

# A Usability Evaluation of Qubes OS

Austin Shafer

amshafe2@ncsu.edu

North Carolina State University

Raleigh, North Carolina

## CCS CONCEPTS

• **Computing Security**; • **Human-Computer Interactions**;

## KEYWORDS

Computer Security, Usability

## 1 ABSTRACT

The Qubes operating system [7] is somewhat unique in that it constrains applications in individual virtual machines, requiring the user to change the way they think about interacting with the desktop. In this system usability has a profound effect on security, as user mistakes can undermine the strict isolation Qubes provides. We evaluate if Qubes has successfully incorporated virtual machines into its user interface in order to determine whether other operating systems should consider incorporating their own security primitives. We use Nielsen's heuristics [14] as a framework for analyzing common tasks in Qubes, and find that in most situations Qubes follows usability best practices. Additionally we find room for improvement in system dialogues and the ability to accommodate new users. Our results show that it is possible to design a usable interface around isolation mechanisms, and that there is more progress to be made in this field.

## 2 INTRODUCTION

Qubes is an operating system built on the Xen hypervisor [1] that follows the "security by compartmentalization" ideology. All applications are run inside of virtual machines, also known as AppVMs, which the user has full control over. AppVMs communicate with the privileged Xen domain (dom0) where the desktop environment runs as a traditional X11 server. An application running in an AppVM is visually the same except for a colored border around the window identifying what virtual machine it is running in. Qubes even takes this a step further, running system resources like the network stack and firewall inside their own VMs to increase security.

Any graphical desktop interface takes the technical abilities of an operating system and allows the user to interact with them in an easy and non-technical manner. The desktop environment needs to handle both implementation and usability problems associated with running applications in a workspace. It needs to handle graphical presentation, client communication, multiple applications and workspaces, and

other sensitive tasks. This interface is the stage on which an operating system presents its features.

Most popular operating systems are interacted with through a traditional set of windows and menus, but Qubes requires its users to learn a slightly new way of thinking. Security primitives such as virtual machines are usually ignored by all but the most technical personnel on popular operating systems, but they are a basic building block in Qubes that the user regularly interacts with. It tries to decrease the friction required to access security features in an effort to increase their use in day to day tasks. This comes with its own drawbacks, as the user needs to be much more knowledgeable on both the interface and the virtual machine technology it uses.

Qubes is an interesting case study because it has placed an emphasis on developing the usability of the system despite deliberately targeting a user base experienced in security. Qubes is well known for its security contributions, but less heralded for its attempts to make those contributions easily accessible to the user.

In this study we will be performing a usability evaluation of Qubes using Nielsen and Molich's nine usability heuristics [14]. These heuristics will provide a framework for analysing particular tasks that a user might commonly be required to do. The tasks chosen were creating an AppVM, launching an application, and copying a file between two AppVMs.

In this paper we present the following contributions:

- Perform a heuristic evaluation on common operations in Qubes OS
- Provide feedback on how Qubes' security related interfaces may be improved
- Based on the results of the heuristic evaluation, recommend if and how other desktop interfaces should consider incorporating their own security primitives.

We find that Qubes does its best to follow usability best practices, and makes some exceptions in areas where security would be negatively impacted. We explain the reasoning behind these deviations and document areas of concern in the interface. Finally, we argue that this security-centered usability design can be adopted by other projects.

## 3 METHODOLOGY

Our goal in this study is to examine the usability of the Qubes operating system through expert analysis. In order to reduce

the scope of this analysis, we will only examine a few short tasks that are common in any use of the system: creating an AppVM, launching an application, and copying data from one AppVM to another.

Applications execute in their own virtual machines, called AppVMs. AppVMs can run one application or many, and can be persistent or disposed of as soon as their application(s) finish executing. Because an AppVM is an important building block of the Qubes desktop we will be analysing how intuitive it is to create and manage a new AppVM. Although users may have experience with virtual machines they probably are not used to managing them as part of a graphical desktop.

Launching an application in an AppVM is also one of the basic tasks all users in Qubes will perform. We include it for analysis because it is unique compared to other desktop environments. Users are not normally required to think about what context their applications are executing in, and this new mindset should be communicated as comfortably as possible.

Likewise, copying data between applications should be both functional and easy to understand or else users will not be productive when using the system. Although AppVMs are meant to be separate, using a file from one application in another is still a must-have feature. We evaluate two types of data transfer: copying a file to a different AppVM, and copy/pasting between AppVMs. This is conceptually the simplest task, but it is important that the user can understand where data is located and why access is so restricted.

We will use Nielsen's nine usability heuristics[14] to form the basis for our comparison, as they are a well established set of best practices. Each task will be compared against all of the heuristics to see if the recommended practices are followed or ignored. While completing a task we will first explore the interface naturally, without help. If we are stuck and do not know how to complete the task we will then refer to the official online documentation.

Nielsen and Molich's Usability Heuristics
Simple and natural dialogue
Speak the user's language
Minimize user memory load
Be consistent
Provide feedback
Provide clearly marked exits
Provide shortcuts
Good error messages
Prevent errors

For each task under analysis we will label each heuristic as being followed or ignored, along with some comments as to why that heuristic was labelled such. The comments will also communicate if a heuristic is only partly followed or

ignored. After commenting on each heuristic there will be a discussion of specific examples of the interface and how they impact both the heuristic evaluation and the user experience.

We will try to document any cases where the recommended practices were intentionally disobeyed for a clear reason. In some cases decisions like this might actually make sense. This should hopefully give us an idea of how focused and intentional the developers were on usability as they developed the user interface.

The author will complete the same general tasks as will be examined as part of the expert evaluation: create an AppVM, launch an application, and move data from one AppVM to another. Although it is important that they do all three tasks at some point, they will be encouraged to explore the system and try to use it however they see fit. We are interested in how much conflict arises when a new user is dropped into an interface they are not familiar with, and how accommodating Qubes makes the transition.

By analysing the usability of basic tasks we can recommend if other desktop-based operating systems should expose security related functionality in the graphical interface.

#### 4 EXPERIMENT SETUP

All testing was done with Qubes 3.2.1 in VirtualBox 6.0.18 running on MacOS Catalina 10.15.2. Unfortunately virtualization was required as Qubes was unable to run on the testing laptop we prepared. The laptop we planned on testing with was an old macbook, which Xen does not support. Qubes is not meant to be installed inside of a virtualized environment, and the latest version (4.0.3) did not work in VirtualBox. Unfortunately this means that the version of Qubes tested is noticeably far behind the current development status of Qubes.

Running Qubes in a virtual machine does impact the evaluation. Firstly, it means we are not evaluating the latest version of Qubes. This is obviously an issue if the latest version has fixed or updated interface components that we are evaluating. The host environment also impacts some of the keystrokes recorded by Qubes. Certain shortcuts will not be possible inside the VM, such as fullscreen operations.

Evaluation was performed by the author, who is a graduate student in computer science experienced in operating system development. They have regularly used MacOS, Windows, Linux, and FreeBSD in a desktop setting for multiple years and have contributed to FreeBSD, JunOS, and the seL4 microkernel. The evaluator is familiar with virtualization technologies, and has used Virtualbox and KVM regularly.

#### 5 RESULTS

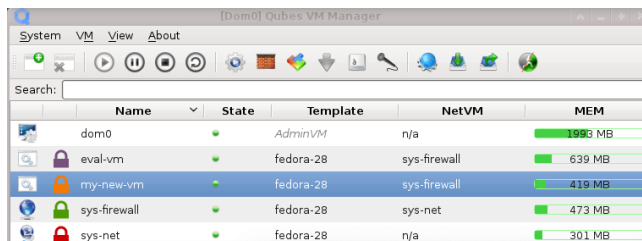
Three tasks were evaluated: creating an AppVM, launching an application in an AppVM, and copying data between two

AppVMs. Each task has been broken down based on the usability heuristics observed, with comments per heuristic.

### Qubes VM Manager

One of the most central applications in Qubes is the virtual machine manager, which is used to create domains to execute applications in. It handles AppVM templating, permissions, network settings, and VM settings. Due to its importance we evaluated a common task required of users: creating an AppVM. Some additional interactions were also evaluated, such as viewing all VMs, as they are a part of the natural use of the Qubes VM manager.

Overall this task went smoothly, but when technical issues were encountered the dialogues were significantly less friendly. One of the error messages observed when starting an AppVM was "qrexec not connected". This message is completely unintelligible to any casual user, and gives them no clues as to what action they need to perform. Even if they look up qrexec and understand its role in Qubes' communication model between AppVMs, they still have no clue what they did wrong and what remedy is required.



	Name	State	Template	NetVM	MEM
	dom0	●	AdminVM	n/a	1993 MB
	eval-vm	●	fedora-28	sys-firewall	639 MB
	my-new-vm	●	fedora-28	sys-firewall	419 MB
	sys-firewall	●	fedora-28	sys-net	473 MB
	sys-net	●	fedora-28	n/a	301 MB

Figure 1: The Qubes VM Manager

*Simple and natural dialogue:* Overall the dialogues contain the minimum amount of info needed to effectively process the system. Messages, such as starting a VM, are intended to be simple and quickly understood. Pop-up dialogues are terse and easy to read at a glance.

*Speak the user's language:* Lots of Qubes specific vocabulary is used in the VM creation dialog, such as AppVM, HVM, DisposableVM. Most of the definitions may not be clear to new users, but they would be obvious to experienced ones. This might not be considered the language of a new user, but it is all vocabulary that they will quickly learn.

Error messages related to technical failures are extremely foreign to any user not involved in the development of Qubes, which is a real problem. The most obvious of these is the 'qrexec not connected' message mentioned in the task summary. Technical error messages should be hidden in the system logs for more tech-savy users to read.

*Minimize user memory load:* Running many virtual machines increases the load on the end user as they need to manage and update all VMs individually. However, most of the memory load required of the user involves mentally tracking what data is in what AppVM. Users have full control over where they place data and where they install applications. The complexity is entirely dependent on how the user organizes their AppVMs. The VM manager by default only shows running VMs, but can also display all active and inactive VMs.

*Be consistent:* There are obvious patterns and notifications in the interfaces icons that the manager communicates to the user. Colored dots naturally convey the running state of each AppVM. Graphically the system is consistent, with all actions using the color assigned to their respective domain. Most any icon can be hovered over to display a text hint of its function, which is especially helpful to a new user.

One exception to this is the VM state, a colored dot showing if the AppVM is running, which does not show any text hints. Most of the status colors are obvious (i.e. green for running) but some of them, such as grey signifying a paused VM, are not.

*Provide feedback:* Most actions are confirmed with a change in icon state or a popup window. Sometimes the pop shows that an action has completed, but the result of the action is not visible. An example of this is observed when a new VM is created. The manager is only showing active VMs so the newly created VM will not be on the list.

*Provide clearly marked exits:* Qubes provides window controls similar to most common operating systems. Users can always press the exit button in the top right of a window and can find buttons to cancel or confirm an action before it takes place.

*Provide shortcuts:* AppVM creation is too complicated for any keyboard shortcut system. Qubes does allow for template VMs, which act as a shortcut allowing the user to quickly create VMs of a well-known type. Many templates are provided in the base installation, most of which are common Linux distributions that the user can try. Creating a new VM from a template streamlines the process and reduces the workload on the user.

*Good error messages:* Error messages related to user actions are easily understood, such as there not being enough available memory to launch VM. Usually the problem is trivial and is easily identified, such as not having enough memory to start a VM. Technical error messages are a completely different story, as they seem to be written for the designers instead of the end users. Again, a good example of this is the qrexec message mentioned in the summary.

*Prevent errors:* The VM manager does an effective job of walking the user through any particular action. Because the process is very structured, it is both easy to use and hard to trigger errors. The manager only presents valid options during configuration, so anything the user chooses will work. Usually the user is choosing from a list of templates, which are all valid and supported configurations.

## Launching an Application

Launching an application in an AppVM is a mandatory interaction for any user of Qubes. Luckily the process is very simple, and is communicated well. As part of this task, we also evaluate the presentation mechanism of the launched application, as users need to be able to interact with it in a clear and secure manner.

Qubes provides a shortcut menu for quickly launching applications through a list of AppVM Shortcuts. These applications are sorted by AppVM, so the user can know what domain they are launching the application in. There is some basic automated detection of applications, so if the user installs a new program in a VM it will show up in the shortcuts menu.

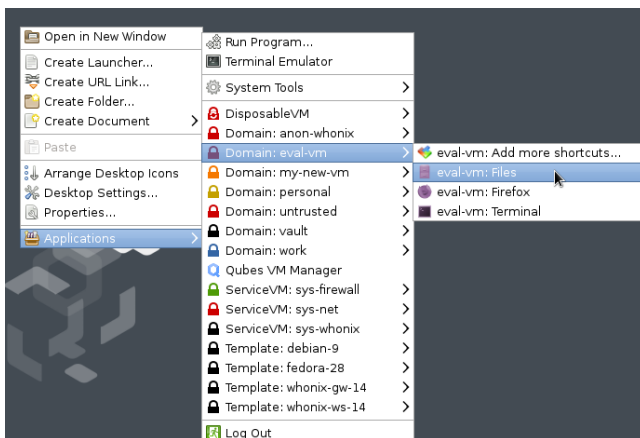


Figure 2: The application shortcuts menu.

*Simple and natural dialogue:* Due to its mostly graphical nature, there is limited dialog in this task. The application shortcut menu is well organized and the titlebars hold the name of the domain the application is executing in for additional clarity. The shortcuts themselves have their names derived from the application they launch.

*Speak the user's language:* Users will find that this task follows the keywords and actions that they have come to expect. Right clicking shows an application menu, which shows a sorted list of programs. Most of the dialog encountered is common to other windowing systems or, in the case of VM names, is set by the user themselves.

*Minimize user memory load:* The cognitive requirements for this are minimal, as all of the information is presented graphically. The user might need to remember in which AppVM they installed the application they want to launch, but this can easily be found by poking around in the shortcuts menu.

*Be consistent:* Both the window controls and the shortcut menu follow consistent design patterns. The window titlebar has minimize/maximize/close buttons familiar with any desktop user, and the titlebar matches the color assigned to the AppVM. The shortcut menu also displays assigned colors for each AppVM, and common utilities such as file browsers show up in all AppVMs.

*Provide feedback:* While Qubes does provide application and VM launch feedback in the form of pop notifications, sometimes an AppVM might have to be started before the application can execute. If the VM takes a long time to start up, it may seem like nothing is happening. Feedback on the state of the AppVM can be seen in the VM manager, but the user needs to know to look there.

*Provide clearly marked exits:* For normal operation this practice is followed, as all titlebars have a close button.

However this goes downhill quickly if the user makes an application fullscreen. The usual titlebar is no longer visible, there are no visual indicators for how to undo the fullscreen operation. The user needs to know the non-obvious keyboard shortcut to undo this (Ctrl-F8 and Ctrl-F9). This shortcut did not work in Virtualbox on MacOS during our evaluation. The user can still Alt-Tab to escape, meaning this is only a usability problem not a security issue.

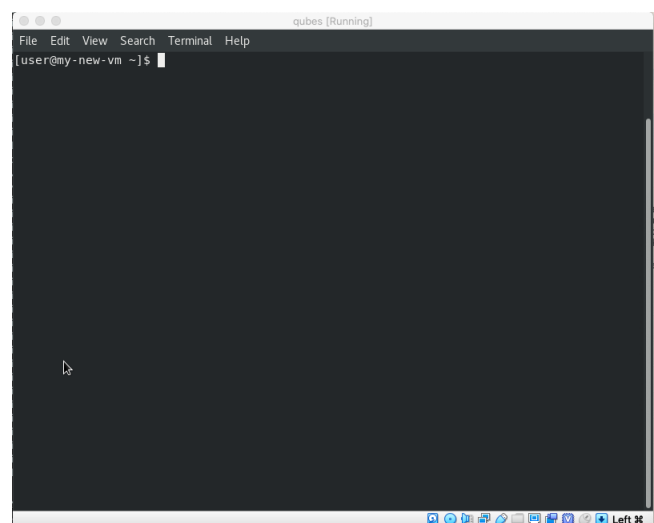


Figure 3: The terminal application in fullscreen mode with no apparent method of escape.

*Provide shortcuts:* There are some normal keyboard shortcuts for closing windows and such, but the fullscreen related keybindings are terrible. Qubes inherits these keybindings from KDE, since it is the desktop environment Qubes is derived from. The user is free to rebind these shortcuts by modifying a configuration file, but that is hardly a usability focused solution.

A much better way to handle this problem is MacOS' approach. When the user hovers their cursor near the top of the screen the titlebar reappears, and the user can minimize the window. A graphical solution like this would prevent users from getting lost in a fullscreen application.

*Good error messages:* Because this test was conducted using the file browser from an AppVM template, no specific errors were encountered. Any errors during AppVM startup will mirror the previous task where we evaluated the Qubes VM Manager.

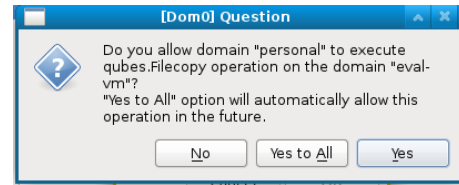
*Prevent errors:* Again, no errors of any kind were observed. The file browser was chosen because it was known to work, and the application launcher only showed valid options to prevent any issues.

### Copying Data Between AppVMs

AppVMs provide security through their isolation, but certain tasks require data to be accessed from a separate VM. It is important that data movement capabilities are provided in a secure manner as this has the potential to interfere with the strict isolation normally provided by Qubes. We analyse two common ways of sharing data: copying files between AppVMs and Copy/Paste. We find that Qubes does a very good job of ensuring the security of the user while performing these actions, but could still use some minor usability improvements.

One of the recurring techniques used is the use of explicit actions to prevent accidental mistakes. File copy requires the user to type the full name of the destination VM and copy/paste requires extra keystrokes. This aids security at the expense of usability. In most systems this trade-off would not be worth it, but leaking data between AppVMs in Qubes undermines the strict isolation of the system.

*Simple and natural dialogue:* Although there is limited dialog, it is succinct and to the point. When fetching the AppVM's clipboard, popups inform the user of the next action they should perform. These popups usually time out and disappear before the user has enough time to read the entire message, but the user can hover their cursor over the message to prevent this.

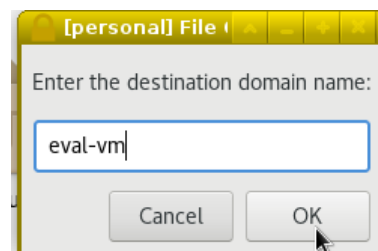


**Figure 5: Requesting confirmation to perform a copy operation.**

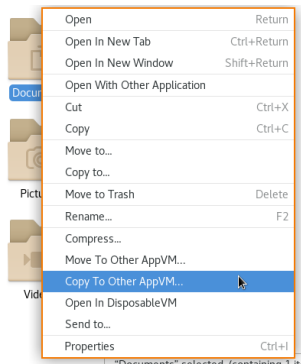
*Speak the user's language:* All dialog is built on established vocab. There is some new vocab related to clipboards, but this is identical to other desktop systems the user is familiar with. Again there is opportunity to provide an introduction or help text to aid a new user who might not have read the manual yet.

*Minimize user memory load:* Copying data isn't a trivial or effortless operation, but this is by design. Qubes forces the user to be purposeful about the transfer they are initiating, whether that be by making them type the destination AppVM name or explicitly sharing the copy buffer. The user has to actively think while performing both tasks, decreasing the likely hood that they make a mistake and leak sensitive information. The load on the user might be increased, but the potential for disaster has been decreased.

*Be consistent:* Copying files between VMs had excellent usability, security, and consistency. Right clicking in the file browser to share between AppVMs is the type of interface the user expects, and even new users can complete this with relative ease.



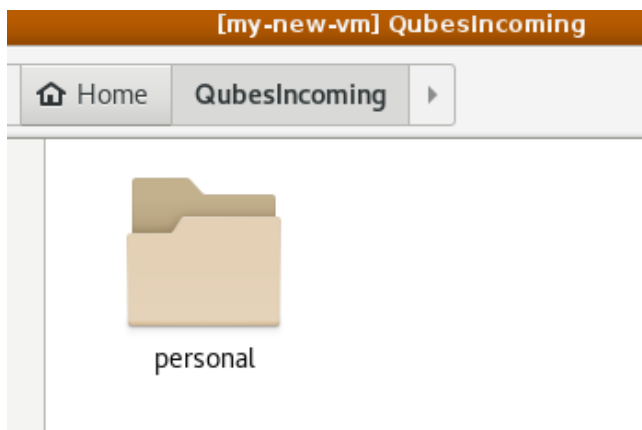
**Figure 4: Initiating a file copy requires explicit specification of the destination AppVM.**



**Figure 6: Right clicking in the file browser shows an intuitive copy dialog.**

Copy/Paste was a much different experience. There is no right click menu for sharing the AppVM's copy buffer and the user needs to know a non-trivial set of keyboard shortcuts without being provided any hints. The documentation online is quite nice, but unless a user actively searches online they might not even realize that inter-AppVM copy/paste is possible.

*Provide feedback:* When copying a file to another AppVM, there is no feedback on where that file went. The user needs to know files are always copied to the 'QubesIncoming' folder in the destination VM. This could easily be solved with a popup saying that the copy completed and the file was placed in 'QubesIncoming'.



**Figure 7: The 'QubesIncoming' folder showing a directory holding files copied from the 'personal' AppVM.**

When performing the keyboard shortcuts for copy/paste, popups will display that the copy buffer has been shared. These popups timeout quickly, which is appropriate considering that users will only want to read the popup the first time they perform a copy.

*Provide clearly marked exits:* Windows related to file sharing are familiar, and Qubes has designed their interface to make sure that there is no easy way to accidentally share information. If the user wants to stop a copy they can just close the window, or not perform the 4 shortcuts needed to copy and paste data.

*Provide shortcuts:* The shortcuts for copy/paste are more complicated than other systems. The user must copy the data to the AppVM's copy buffer, then fetch that buffer and share it with another AppVM. Finally they perform a paste in the destination AppVM. Although this hurts usability it is motivated by security, as it ensures that copy buffers are not shared without user intervention.

*Good error messages:* If an incorrect VM name is specified Qubes will inform the user. There is potentially one missing error message: if the user tries to share a clipboard when none has been selected nothing will happen. Ideally this should create a popup saying that there was no fetched clipboard to share.

*Prevent errors:* No errors were observed. This is another example of Qubes only presenting valid operations to the user. Because of the complexity of these seemingly normal operations the user is less likely to make a mistake and compromise their security.

## 6 FUTURE WORK

Due to our evaluation being performed on an out-of-date version of Qubes, it is hard to recommend Qubes-specific improvements. The problems found in our evaluation were minor, and fixes like changing shortcut keys are trivial. The developers have obviously kept usability in mind, and are continuing to do so moving forward.

The Qubes interface is easy to explore and discover, but there is room for improvement in the way they introduce new users to the interface. After booting for the first time Qubes could open the top-level documentation page in an AppVM's web browser to help the user get their bearings. The online documentation is very helpful, and routing confused users to it would help them learn the system easier. There are many popups providing feedback on Qubes specific actions. Allowing an ability to click on such a popup to open the relevant documentation would also be a useful measure.

The interface design present in Qubes does afford some possibility for reuse by others, although direct applications are not trivial. One major problem is the overhead that AppVMs incur, which could be solved by using a more lightweight security primitive such as a container or sandbox. Most modern desktops provide mechanisms to sandbox applications, and Qubes provides a reference implementation of how to expose information to the user about the execution

context of their programs. Other interesting ideas, such as Qubes' approach to copy buffer sharing, could also be used by other operating systems due to their efficient prevention of accidents which could compromise security.

## 7 RELATED WORK

There are a variety of desktop environments and operating systems focusing on isolation. CapDesk[10] runs applications and application subcomponents in "caplets" which are isolated from each other. CapDesk tries to aid usability by making all caplet operations completely invisible to the user, and it is the first capability based desktop with a traditional point and click interface.

Virtics[16] is a virtualization based operating system whose design is similar to Qubes[7]. Both projects are built on the Xen Hypervisor[1] and run applications in virtual machines, which communicate over the X11 protocol[19]. A hypervisor was chosen for its effective isolation of virtual machines while still retaining flexibility[12].

Unlike Virtics, Qubes also runs some core operating system facilities, such as the networking stack and firewall, in separate virtual machines. The authors of Virtics do value usability as a primary feature, and they perform a simple usability evaluation.

SAFE-OS[9] is another predecessor to Qubes, but unlike Virtics the authors do not evaluate the usability of their created system. Their usability evaluation consists of a performance benchmark to show that latencies are tolerable. Of the three, Qubes is the only project seeing any continued development or real world use.

Numerous frameworks for evaluating the usability of a system have been proposed. One of the most popular is Nielsen and Molich's Heuristic Evaluation [13][14][11]. They outline 9 general heuristics by which an expert can use as a guideline for analysis of a system. Other contributions, such as [5], propose general principles which should be followed during development to ensure that the resulting product. [5] additionally tries to answer why certain "obvious" guidelines are not followed in practice. Similar frameworks [21] instead analyse security from a usability perspective in an attempt to align both usability and security goals.

[8] and [3] are both examples of formal evaluations of application usability in various Microsoft products. They outline problems they found with the software under analysis and draw comparisons to Nielsen's heuristics[14].

User-defined access control [17] is a system for automatically granting permission to resources based on user intent. It provides an effective alternative to pop up dialog boxes, which can condition the user to automatically accept them.

Other resources [15][4] try to provide an overview of the history of usability and security. Although they document

changes in interface design they do not focus much attention on desktop environments themselves. [18][6] provide a collection of usability evaluation methods which can be used in practice, and are very relevant to our evaluation of Qubes.

The inspiration for this paper [20] examines the effectiveness of automating complicated HTTPS configuration with Certbot. It argues that Certbot noticeably reduces the friction required to set up an HTTPS certificate, with the downside that users aren't as knowledgeable on what actions Certbot performed. [2] investigates if this automation is harmful to users in the long run, although they agree that automation in security is necessary.

## 8 CONCLUSION

In this work we evaluated three essential tasks which are part of the Qubes OS desktop interface. We evaluated the tasks against nine usability heuristics, showing that Qubes does its best to follow best practices in most situations, and offer suggestions for improvement where appropriate. We argue that other operating systems can learn from Qubes, and can incorporate its ideas into their own interfaces.

## REFERENCES

- [1] Paul Barham, Boris Dragovic, Keir Fraser, Steven H, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the Art of Virtualization. *ACM SIGOPS Operating Systems Review* 37 (11 2003). <https://doi.org/10.1145/945445.945462>
- [2] W. Edwards, Erika Poole, and Jennifer Stoll. 2008. Security Automation Considered Harmful? (01 2008). <https://doi.org/10.1145/1600176.1600182>
- [3] Steven Furnell. 2007. Making Security Usable: Are Things Improving? *ECU Publications* 26 (09 2007). <https://doi.org/10.1016/j.cose.2007.06.003>
- [4] Simson Garfinkel and Heather Richter Lipford. 2014. Usable Security: History, Themes, and Challenges. *Synthesis Lectures on Information Security, Privacy, and Trust* 5 (09 2014), 1–124. <https://doi.org/10.2200/S00594ED1V01Y201408SPT011>
- [5] John Gould and Clayton Lewis. 1985. Design for Usability: Key Principles and What Designers Think. *Commun. ACM* 28 (03 1985), 300–311. <https://doi.org/10.1145/3166.3170>
- [6] James Hom. 1996. The Usability Methods Toolbox Handbook. (01 1996).
- [7] Rafal Wojtczuk Joanna Rutkowska. [n.d.]. QUBES System Architecture. <https://www.qubes-os.org/attachment/wiki/QubesArchitecture/arch-spec-0.3.pdf>
- [8] J. Johnston, J.H.P. Eloff, and Les Labuschagne. 2003. Security and human computer interfaces. *Computers Security* 22 (12 2003), 675–684. [https://doi.org/10.1016/S0167-4048\(03\)00006-3](https://doi.org/10.1016/S0167-4048(03)00006-3)
- [9] François Lesueur, Ala Rezmertita, Thomas Herault, Sylvain Peyronnet, and Sebastien Tixeuil. 2010. SAFE-OS: A secure and usable desktop operating system. 1–7. <https://doi.org/10.1109/CRISIS.2010.5764916>
- [10] Mark S. Miller. [n.d.]. E and CapDesk: POLA for the Distributed Desktop. <http://www.combex.com/tech/edesk.html>
- [11] Rolf Molich, Rolf, Nielsen, and Jakob. 1990. Improving a Human-Computer Dialogue. *Communications of ACM* 33 (03 1990), 338–. <https://doi.org/10.1145/77481.77486>

- [12] Mihir Nanavati. 2011. Breaking up is hard to do : security and functionality in a commodity hypervisor. (01 2011).
- [13] Nielsen and Jacob. 2001. How to conduct a heuristic evaluation. (01 2001).
- [14] Nielsen, Jakob, Rolf Molich, and Rolf. 1990. Heuristic evaluation of user interfaces. <https://doi.org/10.1145/97243.97281>
- [15] Bryan Payne and W. Edwards. 2008. A Brief Introduction to Usable Security. *Internet Computing, IEEE* 12 (06 2008), 13–21. <https://doi.org/10.1109/MIC.2008.50>
- [16] Matt Piotrowski and Anthony Joseph. 2012. Virtics: A System for Privilege Separation of Legacy Desktop Applications. (05 2012).
- [17] Franziska Roesner, Tadayoshi Kohno, Alexander Moshchuk, Bryan Parno, Helen Wang, and Crispin Cowan. 2012. User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems. *Proceedings - IEEE Symposium on Security and Privacy* (05 2012), 224–238. <https://doi.org/10.1109/SP.2012.24>
- [18] Rubin, Jeffrey, and Dana Chisnell. 2008. *Handbook of Usability Testing*.
- [19] R. Scheifler. 1987. X Window System Protocol, Version 11. (01 1987).
- [20] Christian Tiefenau, Emanuel Zezschwitz, Maximilian Häring, Katharina Krombholz, and Matthew Smith. 2019. A Usability Evaluation of Let's Encrypt and Certbot: Usable Security Done Right. 1971–1988. <https://doi.org/10.1145/3319535.3363220>
- [21] Ka-ping Yee. 2003. User Interaction Design for Secure Systems. [https://doi.org/10.1007/3-540-36159-6\\_24](https://doi.org/10.1007/3-540-36159-6_24)