. One such approach is to use a software partition that can restart or even update the main OS from a clean state, and that is isolated from the main OS kernel and software. This isolation can be implemented in hardware using a separate CPU to run the update/restart process, or more cost-effectively in software using an array of hardware/software virtualization technologies.

This paper shows the application of a light-weight type-1 hypervisor to isolate the router software, including the Linux kernel and the user-land applications, into a virtual machine (VM). The secure update/restart process runs in a separate VM completely isolated from the rest of the system. Our work is based on the open-source L4Re hypervisor. This hypervisor leverages the hardware virtualization support of modern CPUs to provide isolation and efficiency and, most importantly, the ability to run unmodified Linux Software.

The live demonstration starts by downloading the necessary code from various open source repositories. We then configure, build and install a new hardened firmware image to create multi-domain security via hardware virtualization. We then launch in real time several network attacks to exploit known vulnerabilities of the Linux instance. This shows how the breach is contained to the target VM, while the system critical components remain unaffected. This session is aimed at security architects, penetration testers and anyone who wants to see how a real-world attack is conducted and how hardware virtualization can effectively mitigate the overall impact on the system.

This paper is organized as follows. In Section II, we discuss virtualization as a security mechanism and introduce virtualization concepts such as full virtualization, paravirtualization, and containerization. Section III introduces the open source L4Re hypervisor. Section IV explains the home-router setup referenced throughout this paper. Section V outlines the live-hacking scenario we present during the live demonstration, before we conclude the paper in Section VI.

.

II. Virtualization and Security

In general, virtualization is the concept of abstracting away from the specifics of an underlying hardware mechanism. For example, most OSes offer virtual memory as an abstraction of physical memory, providing programs with the illusion of an abstract computer with a separate, isolated memory space. In this paper, we use virtualization in a narrower, more specific definition: as a mechanism for providing *virtual machines* (VMs) that provide user software with the illusion of running on a separate physical computer.

Virtualization can be provided on various levels. The Linux already comes with several virtualization mechanisms, including *control groups and containers*, which provide the illusion of several isolated Linux instances although all instances share the same Linux kernel, and *kernel virtual machines* (KVM), which provides VMs that look like physical computers and that run unmodified OS kernels (such as another Linux kernel) as unprivileged VM guests. These mechanisms all share the property that all VMs must trust the hosting Linux kernel and Linux-based OS, which can be undesirable from a security perspective.

The alternative solution is to deprivilege all Linux instances by running them on top of a small hypervisor such as the L4Re hypervisor. Depending on which critical services these Linux guests provide, it is possible to remove the Linux OS from the critical trusted path, or *trusted computing base* (TCB), of a security-sensitive application. If the hypervisor is small, the critical application’s TCB can be several orders of magnitude smaller than the Linux kernel alone.

Full virtualization (allowing unmodified OS kernels to run in VMs) can greatly benefit from virtualization assists provided in hardware by the platform’s CPU. Fortunately, modern server, desktop, and embedded CPUs all provide hardware-assisted virtualization features (i.e., nested paging, interrupt-controller virtualization, and I/O-MMUs). Where these hardware features are not available, either hypervisors need to resort to costly emulation of VM features that allow unmodified guest OSes to run; or, guest OSes need to be modified to be able to run as VM guests on top of the hypervisor. In the latter case, the guest OS kernel is said to be paravirtualized*;* of course, this is feasible only for OSes for which source code is available. There is also a hybrid approach in which the OS kernel proper runs unmodified, using hardware-assisted virtualization, but certain device drivers use hypervisor APIs directly (instead of emulated device interfaces) for performance reasons. The VirtIO set of APIs is a well-known example for such a paravirtualized device API.

Hardware-assisted virtualization and paravirtualization are conceptually similar when implemented in the same software layer. Minor differences in complexity and attack surface mostly stem from additional emulation layers needed to provide the physical-machine illusion for full, hardware-assisted virtualization.

III. The L4Re Hypervisor and Operating System

The L4Re system is a light-weight, microkernel-based, real-time operating system with support for hardware-assisted virtualization and paravirtualization [7,8]. The system components include:

- The L4Re Hypervisor
- The L4Re Runtime Environment, a POSIX-like programming environment for implementing native, trusted L4Re microapps
- A VMM component for hardware-assisted virtualization of unmodified guest OSes (i.e., Linux and FreeRTOS)
- L4Linux, a paravirtualized Linux kernel

The L4Re system supports many platforms including x86, ARM and MIPS architectures in 32-bit and 64-bit mode. Hardware-assisted virtualization and device-memory virtualization (IOMMUs) are also supported if available. Additionally, experimental support is available for PowerPC and Sparc. L4Re is easily portable: developing a board-support package (BSP) for a new platform usually takes few developer-days.

L4Re is a mature OS that has been in development since 1997. Originally developed at TU Dresden, it has recently obtained vast commercial uptake and support. The L4Re software is licensed under GNU GPLv2 and it is available for download at https://www.kernkonzept.com. A dual-licensing schema is available for commercial applications if desired.

The L4Re system aims at minimizing each application's or VM's TCB. The hypervisor is a classic L4 microkernel as it implements only those OS mechanisms that are required to securely implement isolation (i.e. address spaces/VMs, threads/virtual CPUs, scheduling/clocks, and inter-process communication) and leaves implementing all other typical operating-system services (such as resource or file management) to user-level programs.

One such user-level component is L4Re's Virtual Machine Monitor (VMM), which is used to emulate the virtual platform that is made available to (hardware-assisted) virtual machines. Components that do not need virtualization do not have to depend on the VMM which then yields a smaller TCB. In fact, each VM can have its own (custom) VMM, allowing to further reduce the trust relationship among different, mutually untrusting VMs.

For virtualization-friendly guest OS kernels such as Linux, the L4Re system provides a special, tiny VMM called *uvmm*. This VMM does little more than providing a boot API for guest OSes, providing virtual interrupts and CPUs, connecting the

VM to VirtIO-based virtual devices (such as a virtual network interface), and passing through physical devices the VM is allowed to access.

Apart from the VMM, the L4Re system provides components for memory and CPU management, for setting up VM and application resources such as physical memory and communications relationships, for bus virtualization and platform and device management, and for securely multiplexing a GUI. The loader component can be scripted with Lua language and allows static or dynamic device (pass-through), memory, and communication-relation assignments.

For more information on the L4Re system, please refer to our EW2016 paper [3].

IV. Virtualization of a Linux-Based Router OS

Our demonstration and evaluation vehicle for running a router OS in a virtual machine has been implemented on the NXP FRDM platform and uses two hardware-assisted VMs.

NXP's QorIQ FRDM-LS1021A board uses an LS1021A SoC that provides two ARM Cortex-A53 processors. These CPUs provide ARM's virtualization technology, which allows using hardware-assisted full virtualization on this board. To provide Wi-Fi routing capability, we have attached a Wi-Fi dongle to the board's USB interface.

Using uvmm, we run the following two VMs on top of the L4Re hypervisor:

Router OS—This VM runs a copy of OpenWrt with an unmodified Linux kernel. This VM drives the Wi-Fi device, which is passed through into this VM, and exposes its configuration interface over the Wi-Fi interface. As its Internet uplink, Router OS has a virtual network connection to Telco OS:

Telco OS—This VM runs a simple, FreeRTOS-based system with two main functions: It has pass-through access to one Ethernet interface that serves as the uplink port and passes all traffic on to the Router OS via the virtual-network interface—except that it accepts commands on a "telco" service it implements itself. This service is intended for use by the Internet provider (or telco operator) to trigger reboots of the Router OS from its boot image, which is invisible to, and unmodifiable by, the Router OS, and therefore always represents a clean state from which the first VM can restart. Reboots of the Router-OS VM do not require a platform reset or reboot. (In the future, this service may also update the Router OS's boot image.)

This architecture provides only a minimal attack surface for the Telco OS because it does not inspect data intended for the Router OS, and does not implement any application or configuration services.

This architecture has the property that any compromises of the Router OS, initiated either externally (from the Internet) or internally (by a rogue or cracked IoT device) can be undone from within Telco OS, without having to trust Router OS at all.

V. Live Demonstration

In our live demo session, we will run an exploit against a known bug in OpenWrt.

At first, we will demonstrate how an instance of regular OpenWrt running natively (without a hypervisor) will become unresponsive once the attack is performed.

Then, we'll run OpenWrt in a virtual machine as described in the proceeding section. We will show that attacks on OpenWrt are still possible, but can be mitigated by the telco by remotely rebooting OpenWrt from a clean state, and possibly even updating OpenWrt from Telco OS.

VI. Conclusion

The security benefits of virtualization are no longer confined to big iron datacenter applications. Virtualization can effectively be implemented in resource-constrained embedded systems such as home routers. It allows to separate complex operating system software, such as the Linux-based OpenWrt, from the trusted computing base in critical applications.

In preparation for the hand-on workshop: please download the software from https://l4re.org/download.html. The authors will provide additional download links and instructions for the demo during the class.

References

[1] The L4Re System, https://l4re.org/

[2] A. Lackorzynski and A. Warg. Taming subsystems: Capabilities as Universal Resource Access Control in L4. In Proceedings of the Second Workshop on Isolation and Integration in Embedded Systems, Eurosys affiliated workshop, IIES '09, pages 25–30. ACM, 3 2009. ISBN 978-1-60558-464-5.

[3] M. Röder, M. Hohmuth and A. Lackorzynski. Tux Airborne: Encapsulating Linux — real-time, safety and security with a trusted microhypervisor. In Proceedings of the Embedded World Conference 2016

[4] Security Guidance for critical areas of embedded computing – prpl Foundation, January 2016 https://prpl.works/security-guidance/