TowerOS: An Operating System for Network-Boundary Converged Multi-Level Secure Computing

Adam Krellenstein
adam@krellenstein.com

I. Background

A converged multi-level secure (MLS) computing system is one that allows the user to operate across distinct security domains through a single user interface (UI). Traditional MLS systems rely on hardware-level isolation using a keyboard-video-mouse (KVM) switch and no UI compositing[1] or more recently with software-level isolation and software-based UI compositing (e.g. using a hypervisor).[2] While hardware-level isolation is theoretically much more secure than software-level isolation, the overall usability of any MLS system without user-interface compositing is necessarily poor in comparison, because there is no single, unified interface provided for the user.

"Extant software solutions do combine the user interfaces for multiple domains onto the same desktop, however these rely on large trusted computing bases comprising hypervisor, security domain software, and drivers—making them too complex to evaluate and too risky to accredit for high assurance use. Software solutions fail to address the increasing risk of compromised hardware, implicitly incorporating many hardware components into the trusted computing base."[3]

Recently, a system for hardware-level isolation with hardware-based UI compositing was developed;[3] but the usability of even this design is still much lower than that of those with software-based compositing because all interfaces between the security domains must be implemented *in silico*. Raytheon's Forcepoint Trusted Thin Client and Remote allows users to access multiple isolated networks from a single thin client, but has no capability for user-interface compositing.[4]

II. Architecture

TowerOS implements a new, hybrid design which performs *software-based user-interface compositing* with *hardware-level isolation* using standard network interfaces. TowerOS relegates each security domain to an independent headless computer, each with its own application state and security policies. These **Hosts** are networked together over a LAN and accessible by the user through a **Thin Client** device that is connected to the same network. The applications running on the various hosts are composited within a single user interface running on the thin client using a combination of multi-function network protocols (such as SSH) and desktop-sharing software (such as VNC over SSL).

Instead of having to trust an operating system to be able properly to isolate different security domains all running on shared hardware, our design relies on cryptographically secure networking protocols to connect multiple independent computers together to form a single, virtual device that from the user's perspective functions very much like a normal desktop computer. Instead of running multiple virtual machines on a single computer (whether to save costs or to isolate different security domains at the level of a hypervisor) we instead merge together multiple computers into a single virtual machine, where the actual hardware that any given application runs on (for security, or, for that matter, for performance) is abstracted away. This provides for the best of both words: the security guarantees of hardware isolation plus the usability and flexibility of interfaces implemented in software.

Such a system may be built exclusively with commercial off-the-shelf (COTS) hardware, and its trusted computing base (TCB) of the system is limited to the codebase for the networking protocols (SSH, etc.), which may be both widely used and easily audited. For example, each host would run whichever user applications are allowed within the security domain associated with the device in question. So one host might be running an e-mail client, another a word processor, another a password manager, and another a web browser. One host might be left stateless and reserved for hotloading with fresh copies of an operating system. The user would be able to access each of these applications from the laptop thin client using SSH and VNC, with application windows composited into a single graphical user interface (GUI) using the desktop compositor. Clipboard management could be performed on the laptop using the thin client, and file transfers could be handled easily with `scp` or with a local file browser and `sshfs`.

III. Threat Model

The security properties of this design compare very favorably to those of software-boundary multi-level secure systems. First and foremost, such solutions rely on a large trusted computing base, including not only the (very complex) hypervisor, but also much of the underlying hardware (also very complex!) The network boundary is an ideal security boundary because it was historically designed explicitly for the interconnection of independent devices, often with different security policies. Both the hardware interface and the software compositing layer are small and well understood.

The only data being *pushed* to the thin client are pixels, clipboard data and audio streams from the hosts (and data are never communicated directly from host to host.) As a consequence, so long as the user of the thin client doesn't explicitly *pull* malware onto the device, say with SSH, the risk of compromising the thin client (and by extension, the other hosts) is practically-speaking limited to the risk of critical input validation errors in the screen-sharing software itself or at the level of the network drivers. That is, even if the UI compositor on the thin-client machine does not enforce any security boundaries between application windows, the primary attack surface is limited to the only application running *in* those windows, e.g. VNC.

IV. Comparison with Qubes OS

The state-of-the-art in secure computing systems[5] is Qubes OS is an open-source converged multi-level secure operating system that uses hardware virtualization (with Xen) to isolate security domains. As the former lead developer of GrapheneOS put it:

> *You can think of QubesOS as a way of approximating having 20 laptops with their own purposes, but all on 1 laptop. The security of each compartment still matters, and beyond isolating some drivers it doesn't do much to address that, but it does successfully approximate air gapped machines to a large extent. It's still significantly more secure to have separate machines but it's very impractical / unrealistic especially at that scale. There is no better option for approximating the security of using separate computers for different sets of tasks / identities.[6]*
>
> — *D. Micay*

With TowerOS, we hope to address this deficiency in the software ecosystem. In the following, we compare TowerOS with QubesOS, and we note the advantages and disadvantages of the different architectures.

A. Advantages

1. Most importantly, Qubes OS relies heavily on the security guarantees of Xen, which is large, complicated, and has a history of serious security vulnerabilities.[7]

   > *"In recent years, as more and more top notch researchers have begun scrutinizing Xen, a number of security bugs have been discovered. While many of them did not affect the security of Qubes OS, there were still too many that did."[8]*
   >
   > — *J. Rutkowska*

2. Qubes OS relies on the security properties of the hardware it runs on.

   > *"Other problems arise from the underlying architecture of the x86 platform, where various inter-VM side- and covert-channels are made possible thanks to the aggressively optimized multicore CPU architecture, most spectacularly demonstrated by the recently published Meltdown and Spectre attacks. Fundamental problems in other areas of the underlying hardware have also been discovered, such as the Row Hammer Attack."[8]*
   >
   > — *J. Rutkowska*

3. The complexity inherent in the design of Qubes OS makes the operating system difficult both to maintain and to use. Accordingly, Qubes OS development has slowed significantly in recent years: as of December 2022, the last release (v4.1.x, in February 2022) came almost four years after the previous one (v4.0.x in March 2018).[9]
4. Qubes OS has support only for extremely few hardware configurations. As of December 2022, are only three laptops that are known to be fully compatible with Qubes OS.[10] With only moderate effort, TowerOS may be hybridized with any modern operating system so long as that operating system supports the standard network interfaces required for SSH, etc. This flexibility can enable the system to run a wide variety of software and hardware.

B. Disadvantages

1. The primary disadvantage of the proposed design is the additional physical bulk of the computer in

comparison to a single laptop running a software-boundary solution such as Qubes OS.

2. With Qubes OS, each security domain has no hardware footprint, so it is theoretically easier to support a greater number of security domains.
3. In some cases, the security domain isolation within Qubes OS may be able to be more granular. For example, Qubes OS supports isolating the USB-protocol processing and the handling of the block device, when loading data from a USB key; however this would not be very practical with TowerOS.

V. Conclusion

Using "physically separate qubes" was proposed in the Qubes OS blog post cited above (in a hybrid design similar to what is being described here);[8] but the suggested architecture would leave hardware-boundary isolation as a second-class citizen and, by continuing to rely on a derivative of today's Qubes OS, preserve all of the hardware-support, maintainability and usability issues that the OS suffers from today. An operating system desired specifically for pure network-boundary converged multi-level secure computing, as described in this document, is simultaneously much simpler, more secure and more user-friendly than Qubes OS. Indeed, this design addresses all of the major problems with QubesOS completely.

Bibliography

[1] A. Soffer and O. Vaisband, "Secure KVM device ensuring isolation of host computers", Jul. 01, 2014

[2] A. Issa, T. Murray, and G. Ernst, "In search of perfect users: towards understanding the usability of converged multi-level secure user interfaces.", 2018.

[3] M. Beaumont, J. McCarthy, and T. Murray, "The cross domain desktop compositor: Using hardware-based video compositing for a multi-level secure user interface.", 2016.

[4] Raytheon Company, "Raytheon Trusted Thin Client". [Online]. Available: https://www.raytheon.com/capabilities/rtnwcm/groups/gallery/documents/digitalasset/rtn_216411.pdf

[5] E. Snowden, [Online]. Available: https://twitter.com/snowden/status/781493632293605376

[6] D. Micay, "OS Security: iOS vs GrapheneOS vs stock Android". [Online]. Available: https://www.reddit.com/r/GrapheneOS/comments/bddq5u/comment/ekze9n6/

[7] T. d. Raadt, "Re: About Xen: maybe a reiterative question but ..". [Online]. Available: https://marc.info/?l=openbsd-misc&m=119318909016582

[8] J. Rutkowska, "Qubes Air: Generalizing the Qubes Architecture". [Online]. Available: https://www.qubes-os.org/news/2018/01/22/qubes-air

[9] Qubes OS, "Download Qubes OS". [Online]. Available: https://www.qubes-os.org/downloads

[10] Qubes OS, "Certified Hardware". [Online]. Available: https://www.qubes-os.org/doc/certified-hardware