2018

The Legacy of Multics and Secure Operating Systems Today

John Schriner
CUNY Queensborough Community College

John Schriner
The Legacy of Multics and Secure Operating Systems Today

Part 1: Timeline

The idea of the secure modern operating system began with Multics in 1963. "For secure operating systems, the ideas of reference monitor, protection systems, protection domain transitions, and multilevel security policies" (Jaeger, 2008, p. 23) and other security concepts we take for granted, began with the developers of Multics. It was a revolution, as it was a timesharing system—as opposed to the batch systems that would process jobs in sequence, one at a time. Multics was "designed from the start for security" and "more secure than other contemporary (and current) computer systems" (Karger & Schell, 2002). This paper aims to look at modern "reasonably secure" operating systems in light of the legacy of Multics and the research along the way.

Multics was unique in that "information protection has been permitted to influence the entire system design" (Saltzer, 1974). Saltzer provided five principles inherent in Multics: the default situation should be lack of access; there should be regular audits that maintain current authority; the design should be open and collaboration should be supported by peer review; it should incorporate the principle of least privilege; and lastly there should be ease of use so that the user doesn't have to think about the underlying design (Saltzer, 1974). Most apparent is that Multics was unique but also continued and improved the product of past research (e.g. extending the use of *addressing descriptors* in protecting primary memory—the foundation of which came from a system called Burroughs B5000).

A theme we follow through the development of projects like Multics is the foundation of collaborative development. Known weaknesses are corrected "as rapidly as feasible" (Saltzer, 1974). This could be akin to the present-day open source software movement and the benefits of the ticket/versioning control systems like GitHub, GitLab, or Beanstalk.

Clearly Saltzer, Karger & Schell, Corby, Schroeder, and Vyssotsky were the prominent and

productive researchers of Multics from inception at the *1965 Fall Joint Computer Conference* in Las Vegas.[1] The Association for Computing Machinery (ACM) led annual symposia on *Operating System Principles* as well as publishing Multics research in *Communications of the ACM*.[2] At the heart of Multics was research and scholarly communication in the academic sphere. Their work was to describe the architecture of Multics, offer critiques, and offer solutions and ways to move forward. The best example of this discussion may be Schroeder who offered the paper *Security Kernel Evaluation for Multics* (1975) in which the critiques of Multics lead to the idea that a security kernel should be added and the conceptual work being done to define and structure the security kernel. The stated goal was to be able to perform an audit and verify the correctness of information protection and initial functions of Multics.

The topic of secure operating systems is still very popular among researchers. Although Multics as a project is not heavily discussed in white papers or secure operating system listservs[3], the legacy of Multics is foundational.

1 http://multicians.org/papers.html
A voluminous collection of Multics research with a link to further research collected at the Computer Security Paper Archive Project at UC Davis, California.

2 https://sigapp.org/sac/
ACM is still very active today with the annual and now 32nd Symposium on Applied Computing (2017). Interestingly, Corby won the ACM Turing Award in 1990 and the talk is entitled "Building Systems That Will Fail"

3 https://secure-os.org/desktops/charter/
"Secure Desktops is a mailing list for the discussion of security and privacy challenges in open source desktop computing systems."

Part 2: Annotated Bibliography

This part will look at research from the inception of Multics to current secure operating systems. The articles are arranged alphabetically to aid in locating sources. Each entry addresses a few given questions and an explanation to the credibility and relevancy of the research.

Colp, P., Nanavati, M., Zhu, J., Aiello, W., Coker, G., Deegan, T., ... & Warfield, A. (2011, October). Breaking up is hard to do: security and functionality in a commodity hypervisor. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (pp. 189-202). ACM.

This paper presents Xoar which is a modified version of Xen that retrofits isolation principles used in microkernels onto a "mature virtualization platform" (i.e. Xen). This is necessary because with Xen or Hyper-V hypervisors we see that the TCB is in fact quite large. This large "control VM" houses device emulation, system boot, administrative tools—and it is a liability. To address the problem of a large TCB, the authors note that the history of OS development says to break it into smaller pieces, isolate those pieces, and reduce each to the least privilege necessary to do its tasks. To harden the TCB, Xoar developers have disaggregated Xen's control VM and brought about new capabilities: a bootstrap VM that's disposed of once the system starts; a secure audit log that makes it possible to see a component's dependencies in the case of a vulnerable package; 'rearchitected' critical services using isolation and microreboots that make it so an attacker would have to exploit two isolated VMs to compromise the service.

The TCB is responsible for the security policy of the computer and encompasses hardware, firmware, and software. The Xen hypervisor runs at the highest privilege level so a compromise of any component could potentially allow the attacker access to other services and interfaces (e.g. exploiting XenStore key-value storage, which runs with *dom0* privileges, would lead to system-wide vulnerabilities). Using the CERT vulnerability database[4], the researchers found that 21 of the 23 current attacks ranging from buffer overflows to DOS attacks, were against the service components in the control VM.

dom0 is responsible for exposing devices to the guest VMs. Devices may be virtualized, passed through, or emulated. For virtualized devices, Xen handles the shared memory and keeps the ACL in *grant tables*. For devices to be passed through, Xen can assign direct access of devices to other VMs. Lastly, an emulation layer is handled by a per-guest Qemu instance for "commodity OS's" that expect to be run on a standard platform.

Xoar and *dom0* disaggregation are being pursued in Qubes, and the work of these researchers is certainly influencing the design.[5,6]

4 https://www.kb.cert.org/vuls
5 https://wiki.xenproject.org/wiki/Dom0_Disaggregation
6 https://www.qubes-os.org/research/

Laiserin, Jerry. (2000). The Pre-History of Internet Collaboration.(Technology Information). Cadence, 15(12), 59.

Continuing the central theme of collaboration, this paper traces the history and culture of collaboration in computer science. The author notes that Multics "indirectly spawned the collaborative-computing revolution of today" by inspiring Thompson and Richie to develop UNIX for interactive access on a shared or collaborative basis. UNIX's inherent collaborative nature and Bell Lab's cheap or free licensing of its non-telephone-related technologies meant that UNIX witnessed widespread adoption leading eventually to a lineage including Windows NT, Mac OSX, and Linux.

The author describes the personal computer revolution as a cultural aberration or a detour from the collaboration that preceded it: in the PC world of 1980's users had "my" files in "my folders." Lotus Notes was the first commercially-viable system that connotes "groupware" in its repository of shared documents, threaded discussion, and its support of work "within and among groups of collaborators—hence the term 'groupware' and the slogan 'communicate, coordinate, collaborate.'"

This paper, written in 2000, builds on the work of William Nickerson who crafted "A Taxonomy of Group Computing Applications" in a 1997 issue of *Group Computing* magazine. Of course this discussion predates collaborative coding repositories like GitHub (founded in 2008) and collaborative writing in Google Docs (2006).

Lesueur, F., Rezmerita, A., Herault, T., Peyronnet, S., & Tixeuil, S. (2010). SAFE-OS: A secure and usable desktop operating system. Risks and Security of Internet and Systems (CRiSIS), 2010 Fifth International Conference on, 1-7.

SAFE-OS is a secure operating system project using virtual machine containers with a shared standard desktop interface. This original research looks at related work in both kernel-based containment and Mandatory Access Control (MAC) systems (e.g. SELinux and AppArmor) as well as other virtualization-based containment (e.g. SVFS, NetTop, Bitfrost and Qubes OS). As opposed to a very complex kernel or the high administrative costs of a MAC system, type 1 virtualization technologies like Xen hypervisor provide much simpler and auditable virtualization code that leads to higher trust and security.

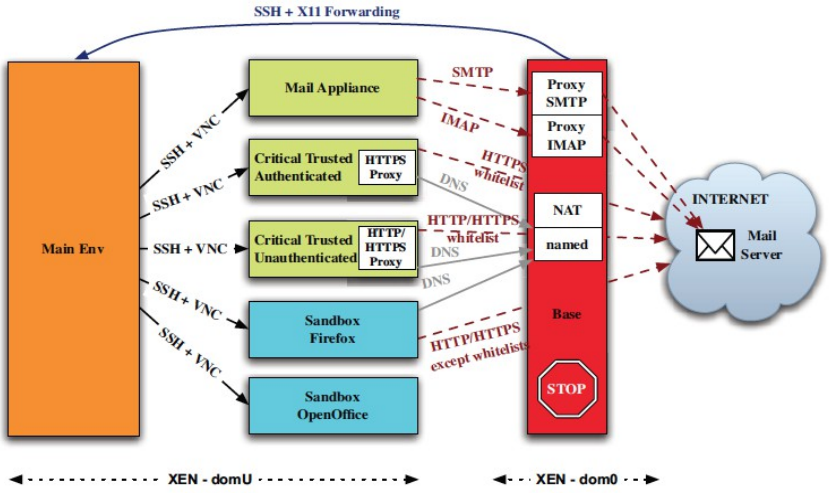
The authors examine the shortcomings of other virtualization projects: in SVFS[7] a corrupted application can access and alter user data as SVFS only secures the integrity of some system files; in NetTop (built on top of VMware and SELinux) the user gets completely isolated VMs with no way to share data securely between them; in Bitfrost the user doesn't have fine-tuned access to the VM but rather generic capabilities like access to the webcam; in Qubes there are no constraints to running any application in any container—the roles are present but not enforced.

The paper describes an exquisite and secure base Xen environment called *dom0* that all communications proxy through. Unprivileged security modules run on specific ports at *dom0* and only "accepts incoming connections to specific proxy services that implement the security modules of the

7 https://github.com/macan/SVFS

appliances." A GNOME desktop environment runs on *Main Env* but itself has no internet access: it solely communicates with appliances "to control and display them."

The OS is designed with three critical appliances: Mail (via Thunderbird)[8], Trusted Websites (whitelisted and https only via Mozilla Firefox), and Unauthenticated Websites (similar to Trusted Websites with the exception that port 80 is allowed). There are also sandboxed VMs: one sandbox is for untrusted websites, and the other sandbox is for Office applications. These sandboxes are, by design, self-contained so even malicious code could be run without the risk of affecting other VMs.



Communications among the VMs (Lesueur, F. et al., 2010, p. 3)

Even the displays are virtualized as each appliance uses an SSH tunnel and VNC to display in *Main Env*. X-server runs on base, *dom0*. SAFE-OS's base VM also performs automatic updates at boot so if there was malicious software (outside of personal files) it would be removed when the VM updates. Transferring files is antagonistic to the containment design so the developers created a trusted GUI for transferring data between VMs. The GUI relies on a file migration policy that takes into account a VM's access to the Internet and whether the file is tagged as 'critical' or 'not critical.' This system relies on warning a user if moving a file may corrupt or expose files in another area.

SAFE-OS is very similar to Qubes OS with containment and isolation as the conceptual core. Unfortunately the project page has been removed from the *Laboratoire de Recherche en Informatique* webpage and it was last archived in 2013[9]; the project is very likely dead.

Rutkowska, J. (2014, August). Software Compartmentalization vs. Physical Separation . Retrieved December 5, 2017, from https://invisiblethingslab.com/resources/2014/Software_compartmentalization_vs_physical_

8 The appliances themselves are proxied and have no Internet access—indeed even DNS is blocked to remove a possible covert channel.

9 https://web.archive.org/web/20130619223002/http://safe-os.lri.fr/
The most recent working version of the project's site in 2013.

separation.pdf

Joanna Rutkowska is the main developer of Qubes OS. This paper discusses the vulnerabilities of physical separation in securing Tor vs. software compartmentalization of Tor via Qubes OS. The paper explains while although some prefer to use physically separate machines (one Tor gateway and one client computer), this has the potential to increase the attack surface: we have to worry about potential attacks on wifi/ethernet drivers. The separate Tor process ends up having as much security as the manipulatable networking stacks it relies on. Another known problem is security in USB handling when, for example, moving files around.

Qubes OS R2 (we're currently at R3.2) is diagrammed here, addressing the 'Bad Internet' and 'Bad USB':

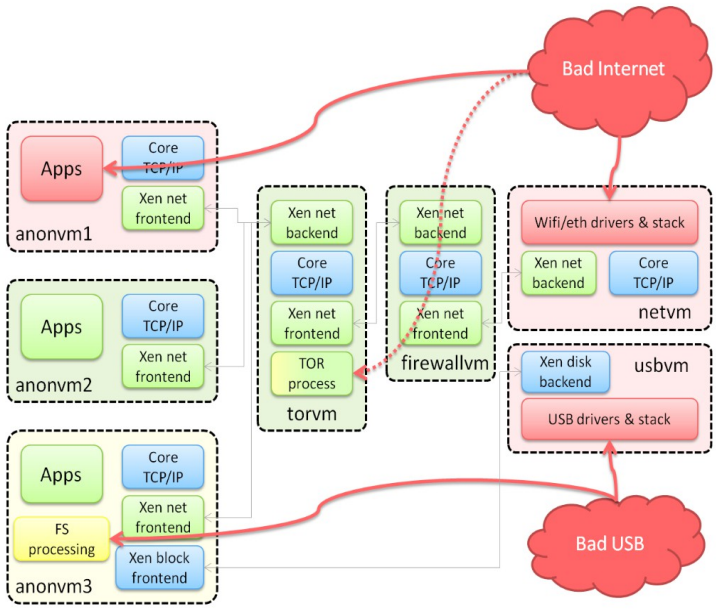


Figure 2 Securing Tor on Qubes OS. As in the previous picture, colors have been used to signalize relative ease of attacks on different components. The discreet blocks encricled in dashed lines represent, this time, Qubes VMs. The other elements of the system and their associated attack surfaces, such 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).

Qubes uses lightweight AppVMs and ServiceVMs. Qubes mitigates the problems above by isolating both the networking and USB stacks in isolated VMs[10]. The purpose of isolating and compartmentalizing is to lessen the attack surface by reducing the interfaces. If NetVM is compromised, even those VMs that use NetVM are vulnerable as NetVM exposes much less code to the other VMs than it does to the "outside world."[11] Given enough development, Rutkowska notes that we could eventually remove all AppVM networking code and work instead with inter-VM pipes.

Qubes mitigates attack vectors via drivers or modules in using the UsbVM similarly to how the NetVM exposes much less to the VMs than would be exposed to an operating system.

"Primitive "airgap" or "security by virtualization" solutions often attempt to enforce a way too

10 Introductory information about Qubes may be found in 'Overview - Admin: An Introduction to Qubes' (2016)

11 Rutkowska notes that although MITM attacks with a compromised NetVM would be possible, such attacks like sniffing traffic are also possible on local networks—the solution, naturally, is to always encrypt traffic.

> 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."

This paper is useful in determining whether or not Qubes and other current "reasonably secure" operating systems are working towards the security of what we had with Multics or doing something entirely different.

Shrobe, H., & Adams, D. (2012). Suppose We Got a Do-Over: A Revolution for Secure Computing. Security & Privacy, IEEE, 10(6), 36-39.

"Back in the 70's, computer security was a niche concern; today it affects all of us." Howard Shrobe of DARPA has led two nascent programs that look at how we would design operating system security from a clean slate. A clean slate would not be scrapping all prior knowledge and visionary work but rather to design without worrying about supporting legacy architectures—we could be free to "draw inspiration from a variety of sources that might help us design secure, resilient systems." The first creative exercise was to imagine if we had the computing power and the transistors that we enjoy today back when Multics was so limited by its computing power—how would we design differently?

Shrobe and Adams took inspiration from computer science research but also from biology. The analogy is illuminating: all biological systems develop in the presence of predators and pathogens whereas computer system architectures were developed in benign, isolated environments; biological systems have excess of resources whereas computer systems have had limited system resources until Moore's law makes resources virtual free; in biology, design is driven by Darwinian pressure whereas computer systems have been driven by a market desiring "cheap systems filled with features;" ecological pressure has led to diversity in species but in computer science it has led to "virtual uniformity in computer design." The conclusion in this exercise is that in biology we have *innate immunity* with an additional *adaptive immunity* to new pathogens: biological systems function even though they are constantly under attack whereas computer systems are brittle.

CRASH (Clean-Slate Design of Resilient, Adaptive, and Secure Hosts) is a series of projects that address issues in computer security and approach these problems by revisiting earlier computer science research into complete mediation, least privilege, and dynamic enforcement. One project goes in the opposite direction from protection rings to capability systems that enforce capability access rights. Another project tests all inputs in a *crumple zone* which is virtual machine with completely mediated interactions with the host.

This paper is useful because it helps us to step back and envision the architecture (i.e. a do-over) of secure operating systems knowing what we know now and with a clean slate on which to build.

Toomey, W. (2010). First Edition Unix: Its Creation and Restoration. IEEE Annals of the History of Computing, 32(3), 74-82.

Upon the resurrection and restoration of the first UNIX's assembly code, Toomey provides the features of the first UNIX operating system as well as the philosophical and technical underpinnings—many gleaned from Multics and other concomitant operating systems like the Berkeley Time Sharing System. The first UNIX was on a PDP-7 and had some notable Multics ideas including: a tree-structured file system, a command line interface with the name of *shell* that was borrowed from Multics; text files consisting of sequences of characters separated by new lines; and the semantics of I/O operations like read and write which would be used to obscure the 'underlying disk blocks' for the user and instead refer to file handlers, a buffer, and a count. Indeed, Dennis Richie notes: "UNIX wasn't quite a reaction against Multics.. Multics colored the UNIX approach, but didn't dominate it."

The first UNIX brought a vast simplification to I/O semantics:
e.g. *ls* > *xx* in UNIX to Multics':

```
iocall attach user_output file xx
list
iocall attach user_output syn user_i/o
```

Toomey notes that it's a testament to the system design that nearly four decades after the first UNIX, nearly all of the 34 system calls available to the user are still present in the UNIX kernel and similar systems: e.g. *open()*, *close()*, *fork()*, *wait()*, *exec()*. The first UNIX on the PDP-11/20 was a multi-user environment supporting six simultaneous users and a kernel of 4200 lines of assembly code and taking up only 16kb of main memory.

Toomey echoes the overarching theme of collaboration in his telling of the history of the first UNIX operating system. Although there was a loss of momentum after losing the Bell Labs funding and being denied proposals for the PDP-10 by management, it didn't stop their research. The first UNIX operating system had a "sophisticated environment" for both typists/users and programmers as it supported several programming languages.

This paper is useful to our discussion because it helps to reveal, as the restoration of the first edition UNIX did, that UNIX developers built atop the legacy of Multics but went in their own direction.

Part 3: Current Research

Current research in this area is vast and it grows with each new development in secure operating systems. The discussion is prevalent at technology/hacker conventions worldwide. Researchers often present white-papers on their lab webpages or they publish in scholarly journals. The scholarly communication of these researchers has also moved to Twitter and GitHub project pages. In this way the discussion is alive.

There are several conferences where papers like those above are presented. In New York City, there is the bi-annual Hackers On Planet Earth conference, in Las Vegas there is DefCon and BlackHat, in D.C. there is SchmooCon, and in Germany there is the annual Chaos Communications Congress. There are often presentations ranging from *pwning* devices/automobiles/voting machines, to innovations in secure operating systems and the underlying technologies. ACM and its special interest groups host more than 170 computing conferences, workshops, and symposia yearly.[12] On the more local or regional level we see the planning of BSides. BSides function as structured or 'unconference' events for security-focused talks and workshops.[13]This is reminiscent of similar events around the globe called Cryptoparties in which participants learn and teach the basics of securing their machines and communications.[14]

Several common themes can be extracted from the papers above. The idea of threat modeling is something discussed in early secure operating system design—and it continues today. Threat modeling can be defined as a consideration of how a system "achieves its *security goals* under a set of threats" (Jaeger, 2008, p. 22). In discussing Qubes OS, we see that threat modeling (sometimes called an Adversary Model) weighs heavily in the design of the operating system components (TorVM, 2017). We can pair the use model with the threat model and we have a better understanding of how to secure a

12 https://www.acm.org/conferences/about-conferences
13 http://www.securitybsides.com/w/page/12194156/FrontPage
14 https://www.cryptoparty.in
Usually this will include key-signing for GPG, setting up Tor Browser or relays, or booting into Tails.

machine (Rutkowska, 2012). An online tool has been developed by the Electronic Frontier Foundation to help users build and assess their threat model and act accordingly.[15] "Threat modeling helps you identify threats to the things you value and determine from whom you need to protect them" (Assessing Your Risks, 2017). With the growing knowledge of NSA and other intelligence agency's penchant for mass surveillance (as opposed to targeted surveillance with accountability), more people are becoming aware of their security needs and using terms like "threat model" when discussing their communications.

"If you're serious about security, Qubes OS is the best OS available today. It's what I use, and free."

— Edward Snowden, *whistleblower and privacy advocate*

@Snowden. (2016, September 29)

Among the most important common themes is the reliance on current research. Each project above is *research-driven.* From Qubes OS[16], to Tor Project[17], to Subgraph[18], to Tails[19], research drives the development. All of these projects are linked via collaboration with Tor Project—as the Tor protocol and Tor Browser are present in each of these distributions. To look at SAFE-OS above, we can piece together how that lab's work informed the work of those trying to scale down or virtualize services to limit attack vectors. Differentiating between most Linux operating systems' reliance on X-based servers as the GUI-server, Qubes developers furthered the idea of GUI-level isolation that we find also in the work above on SAFE-OS.

In a similar vein to noting the value of research, it was essential above to include a paper on

15 https://ssd.eff.org/en/module/assessing-your-risks
16 https://www.qubes-os.org/research/
17 https://research.torproject.org/
18 https://subgraph.com/about-us/index.en.html
"The founders of Subgraph have deep roots in the world of open security research."
19 https://tails.boum.org/blueprint/Tails_research/

collaboration. Open source software and collaboration on projects like Qubes and Tor Project help to keep the software up to date and to help developers fix known bugs and vulnerabilities. Qubes is transparent in addressing its vulnerabilities using *security bulletins*.[20] Tor Project, as something that is constantly being attacked by nation states with large budgets, necessitates frequent updates and quick resolution to vulnerabilities. The executive director of Tor Project, Shari Steele, said: "within the State Department there is offensive and defensive. The same branch of the government can be both trying to defend the network and trying to go out there and attack other people" (Farivar, 2016). This cat and mouse game leads to the need for constant and timely research from both security labs and academia.

The most influential researchers in this field are currently the many developers involved in these projects. Tor developers include: Roger Dingledine, Nick Mathewson, Isis Lovecruft, Philipp Winter, Matt Blaze, and countless others. Qubes main developer and research Joanna Rutkowska, prior to Qubes, developed malware called Blue Pill (Greene, 2010) which functioned as a rootkit that ran an instance of the OS on a thin hypervisor and then virtualized the rest of the operating systems: this effectively meant that anything could be intercepted and, although disputed by others, completely undetected (Messmer, 2007).

The idea of a secure operating system didn't end with Multics. Users nowadays will find that after they build their threat model, an operating system that's "reasonably secure" will suffice. The research doesn't stop and the threats don't stop: e.g. Shellshock showed us that no OS is fully secure (Stevenson, 2014). As new anonymizing protocols are adopted and better ways to virtualize services are developed, we'll be on our way towards more secure operating systems in a networked world; we'll see we need secure operating systems more than ever.

20 https://www.qubes-os.org/security/bulletins/

A note on citation style: I've preferred to use APA, 6th Edition for References and in-text citations. This necessitates repetitive in-text citations because *ibid.* is not used in APA. For quick references to websites and research portals I've used footnotes for ease of a quick click that doesn't require a full reference. I chose not to double-space the annotated bibliography section so as to have fewer pages.

Further Reading:
In an effort to constrain the number of pages in this paper, the author did not properly discuss Subgraph OS and Tails, both great "adversary-resistant" operating systems. Particularly interesting is that Subgraph uses LXC and not Xen. Tails, on the other hand, uses AppArmor. For further readings on these, please consult the following documents:

Subgraph OS Handbook. (n.d.). Retrieved December 1, 2017, from https://subgraph.com/sgos-handbook/sgos_handbook.pdf

Tails: Privacy for Anyone Anywhere. (n.d.). Retrieved December 13, 2017, from https://tails.boum.org/

References

Assessing Your Risks. (2017, November 28). Retrieved December 13, 2017, from https://ssd.eff.org/en/module/assessing-your-risks

Colp, P., Nanavati, M., Zhu, J., Aiello, W., Coker, G., Deegan, T., ... & Warfield, A. (2011, October). Breaking up is hard to do: security and functionality in a commodity hypervisor. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (pp. 189-202). ACM.

Farivar, Cyrus - Jan 10, 2016 2:00 pm UTC. (2016, January 10). Two months after FBI debacle, Tor Project still can't get an answer from CMU. Retrieved December 12, 2017, from https://arstechnica.com/information-technology/2016/01/going-forward-the-tor-project-wants-to-be-less-reliant-on-us-govt-funding/

Greene, T. (2010, April 7). Open source Qubes OS alpha available; Blue Pill malware inventor rolls out secure Qubes OS. Network World. Retrieved from http://link.galegroup.com/apps/doc/A223646024/AONE?u=cuny_queensboro&sid=AONE&xid=7e8b5fa0

Jaeger, T. (2008). *Operating system security*. San Rafael, CA: Morgan & Claypool.

Karger, P., & Schell, R. (2002). Thirty years later: Lessons from the Multics security evaluation. Computer Security Applications Conference, 2002. Proceedings. 18th Annual, 119-126.

Laiserin, Jerry. (2000). The Pre-History of Internet Collaboration.(Technology Information). Cadence, 15(12), 59.

Lesueur, F., Rezmerita, A., Herault, T., Peyronnet, S., & Tixeuil, S. (2010). SAFE-OS: A secure and usable desktop operating system. Risks and Security of Internet and Systems (CRiSIS), 2010 Fifth International Conference on, 1-7.

Messmer, E. (2007, August 2). Questions swirl around virtual-machine rootkit detection; At Black Hat, researcher presents new findings on rootkits, Vista security. *Network World*, 2. Retrieved from http://link.galegroup.com/apps/doc/A167592890/AONE?u=cuny_queensboro&sid=AONE&xid=1fd3d8c9

Overview - Admin: An Introduction to Qubes. (2016, February 1). Open Source FOR You. Retrieved from http://link.galegroup.com/apps/doc/A443074264/ITOF?u=cuny_queensboro&sid=ITOF&xid=06658422

Rutkowska, J. (2012, September 12). The Invisible Things Lab's blog. Retrieved December 12, 2017, from https://theinvisiblethings.blogspot.com/2012/09/how-is-qubes-os-different-from.html

Rutkowska, J. (2014, August). Software Compartmentalization vs. Physical Separation . Retrieved December 5, 2017, from https://invisiblethingslab.com/resources/2014/Software_compartmentalization_vs_physical_separation.pdf

Saltzer, J. (1974). Protection and the control of information sharing in multics. Communications of the ACM, 17(7), 388-402.

Schroeder, & Honeywell Information Systems Inc Mclean VA Federal System Operations. (1975). Security Kernel Evaluation for Multics.

Shrobe, H., & Adams, D. (2012). Suppose We Got a Do-Over: A Revolution for Secure Computing. Security & Privacy, IEEE, 10(6), 36-39.

@Snowden. (2016, September 29). If you're serious about security, **@QubesOS** is the best OS available today. It's what I use, and free. Nobody does VM isolation better. [Tweet]. https://twitter.com/Snowden/status/781493632293605376

Stevenson, A. (2014, October 9). Shellshock proves no operating system is fully secure. v3.co.uk. Retrieved from http://link.galegroup.com/apps/doc/A385107469/ITOF?u=cuny_queensboro&sid=ITOF&xid=1a890964

TorVM. (n.d.). Retrieved December 12, 2017, from https://www.qubes-os.org/doc/torvm/