Qubes-OS

Arian van Putten

October 2015

Abstract

Qubes-OS is a secure operating system that uses Intel Virtualization technologies to provide isolation between different domains of work within the OS. By isolating parts of the system, like network access, device access, etc, into different domains, we can make sure attempts of attack on certain surfaces of the system won't allow compromise of other parts of the system.

In this paper we'll discuss Qubes-OS, a security through compartmentalization operating system that utilizes the Xen Hypervisor and latest Intel Technologies like VT-d to isolate different parts of the system in virtual machines to improve security. By isolating things like networking, the browser and usb storage, many examples of attacks can be mitigated.

Though this way of protecting a system seems effective, recent research has found certain weaknesses in the system.

1 Introduction

Qubes-OS is the child of Joanna Rutowska, a renown security researcher specializing in low-level and hardware level exploitation of systems. It's an operating system that utilizes security-through-compartmentalization to try isolate any attacks on any of the subsystems. Also it's free software. So anyone is able to use it, tinker with it and improve it.

2 Hypervisors

The need for virtualization of machines has always been there. It allows developers to simulate production environments on developer machines and allows multiple people to work on the same server without interfering with each-other (shared hosting). This virtualization was done by emulating hardware in software. Writing a program that literally reads a file instruction by instruction and 'executes' that instruction. All the subsystems of a machine had to be emulated within software. Of course, such a virtual machine is significantly slower then a real machine, hence a solution was needed to this problem.

In 2006, both AMD and Intel introduced new hardware virtualization technologies (VT-x and AMD-V) that allowed multiple operating systems to simultaneously share x86 processor resources in a safe and efficient manner.

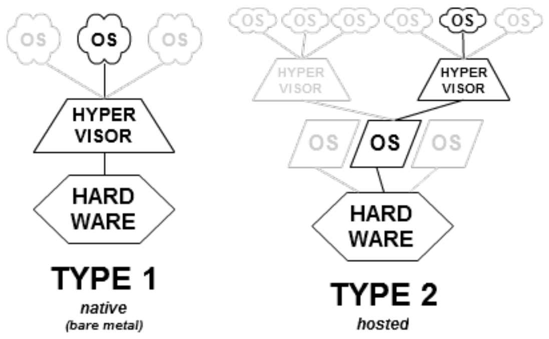

Figure 1: Two types of hypervisors

This opened the gate to modern hypervisors, where instead of emulating hardware to run multiple operating systems, we could actually run multiple operating systems side-by-side on the same system, without any intermediate steps. Examples of these new bare metal hypervisors are Microsoft Hyper-V, Citrix XenServer, and the open source counterpart of XenServer called Xen. In this paper we will only discuss Xen in particular, as this is the hypervisor of choice in Qubes-OS.

In next versions of Qubes-OS, A Hypervisor abstraction layer (HAL) is going to be added to the system, allowing any hypervisor that conforms to the HAL specifications to host a Qubes-based system but we will not discuss HAL in this paper any further. In theory, this will make Qubes-OS hypervisor-agnostic, allowing us to run Qubes on Hyper-V or even as a standalone application on virtual box.

2.1 Intel Hardware Virtualization

Using Intel's VT-x technology, we can run a type-1 hypervisor on the CPU. VT-x allows multiple 'virtual' machines to run on one CPU concurrently. Instead of emulating machines in software, we can actually run multiple machines on the same dedicated hardware. It is important for Qubes-OS to utilize Vt-x because in Qubes-OS it's very common to have more than 10 virtual machines. This would get sluggish really quickly in a traditional Type-2 hypervisor setting because of all the overhead a virtual machine introduces.

Qubes-OS also makes heavy use of Intel's virtualization technology. VT-d allows setting up Direct Memory Access (DMA) remappping to isolate what parts of the running system can access which devices. This way we can make sure a virtual machines that runs within a hypervisor (VMM) can only access certain parts of memory (and thus devices) or can't physically access any devices at all. All these mappings happen within the Intel hardware and thus allow for a robust hardware-level separation of VMs and underlying hardware.

Of course one must trust Intel on having properly built these isolation systems. Also, one has to trust the hypervisor. The hypervisor is the one and only part of software that runs before anything else and can basically do anything it wants within the CPU. This makes hypervisors a great rootkit! If you are able to install an evil hypervisor on a computer, you're in for a treat.They run before your Windows or Linux install even boots up, and they can access every part of the system without your OS (windows, linux) even being able to detect that it runs (as they run in a lower-privileged mode on the CPU). If you're interested in how these rootkits work and how Intel detects and tries to remove Virtualization rootkits, there was a great talk in 2008 by Intel during the Blackhat conference [1].

2.2 The Xen Hypervisor

You want to make the attack surface of a hypervisor as small as possible, because once the hypervisor is compromised, the entire system is compromised. Ideally, you'd want the hypervisor to be so small, that you can formally prove that the system functions correctly and without bugs. Though a lot of research is being put into this subject, and that research even resulted in a formally verified microkernel[2], hypervisors are still a bit too complex to fully verify. So instead of proving them correct, we want to keep them as small as possible so that it's easy for people to audit and maintain the code.

Because no practical formally-proven hypervisor exists today, the Qubes-OS project has chosen to base their system on Xen. Xen is a security-oriented hypervisor originally developed by Citrix, that is used in production by many large hosting companies and has a high reputation regarding taking security holes seriously and fixing them quickly.

Xen uses a small abstraction layer of an idealized x86 instruction set to support as many CPUs as possible. This way, many operating systems can run on Xen with little modification. [3].

3 Qubes-OS

Qubes-OS is a VM-based solution for using personal computers with maximum security. It provides security by compartmentalization by separating work domains in sperarate VMs so that they can not interact with eachother directly or with the underlying hardware, truly isolating different workspaces from eachother and the underlying system. This means that when one part of your system gets compromised, say, your web browser, an attack will not be able to reach other parts of the system or hardware. The whole idea of Qubes-OS is making the attack surface of compartments as small as possible by exposing as little interfaces as possible for each compartment.

Running such a system on a traditional hypervisor type-2 infeasible because so many virtual machines are needed that a computer would need very many resources to be able to emulate all these machines at once. This is why Qubes-OS leverages the Xen Hypervisor to directly run VMs on the underlying hardware.

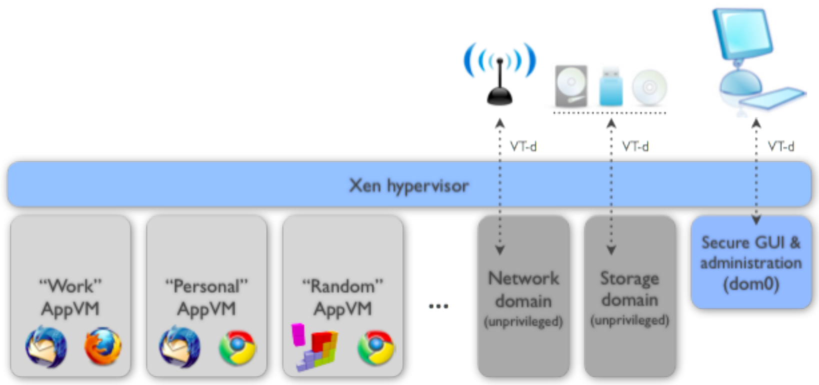
VT-d
VT-d
VT-d
Xen hypervisor
"Work" AppVM
"Personal" AppVM
"Random" AppVM
...
Network domain (unprivileged)
Storage domain (unprivileged)
Secure GUI & administration (dom0)

Figure 2: Qubes-OS

Qube-OS has multiple types of VMs, each with a specific task in mind, that are privileged only to be able to do that specific task. In the next sections, we will discuss each type of VM and the role it has in the system.

For a more in-depth decription of how these systems work, please check out Rutowska's paper [4].

3.1 Dom0

Dom0 is the administrative domain of Qubes-OS. It has almost as many privileges as the hypervisor itself. The amount of code that interfaces with the outside is therefore been kept to a minimum to avoid compromise of this domain at all cost.

Dom0 hosts the XenStore daemon, which is the system-wide registry used for system management. It's a way for Dom0 to share configuration with all the other subdomains.

The Dom0 is also used to host the graphical subsystem of Qubes-OS. The reason for not hosting the graphical subsystem in a separate domain has a rather practical reason. If one has compromised the graphical subsystem, one can eavesdrop all the things someone does with all the domains anyway; it's like someone literally sitting next to you and watching you work over your shoulder. It therefore has no added security benefit to separate it from Dom0.

These are the only two interfaces to interact with Dom0. All other forms of interaction are disabled. Dom0 has no access to underlying hardware or other VMs except through the XenStore and graphical protocols.

3.2 NetVM

NetVM is a service domain oriented to interfacing with the Internet. Through VT-d, it has access to the network interface and only the network interface. Through XenStore (te database in Dom0), other VMs can set up a connection with NetVM, allowing them access to the Internet. Dom0 will then set up a virtualized network interface in the VM, that will be piped to the NetVM. It is important to note that non of the domains interfacing with NetVM have direct access to the physical network interface; only NetVM itself has access to this. This means that exploits in kernel drivers or the hardware itself will only affect the NetVM and will protect other domains from such attacks.

3.3 AppVM

The Internet is a dangerous place. And using a web browser that is executing random javascript code from random ad websites every moment you browse makes you an easy target for hackers. It only takes one zero-day exploit to compromise your system.

Chromium, the project behind the very popular Google Chrome browser, has taken a sandboxing approach to help make users more safe on the web [5]. There's a single supervisor-kernel (similar to our Dom0),and for each tab, a seperate unprivileged rendering and execution engine in spun up. Now if one of these tabs gets compromised, the rest of the browser will remain intact and hopefully the rest of the system too.

Qubes-OS takes this idea of sandboxing from Chromium, and brings it to any application. For each application domain (in which multiple applications can run), a virtual machine is started up in unprivileged mode, meaning it can not access any of the devices hardware. In this virtual machine the application runs in true isolation from the rest of the system.

Through XenStore, the virtual machine can negotiate with Dom0 to be connected to (one of the) NetVMs for example, to give the virtual machine access to the Internet. Or it can negotiate file-transfers between AppVms or clipboard transfers between AppVMs. Or even negotiate filesystem access on a usb storage device.

3.4 Practical usecases

In the next sections, we will describe two scenarios where Qubes-OS shines. These two scenarios were originally described by Rutowska in [6]

3.4.1 TORVM

In this section we'll discuss a possible workflow used in Qubes-OS. To allow for anonymous browsing on the web, TOR is a popular tool. But because laptops have a pretty big attack surface, a lot of out of band information leakage can happen while using TOR, exposing the identity of the user. For example, by default, if you use Chromium over a TOR proxy, it will actually not send DNS traffic over this proxy but over the standard network interface, exposing to the DNS server what websites the TOR user visits. Because DNS traffic isn't encrypted, it is then easy for an attacker to find out at what time a user visited what website. And then if the attacker has access to that website's logs, he can cross-check the times that he saw the DNS requests and the times certain websites were accessed to try de-anonymize the user.

And of course, the web browser is one of the biggest sources of vulnerabilities even today, and because many users of TOR would like to browse the web, this will endanger them rather easily. If the browsers is compromised, it's not hard to also compromise the TOR daemon itself and that way de-anonymize the user as they run on the same machine.

Hardcore TOR users will use a separate computer that is used as an intermediate 'proxy-broker' between the laptop and the TOR network to minimize the damage to the user's anonymity if his laptop would be compromised through a browser exploit.

With Qubes-OS, we can set up a similar environment, but instead of using separate computers, we can set up separate domains within Qubes-OS and get a similar level of protection. We would run the browser in a separate browsing domain and only allow it to connect through the Internet by communicating with a TOR domain which sets up a proxy through the TOR network. The TOR domain itself then again communicates with the NetVM domain to actually connect to the Internet.

Now if the browser domain gets compromised, there is no way for an attacker to reach the TOR daemon and affect it. We've been able to isolate the attack and keep the user anonymous.

Also, because there is only one network interface, the one being piped through TORVM, there is no possible way that a DNS leakage would occur. There is only one network interface so there is no way that a DNS query can go over a different interface than the rest of the traffic.

This is a good example of how reducing the amount of interfaces reduces the risk of the user. You know the VM has access to only one interface, and you know how that interface acts.

Of course this approach can also be used to force all traffic to go over your company's VPN, making sure no secret company information leaks to the outside world.

Figure 3 illustrates this approach this schema.

3.4.2 Storing PGP keys in a separate VM

PGP is one of the most used systems for mail encryption. One important part with PGP is that you keep your private keys secure, but to be able to sign and encrypt mails, you need access to the private key. Which leads to an interesting problem.

One solution is to use an OpenPGP smartcard. The private keys remain secure on the smartcard, and instead of letting encryption and signing happen on your computer, you send the message to the smartcard and you get a signature or encrypted version back. This way your private keys are impossible to leak.

With Qubes-OS, we can simulate a smart card by having a separate PGPVM which stores the private keys. an AppVM can then send a message to the PGPVM and get an encrypted or signed vesion back. Similrarly to the OpenPGP

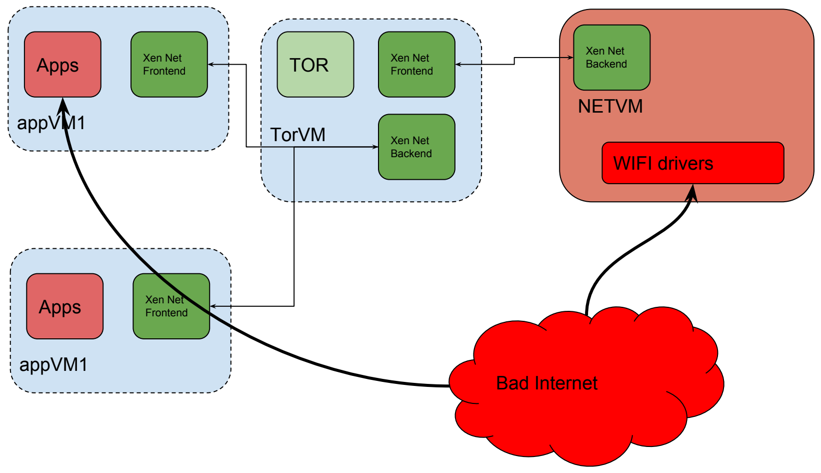

Figure 3: Red parts are parts that are highly vulnerable to attacks. As you can see, a compromise of the NetVM will only affect the NetVM and a compromise of an AppVM (Say through a browser exploit) will only affect one AppVM and not your NetVM or other AppVMs.

smartcard, this uses the standard pgp-agent protocol, allowing mail clients that support PGP to easily integrate with this system.

If there is a compromise of the mail client through an exploit, the worst a hacker can do is abuse the pgp-agent to get messages signed and encrypted. This is bad, but not as bad as the actual private key leaking. As soon as the user discovers the AppVM has been compromised, he can simply shut it down to stop the attack and the keys will remain save. It's not ideal, but it's better than the original scenario: The attacker compromises the mail client, and because the agent is running on the same machine, compromises the private keys as well.

3.5 The boot process and Evil Maid Attacks

Of course all this compartmentalization is great. But what if you're at the Blackhat Conference in Vegas, but a maid at the hotel tries to tamper with your system while you're away. Qubes-OS has some protection mechanisms to help you, which we shortly discuss here.

First of all, using Linux Unified Key Setup (LUKS) [7], the hard drive is fully encrypted. The encryption mode that is used is AES-CBC-ESSIV:SHA256. A sort of CBC and CTR hybrid. It encrypts each disk sector with CBC, using $IV = E_h(DiskSectorNumber)$ where $h = H(key)$ as the initialization vector for each encrypted sector. So if the device is powered off, a person won't be able to look at your data.

The Dom0 and hypervisor code are stored unencrypted however (A machine

machine can't boot an encrypted disk without having some parts of it being unencrypted). To make sure Dom0 hasn't been tampered with, inside a Trusted Platform Module (TPM), hashes of the boot image are stored. Only if the TPM hashes and the hashes of the boot image are the same, will the TPM release a secret that can be used in conjunction with the disk encryption key to decrypt the rest of the system. This means that if the system has been tampered with, the system will simply refuse to boot.

How exactly a TPM chip works is beyond the scope of this article. But a simplified view of it is: One can set certain values within a TPM (like hashes), and those values can only then be removed from the TPM if you know the password. The values on the TPM can then be used to verify aspects of your system during boot. If the verification succeeds, the TPM is able to provide a secret key that can be used.

4 Discussion

Though not as safe as 'airgapping' physical computers, Qubes-OS offers an interesting way to isolate security threats. Of course one must ask how safe the underlying technologies are. Does Vt-d actually make sure I can't access other devices? Can VT-x prevent VMs from reading memory of other VMs?

Because Intel's technology is propitiatory, the answer is: We're not sure. There is no way for us to verify if Intel's system is sound. this means bugs do get discovered every once in a while that can lead to leakage of information.

The biggest problem is though, you cannot fix a processor with a software update. If the hardware is broken, the only way to fix it is to replace it with a new processor.

Recently a new exploit has been presented at Blackhat called "The Memory Sinkhole".[8] It is an exploit that affects the Intel System Management Mode. The most operation-critical part of the entire processor. The SSM manages the Trusted Platform Module, the Direct Memory Access controller, power management and all virtualization security modules. Basically every security feature that we utilize to make Qubes-OS secure. Pwning it is the holy grail of hacking. This means that all chips made by Intel in the last 22 years are not secure anymore and can't be fixed either except by replacing the hardware. This, of course, is terrible news.

The attack is hard to pull off though. It's a privilege escalation that can only happen when you're already full privileged. For example it could happen when there's an exploit in Xen (which runs as the most privileged piece of code) that allows arbitrary code execution. For example a buffer-overflow exploit that would allow overriding memory of the Xen Kernel would allow this exploit to work. This stresses again how important it is for the Hypervisor to be as secure as possible.

But it does make you think; Is proprietary hardware really what we want? How do we monitor that what Intel develops is actually secure? Can we blindly trust them?

Personally I think it's a bad idea to have so much blind faith in hardware vendors. I think we should work towards open hardware solutions to guarantee security in the future. I think we're morally obliged to do so, just like we're morally obliged as developers to work on open source software to make the world more secure.

If you are interested in open hardware. Check out `http://openrisc.io/`

References

[1] David Samyde Yuriy Bulygin. Chipset based detection and removal of virtualization malware a.k.a. DeepWatch.

[2] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM.

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, October 2003.

[4] Joanna Rutowska. Qubes OS Architecture Version 3.0, January 2012.

[5] Adam Barth, Collin Jackson, Charles Reis, and The Google Chrome Team. The Security Architecture of the Chromium Browser. Technical report, 2008.

[6] Joanna Rutowska. Software compartmentalization vs. physical separation (or why qubes os is morre than just a random collection of vms), August 2014.

[7] Clemens Fruhwirth. LUKS On-Disk Format Specification - Version 1.1.1, December 2008.

[8] domas//blackhat 2015. The memory sinkhole: An architectural privilege escalation vulnerability.