Software compartmentalization vs. physical separation
(Or why Qubes OS is more than just a random collection of VMs)

Joanna Rutkowska
Invisible Things Lab
August 2014

Many people believe the Holy Grail of secure isolation is to use two or more physically separate machines. This belief seems so natural, that we often don't give it much thought. After all, what better isolation could we possible get than physical "airgap"?

I would like to discuss two exemplary scenarios involving isolation: one for securing the Tor process and another for securing email operations, and compare the pros and cons of using the physical isolation vs. the software compartmentalization as currently possible on Qubes OS[1].

Securing Tor: the physical separation approach

I'm sure most readers realize the problem of hosting the Tor process in the same operating system as the apps which use it, specifically a web browser which has a very large attack surface: in case of a successful attack against any of the apps, the Tor process also gets compromised easily, and this is pretty much the end of the game.

Some people[2] attempt to remedy this problem by using two physically separate computers – one to run Tor proxy and the other to run client apps. A sample configuration is depicted on the figure below (Figure 1), together with the most obvious attack vectors.

The coloring[3], which represents the level of difficulty to attack certain entity, and so also its probability of being compromised when considering attacks originating from it, has been chosen qualitatively, based on my experience and intuition, as I'm not aware of any meaningful quantitative methods to be used here. It makes most sense to consider them as relative measure of difficulty of attacks among the presented items, rather an absolute measure.

The picture shows potential attacks on WiFi/eth drivers and stacks, including things such as DHCP client, coming both from the "external bad Internet", which might be your local WiFi or ADSL router, as well as from the client laptop, which might got compromised because of the Web browser or other client app attack. We can draw, a rather obvious, conclusion, that the security of the Tor process hosted on such physically separate gateway is no better than the security of all the networking stacks, including drivers and programs such as DHCP, exposed by the gateway.

Normally one would assume the security of the Tor gateway to be still much better than the security of the client laptop, because of the large attack surface against a Web browser. However, we should

[1] https://wiki.qubes-os.org/

[2] See e.g. https://github.com/grugq/portal

[3] Unfortunately this paper is not fully compatible with B&W printers…

realize that in some scenarios it might be the other way around, as in case of e.g. the user “being very careful”[4] when browsing from the client laptop. In that case the security of the laptop, which presumably now represents the most sensitive elements of the system (e.g. because it hosts important documents) suddenly is still no better than the security of Tor gateway, as the attack surface via the locally exposed networking code is surely no smaller.

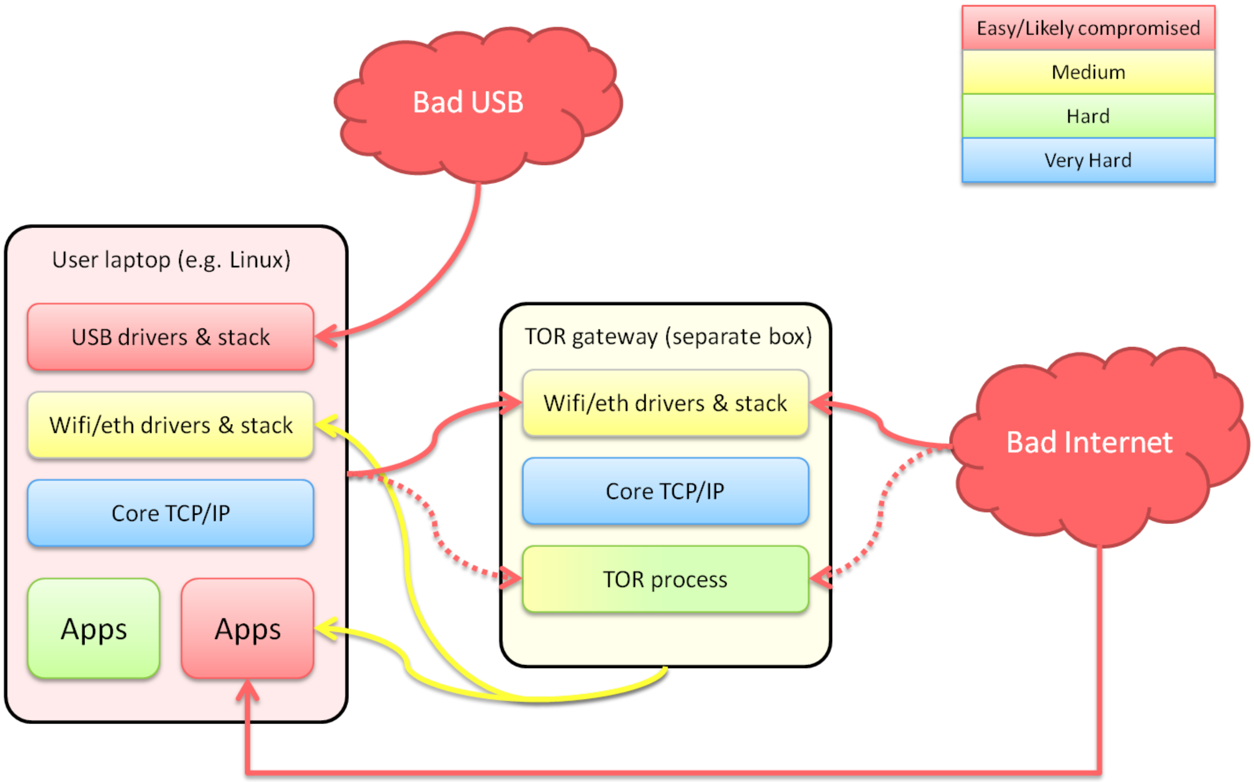


Figure 1 Securing Tor using separate machines. Colors illustrate attack surface size/difficulty (see the legend box). Arrows show potential attacks origins and their colors match that of the orignating entity color. The color of the whole machine is the color of the least secure item hosted on it.

Plus, as every laptop these days, it does expose a huge and way-to-simple-to-exploit attack surface associated with the USB device handling[5]. The potential attacks that might have compromised the laptop via USB devices (e.g. when the user decided to copy some documents to/from the laptop) might, of course, also try to attack the Tor gateway from the “other side”, as also illustrated on the diagram.

All in all, the overall security of this seemingly well secured system seems to be surprisingly weak, in spite of physical separation used.

[4] The phrase: “being careful when browsing”, is often pretty meaningless, but might be interpreted as e.g. using Firefox with all the scripting disabled, etc.

[5] For a survey of various USB attacks against desktop systems, see: http://theinvisiblethings.blogspot.com/2011/06/usb-security-challenges.html

Securing Tor: software compartmentalization approach (Qubes OS)

Let's now look at how the above problem – secure Tor usage – could be solved via fine-grained software compartmentalization aided by modern hardware isolation technologies, specifically virtualization and IOMMU/VT-d.

The picture below (Figure 2) shows how this could be done on Qubes OS R2, a flexible platform designed around the Security by Compartmentalization principle. Qubes OS R2 builds on top of Xen and uses lightweight Linux AppVMs and ServiceVMs[6].

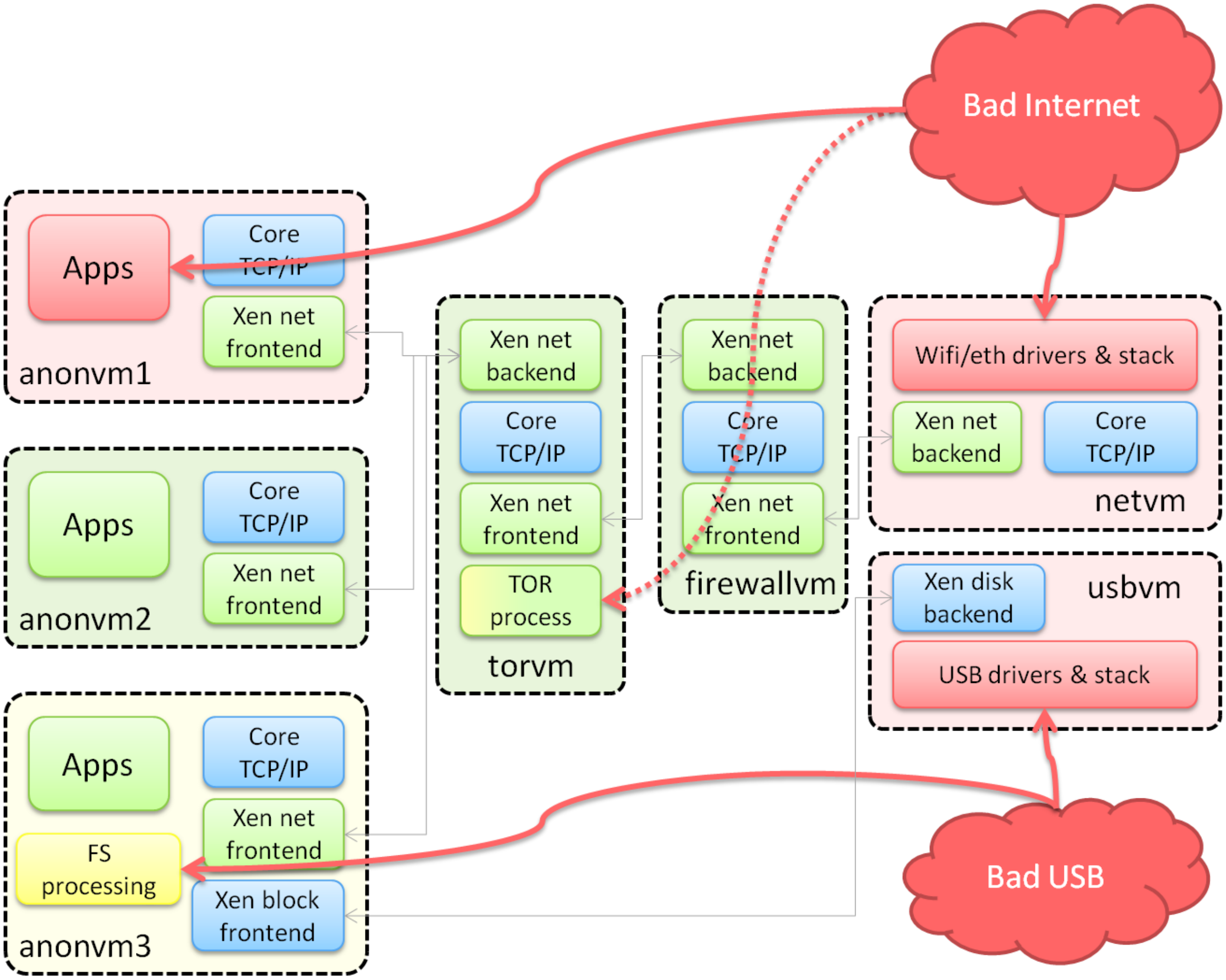


Figure 2 Securing Tor on Qubes OS. As in the previous picture, colors have been used to signalize relative ease of attacks on different components. The discreet blocks encricled in dashed lines represent, this time, Qubes VMs. The other elements of the system and their associated attack surfaces, such as the Xen hypervisor and select Dom0 services exposed to VMs, have not been shown, but it is assumed they are not worse than "blue" (their secrity is discussed later in the text).

[6] Qubes OS R2 also fully supports Windows 7-based AppVMs, although they are not so lightweight as the Linux AppVMs. In fact this very article has been prepared in MS Office running on Windows 7 AppVM on Qubes OS.

Qubes OS allows to isolate both the networking and USB stacks in unprivileged service VMs, a functionality made possible by the Xen hypervisor and IOMMU technology (called VT-d on Intel hardware), available on many laptops today.

But mere creation of a networking VM (called “NetVM” for short in Qubes parlance[7]) or USB controllers-hosting VM (called “UsbVM” in Qubes) hardly solves any problem *automatically*. This is because we still would like to *use* networking, and, often, also USB devices…

The real security benefit comes from the *reduction of interfaces* that are possible in this software compartmentalization. Consider again the attacks coming from the local environment targeting WiFi drivers and stacks and DHCP client. What the attacker can gain from those attacks is to compromise the NetVM at best[8]. It is tempting to think that the same attacks are now also possible against other VMs which connect to the NetVM. This, however, is not correct, because the amount of networking code exposed by other VMs is significantly smaller, compared to that which the NetVM exposes to the outside world. We don’t need complex WiFi drivers and stacks in the VM, neither we need DHCP client there. This is because we operate now in a much simplified scenario, using Xen shared memory to exchange data instead of physical wires or radio waves, we control all the aspects of the system, so we don’t need protocols such as DHCP to configure “networking”. In fact we don’t need networking at all, yet, for backward compatibility, we need to use some Ethernet-like dummy drivers, which are represented by Xen front- and backend drivers in the picture above, as well as the *core* TCP/IP stack.

We can easily imagine eliminating virtually all the networking code from most of the AppVMs, and replacing it with simple inter-VM *pipes* (such as those provided by qrexec in Qubes OS). That would require some modifications to the applications (such as Split GPG discussed later), or use of application-level proxies, though.

The benefit of reduced networking interfaces also works to protect the user apps against potential attacks coming from the Tor VM, an important consideration in some scenarios as discussed in the previous section. Furthermore Qubes agility in creation of multiple AppVMs makes it significantly easier to keep some of the AppVMs less exposed to attacks, something that is a topic of one of the next sections.

[7] Qubes OS allows to create multiple NetVMs and ProxyVMs, see: http://theinvisiblethings.blogspot.com/2011/09/playing-with-qubes-networking-for-fun.html

[8] Some people find it disconcerting that the attacker, who compromised the NetVM, might use it to perform all sorts of networking-oriented attacks, such as sniffing or various MITM attacks. That’s true, but remember all the same attacks are possible to be conducted from the local WiFi network, and should be addressed on the protocol level (use of encryption).

Handling of USB in Qubes OS

Similarly to compartmentalized networking, we can get various benefits also when we contain all the USB stacks in a UsbVM[9]. And this applies to the scenario of securing Tor just as well as to any other scenario on a desktop system (hence a distinct section).

The primary attack surface, which includes potential bugs in the USB stacks parsing code or USB-device-specific drivers, including various filesystem drivers or modules, can now be contained within a UsbVM[10]. Similarly as in the case of NetVM, the most benefit can be seen when we can expose the USB functionality from the UsbVM over much simplified interfaces, than the USB device exposes to the OS. In Qubes there are several ways of how to do that, but currently only for mass storage devices (aka USB disks).

It is possible to expose block devices (aka disk volumes) using the Xen block backend hosted in the UsbVM[11]. This way, all the USB-specific, complex processing happens in isolated UsbVM while the important outcome, the block device, is exposed to the user AppVM, where it could be mounted just like a normal disk[12].

This approach still exposes the receiving VM for some hazard though. This is because the block device, which looks like a disk, does trigger some amount of parsing in the Linux kernel (or whatever other OS kernel that is running there). Thus it is possible for the attacker (who e.g. compromised UsbVM, or just for the compromised USB device) to expose malformed partition table or LUKS-headers, hoping to exploit a hypothetical flaw in the kernel (in case disk encryption isn't used, this could also include malformed filesystem metadata and, of course, malicious files – even more significant attack surface). Additionally, similarly as in case of the networking, there is also some possibility for an exploitable bug in the block *front end*. To further protect against such attacks, it is possible to use Qubes security-optimized inter-VM file copy operation (see qvm-copy-to-vm command). This, combined with a file-hosted encryption containers, takes volume metadata processing code out of TCB.

Speaking of USB, we shall note Qubes also has special support for making system-wide backups with ability to store them on a USB disk (or some Network Attached Storage, or somewhere in the Cloud) without the need to trust any USB or volume parsing code, and without requiring to mount any USB disks in Dom0. This is a subject for another article, though.

[9] Qubes installer does not create UsbVM by default (as it does for NetVM), yet the user is able to create it with a few clicks in Qubes Manager. It is also recommended in Qubes documentation. In the next release we will likely add default UsbVM creation to the installer also.

[10] It is possible to have as many UsbVMs as there are USB controllers in the system.

[11] Qubes supports this via handy qvm-block command, as well as clickable UX in Qubes Manager. It is even possible to start an AppVM and make it boot directly from a block device hosted in another VM (e.g. downloaded ISO).

[12] It could be also "piped" through something like LUKS or TrueCrypt in order to keep the UsbVM from learning the content.

As of now there is limited support for dealing with classes of USB devices other than mass storage[13]. This means that if one would like to use e.g. USB-based camera with one of the AppVMs (e.g. for Skype), the whole USB *controller* (i.e. a distinct PCIe device) would have to be assigned to that VM. While this offers good isolation (USB bus is inherently insecure, so isolating on USB controller level is much better), the convenience and usefulness of this approach depends on how many physical USB controllers there are in a given laptop and how are the devices inter-connected to them. In future Qubes releases, better support for USB virtualization, should offer much more elasticity and convenience here.

It is important to point out that there is a class of USB devices which is so security sensitive, that it makes little sense to delegate its handling to a UsbVM. It is the keyboard and mouse. These devices convey the user *will* to the system, and so if they could be hijacked, then all is lost anyway[14]. Luckily majority of laptops today do *not* use USB-based keyboards or touchpads[15].

Securing email operations

Software compartmentalization can show benefits compared to physical isolation not only in de-privileging networking or USB stacks. Let's now consider another scenario – securing of email-centric work environment.

Primitive "airgap" or "security by virtualization" solutions often attempt to enforce a way too simplistic threat model on the user by enforcing military-style "top-secret/classified/unclassified" or simple "work/personal" dual persona models. But typical user workflows are in practice much more complex, often involving many more security domains than just two, and typically without any kind of simple relation of trust between them[16].

The next picture (Figure 3) shows a few Qubes AppVMs involved in securing of my daily work-related workflow. The work-email is the central AppVM for me, because today my work mostly revolves around communication with various people via email. Many of the emails I deal with are GPG-encrypted.

However, I don't keep my private keys in the work-email AppVM, because I still consider it relatively vulnerable as the Thunderbird attack surface seems quite large if we consider all the networking code used to interact with my *untrusted* email server[17], as well as all the email headers processing, not to mention parsing of the message contents (which, thankfully can often be opted out).

[13] Qubes already has infrastructure to support USB virtualization (see qvm-usb) yet the Xen PVUSB backend is still not very usable.

[14] In future releases of Qubes, especially commercial spin-offs, this might not be entirely true, because we plan on separating the user vs. admin roles by introduction of GUI vs. Admin domain separation.

[15] Apple laptops being a notable exception here.

[16] Which security domain should be considered more sensitive: work or personal? And how about banking?

[17] It is healthy to always consider external servers to be potentially compromised. This includes a possibility that any 3rd party server might be usually trivially compromised by the internal personnel, aka admins.

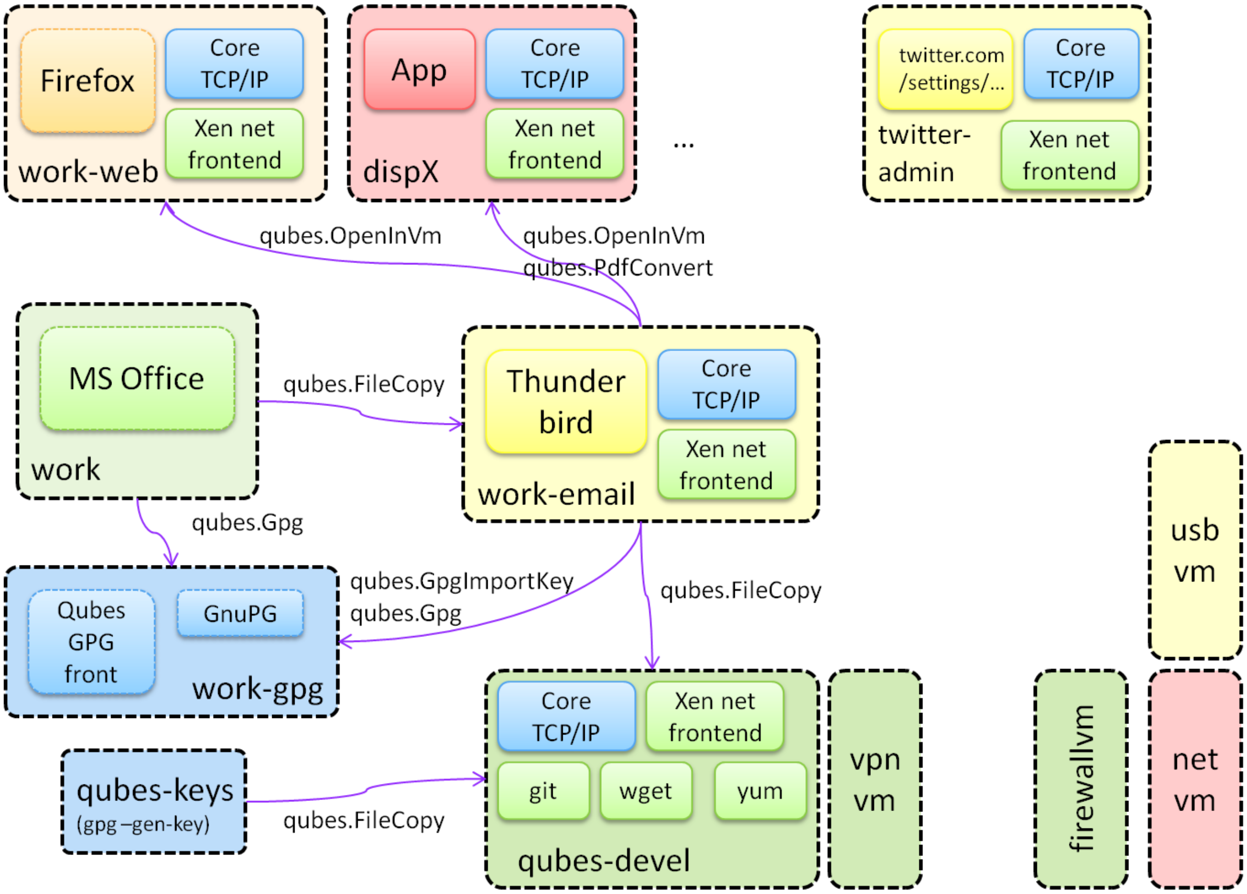


Figure 3 Securing email-centric work environment in Qubes OS. Arrows show allowed executions of Qubes RPC services (qrexec). They are controlled by a centralized policy in Dom0.

In order to protect my private keys, I use Qubes Split GPG application[18], which acts similarly to a smart card, yet offers some additional benefits. First, it doesn't require USB to be part of the TCB. Second, it offers more control over access to the keys, something that a smart card does not, as once it is plugged in, and the PIN got sniffed, the malware can *use* the private keys at will, without the user every noticing anything suspicious.

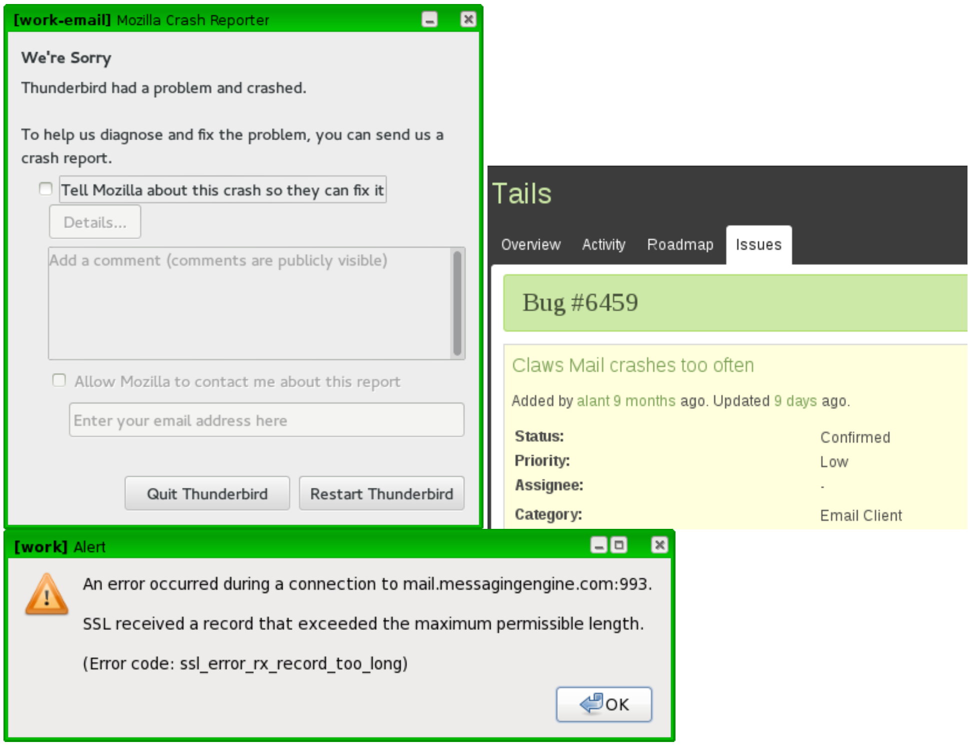
My work-email VM is also configured to automatically open URLs (from email messages I receive) in my work-web VM. This is trivially configurable in Qubes by telling TB to use 'qvm-open-in-vm work-web' command as a web browser. Additionally, Qubes default firewall takes care of ensuring that my work-email VM can only establish networking connection to my email server and nothing more[19].

[18] See: https://wiki.qubes-os.org/wiki/UserDoc/SplitGpg

[19] A word of caution about treating Qubes firewall as an anti-leak prevention mechanism. It has not been intended to act like so, and generally anti-leak prevention is mostly impractical. The firewall is intended mainly as a way to prevent user mistakes (e.g. accidentally opening Firefox in email AppVM), non-intentionally-malicious application misconfigurations (open firewall to allow listening on some port), or as a convenient place to reliably log traffic to/from select VMs.

I also have a dedicated AppVM, this one exceptionally based on Windows 7, which I use to work on various trusted documents (e.g. preparing slides, writing reports, etc). I can directly encrypt or decrypt from within this VM, again using Qubes Split GPG. This way I can keep my work-email domain from learning the content of these sensitive documents when I send them over email. I can also directly open a document, which I receive via email from not-so-trusted source, in a Disposable VM with just two clicks. This way I don't need to risk potential compromise of my email, yet I can also provide adequate protection to the document itself by opening it in an isolated AppVM instead of in the all-purpose yet not-so-trusted work-web VM, something especially important when dealing with NDA-protected documents. I can also use Qubes trusted PDF converter in other to "upgrade" a document and save it into my regular work AppVM[20].

Of course I can easily combine the advanced networking and USB handling as discussed earlier in this text with any or all of my AppVMs. E.g. I can easily attach some of the AppVMs through a Tor or VPN proxy service VM.



[work-email] Mozilla Crash Reporter

We're Sorry

Thunderbird had a problem and crashed.

To help us diagnose and fix the problem, you can send us a crash report.

□ Tell Mozilla about this crash so they can fix it

Add a comment (comments are publicly visible)

□ Allow Mozilla to contact me about this report

Enter your email address here

Tails

Bug #6459

Claws Mail crashes too often

Added by alant 9 months ago. Updated 9 days ago.

Status:	Confirmed
Priority:	Low
Assignee:	-
Category:	Email Client

[work] Alert

An error occurred during a connection to mail.messagingengine.com:993.

SSL received a record that exceeded the maximum permissible length.

(Error code: ssl_error_rx_record_too_long)

Figure 4 Email clients do crash... (courtesy of @bleidl for the Tails ticket, rest is from the author's home collection)

It should be obvious that this sort of fine-grained decomposition of user tasks is not possible, or extremely difficult and costly to implement using physically separate machines. What might not be so evident is that Qubes also allows for this fine-grained compartmentalization to be almost unnoticeable to the user. This is achieved thanks to Qubes seamless desktop integration, automatic memory balancing, and additional mechanisms that could be used to automate the mundane tasks of remembering what should be done in which AppVM, such as automatic URL opening in pre-defined

[20] See: http://theinvisiblethings.blogspot.com/2013/02/converting-untrusted-pdfs-into-trusted.html

AppVM or Disposable VM. Dealing with documents speared among different AppVM is, in fact, not significantly more inconvenient than dealing with multiple folders or disks on a traditional desktop system[21].

In order to further ease the management of many VMs, Qubes provides support for sharing root filesystems between AppVMs in a read-only manner[22]. Thanks to this, not only we can save huge amount of disk space, as there is no need to duplicate the root filesystem across all the VMs, but it is also possible to update software in all the VMs all at once. This is supported both for Linux-based AppVMs (Fedora, Debian, and Archlinux-based), as well as for Windows-based ones.

Some people even consider this as an extra security feature – namely if an AppVM gets restarted, its root filesystem is returned automatically back to the template's "golden image" state. However, one should be careful here, as the VM-compromising code might very well be triggered from within the user home directory, which, naturally, is preserved across VM reboots (except for Disposable VMs). One advantage of the non-persistent rootfs though, is that the malware is still inactive before the user's filesystem gets mounted and "processed" by system/applications, which might theoretically allow for some scanning programs (or a skilled user) to reliably scan for signs of infections of the AppVM. But, of course, the problem of finding malware hooks in general is hard, so this would work likely only for some special cases (e.g. an AppVM which doesn't use Firefox, as otherwise it would be hard to scan the Firefox profile directory reliably to find malware hooks there). Also note that the user filesystem's metadata might got maliciously modified by malware in order to exploit a hypothetical bug in the AppVM kernel whenever it mounts the malformed filesystem. However, these exploits will automatically stop working (and so the infection might be cleared automatically) after the hypothetical bug got patched and the update applied (via Template VM update, so reliably), which is still a much desired security feature.

Securing Twitter Account

Twitter accounts become increasingly lucrative target for criminals these days, with some studies presenting them as even more attractive for attackers than credit card numbers[23]. One would think that the best method to secure one's twitter account might be to have a dedicated AppVM[24] for this… But that's not quite the best strategy…

Most people seem to be using Twitter Web Client (TWC), i.e. twitter.com website, because it is convenient, simple and powerful. In fact it is too powerful. TWC tries to be both the daily user tweet reader, as well as the ultimate administrator tool for one's tweeter account. Unlike for 3rd party apps, which can be easily revoked access from within TWC interface (see Figure 5), the compromise of the

[21] Admittedly one important functionality that is lost is system-wide document search. It is, however, thinkable to write Qubes app offering such service spanning multiple domains. It would require some thought, though to do it right. Sooner or later we will likely introduce it in Qubes R3, probably together with a unified File Manager for all the user files.

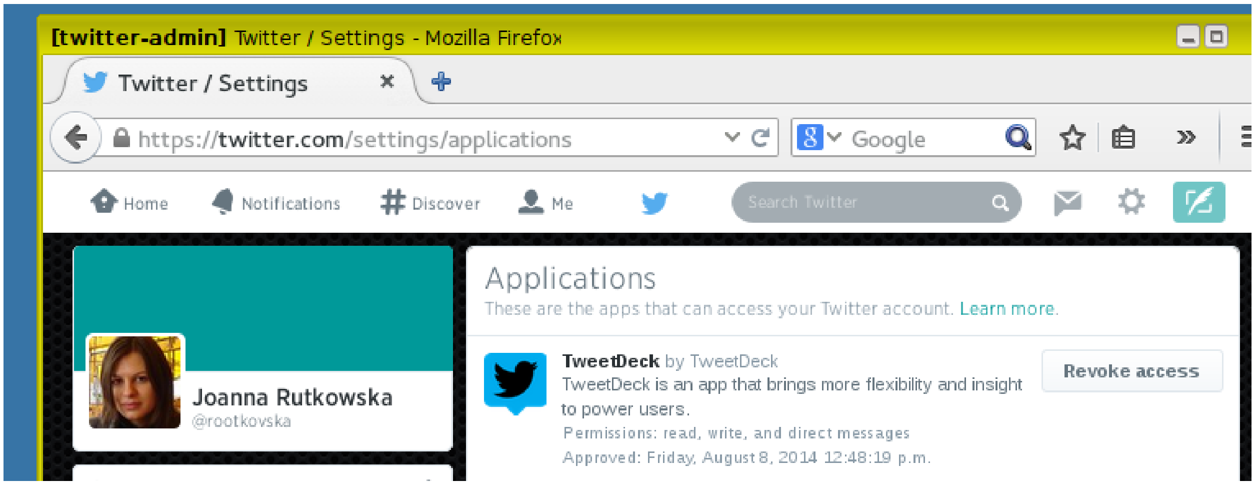
[22] See: https://wiki.qubes-os.org/wiki/TemplateImplementation

[23] See e.g.: http://www.businessinsider.com/why-hackers-want-your-twitter-account-2014-8?utm_source=rand_social&utm_medium=hootsuite_rand&utm_campaign=hootsuite_rand_social

[24] As follows from the earlier discussions, a Qubes AppVM should be safer than a dedicated physical laptop for twitter, because of all the networking and USB stacks that could be easily sandboxed on Qubes OS.

Web browser (where the TWC lives) might be much harder for the user to recover from. Using TWC for daily tweeting is similar to using root or Administrator account for daily desktop computing. It should be obvious the regular Twitter use exposes one to a huge attack surface: all those pictures, videos and exciting links one is offered to click are just an accident waiting to happen.

Thus a more reasonable way to approach this problem is to use a Twitter *app* (not necessarily in a dedicated AppVM, even) for all every day tweeting, while keeping the Twitter Web Client, together with its password, in a dedicated AppVM (see twitter-admin VM on Figure 3). Of course one should be careful not to use it for regular tweeting. It would be helpful if Twitter considered offering split Web interface, securable with separate passphrases/keys: one for daily tweeting, and the other for administrating of the account, with ability to easily revoke access of the former from the latter.


[twitter-admin] Twitter / Settings - Mozilla Firefox

https://twitter.com/settings/applications

Applications

These are the apps that can access your Twitter account. Learn more.

TweetDeck by TweetDeck
TweetDeck is an app that brings more flexibility and insight to power users.
Permissions: read, write, and direct messages
Approved: Friday, August 8, 2014 12:48:19 p.m.

Joanna Rutkowska
@rootkovska

Figure 5 It would be convenient if Twitter could split its popular Web Client into a normal "tweeting app" (with access revocable like for all other Twitter clients) and an admin interface (which would *not* be a Tweet client at the same time). TweetDeck, that I use, does not implement some of the features that Twitter Web Client provides (actually it "implements" these features, such as showing the list of followers, by delegating them to TWC).

The described approach to securing of Twitter account doesn't really take advantage of any special features of Qubes OS, other than the cheapness and agility with creation of dedicated AppVMs (in this case: twitter-admin), something that wouldn't be so convenient if one wanted to use a separate machine for this task[25].

Weaknesses of software isolation

Everything written so far has been silently ignoring the potential problem of attacks "against virtualization layer". Let's discuss those now and let's start by categorizing potential attacks against virtualization according to what layer they might be targeting.

We thus could imagine attacks against:

[25] It is very likely that such separate machine would be reused also for different purposes than only as Twitter administration, thus decreasing the overall security. E.g. if this separate machine was also used to keep GPG private keys, then using it also for accessing of twitter.com would suddenly become a security problem.

1. The actual virtualization technology, such as VT-x and VT-d,
2. The hypervisor (e.g. Xen),
3. All the *additional* software used by a virtualization system (e.g. qemu, DirectX emulation, etc).

The attacks against the core processor technology (the 1st group), such as VT-x, VT-d, or the good, old traditional CPU MMU/ring isolation, are considered *extremely* difficult. In fact no attack has ever been presented against MMU or VT-x on Intel (or corresponding technology on AMD) processors *ever*[26]*.* One attack against Intel VT-d, presented by us in 2011[27], was only possible because Intel released systems with half-baked functionality (i.e. without Interrupt Remapping units).

Attacks against the core virtualization software, (the 2nd group) i.e. a baremetal hypervisor, are considered *very* difficult. I believe the difficulty here is comparable to, or better than, attacks against core TCP/IP stack in Linux (core, so not considering all the additional stacks, like WiFi, or drivers).

Finally, the last group mentioned above is where the low-hanging fruits can be found. In Qubes we took special precautions to limit this attack surface as much as possible (e.g. we don't trust qemu, we have no networking in Dom0, we use security-optimized GUI virtualization, we have never used PyGRUB, etc). We're not perfect, but we're doing pretty well, I think[28].

The reader is now asked to quickly compare the above attacks against those relevant in physically separate systems scenarios (Figure 1), remembering those systems must have networking and/or USB (or DVD at least) stacks and drivers to be useable for anything in practice, and that these must be part of the TCB. Does it seem more likely to exploit a bug in CPU VMX or in Linux TCP/IP stack? Ethernet driver (or WiFi) or Xen? Qubes qrexec daemon or USB/DVD/filesystem processing code?

Another area of concern for virtualization systems (i.e. other than privilege escalation attacks) is data leaks between VMs. Here we can distinguish between two different kinds of leaks:

1. Cooperative covert channels – i.e. when malware in two (already compromised) VMs conspire together in order to exchange data using unauthorized communication channel,
2. Side channels (non-cooperative covert channels) where malware in one VM tries to learn some facts about processes executing in other (non-compromised) VMs, e.g. in order to guess private key bits.

It is important to stress how significantly different impacts on security those two potential attacks present. While first is mostly irrelevant for most work flows, the second is *fatal* in essentially any scenario.

[26] In public, that is. But consider that essentially all other software got exploits presented in public.
[27] *Following the White Rabbit: Software attacks against Intel(R) VT-d technology* (http://www.invisiblethingslab.com/resources/2011/Software%20Attacks%20on%20Intel%20VT-d.pdf)
[28] The reader is encouraged to read Qubes Security Bulletins we have issued thus far: https://wiki.qubes-os.org/wiki/SecurityBulletins

The cooperative covert channels are possible through a variety of surprising means, such as CPU caches timings[29]. They often offer a very small bandwidth (of the order of bits/s, sometime tens of bits/s) and are often very unreliable (especially in a “noisy” desktop system like Qubes OS running a dozen or so of VMs all at the same time). It is generally considered that elimination of such cooperative covert channels is very difficult or impossible on COTS x86 hardware. Usually any attempts to mitigate those communication channels have negative impact on system performance. This is a popular topic among designers of systems used in military environments.

Let’s make it clear that Xen, neither any other mainstream hypervisor, I think, does not try to limit cooperative covert channels in any way. In fact, it is likely that Xen, in addition to the platform-specific covert channels, also introduces a good deal of additional inter-VM cooperative channels, much more reliable and of significantly higher bandwidth[30]. As noted above, because these are mostly irrelevant for most applications of Xen (and Qubes OS!), nobody has really cared to seriously review Xen in order to identify them and eliminate (something that hopefully might be possible with relatively easy patching).

Admittedly, in some specific use cases, such as the previously discussed Tor lockdown, it might be desirable to limit the cooperative covert channels tough. E.g. we can imagine that both the AnonVM *and* NetVM (refer again to Figure 2) got compromised and they could conspire now together in order to de-anonymize the user. But then, again, if we used physically separate Tor proxy (as in Figure 1) we are not free from this problem either – there are also many potentially covert channels between two network-connected (especially WiFi-connected) systems[31]! Nevertheless, we plan on looking more into this issue in future Qubes OS releases.

Finally, let’s discuss the side-channels. There have been many surprising side channels described in academic literature[32], based on observation of seemingly unimportant details of execution environment, such as power consumption or EM leaks. There is no reason why similar attacks, should not be possible in virtualized environments, where observation of some of the effects of execution of other VMs is even easier measurable (e.g. CPU cache evictions or branch prediction). Such attacks are almost exclusively targeted as crypto operations with a goal to extract (at least some) bits of private keys.

Most (all?) proof of concept side-channel attacks on virtualization systems, described in academic literature[33], have been designed to be used against *server* systems. This is often because in such scenarios the attacker has significantly higher control over the crypto operations that are targeted by the exploit – e.g. the attacker might be connecting to some SSL server, forcing it to conduct certain crypto operation over and over again, increasing the chances of the attack to succeed. A desktop environment, such as this on Qubes OS, where the user might be invoking e.g. GPG

[29] The academic literature on the topic is so vast that I won’t even attempt to provide any references here.

[30] See e.g. http://www.eicar.org/files/xencc-slides.pdf (but likely there are more channels in Xen).

[31] Here Qubes OS might offer some advantage, thanks to separating the actual AnonVM(s) from NetVM via TorVM and FirewallVM, where each might be used to minimize networking-based covert channels. This convenience is not available in a scenario discussed on Figure 1.

[32] See http://en.wikipedia.org/wiki/Side_channel_attack as well as https://privatecore.com/resources-overview/side-channel-attacks/ for a collection of references about attacks in virtualized environments.

[33] For reasons not entirely understood, the academics rarely publish any proof of concept code.

encryption/decryption operations only a few times per day, is significantly more challenging. I thus personally don't consider my GPG keys to be especially at risk from side channel attacks from my other VMs, and I do worry more about other side-channels attacks much more (e.g. sniffing of my passphrase via EM-leak)[34].

It's generally believed that the best protection against these attacks is via careful implementation of the crypto operation (e.g. ensure each loop iteration takes the same amount of time, regardless of the value of the n-th bit of the key).

In any case, it is a mistake to think that side-channel attacks on crypto could be avoided by using separate machines[35].

Bottom line

We have seen how the use of smart software compartmentalization (aided with IOMMU) could allow mitigating lots of security problems that otherwise are very hard to address properly.

Let's look again how that magic have happened: large interfaces have been reduced to much smaller ones, something that was possible because of the simplified environment we operate in (within one physical system, rather than on different systems, so no need for more complex communication protocols), plus we used unique hardware isolation technology, namely IOMMU/VT-d. Additionally, software compartmentalization allows for much cheaper and agile containers creation, which in turn allows for much more fine grained thread mitigation.

Ultimately it's all about the complexity of the interfaces exposed between the hypervisor, VMs, and the outside world.

Qubes OS that has been used as an illustration to our discussions is *not* a magic piece of software. In fact, it is a collection of extremely pragmatic pieces of software. Yet the synergy of these pragmatic building blocks, is quite powerful.

[34] For people who still feel uneasy about the side-channel attacks, Qubes offers a poor-man's solution: to shut down, or pause, all the other VMs in Qubes for the time when the crypto operation is being performed.

[35] http://www.tau.ac.il/~tromer/acoustic/