System Storage as a Service

Michael Orlov

Shamoon College of Engineering, Beer Sheva, Israel
orlovm@noexec.org

Abstract. I propose a secure yet user-friendly computing environment where all local storage is decoupled from the system, allowing for complete separation between computation and storage. This method solves the dilemma faced by security-conscious users, who at present have to choose between using a familiar computing environment and a specific secure operating system. Additional benefits include system access that must be confirmed by third party, uncircumventable global network access policies, security keys introspection, and more.

Keywords: secure environment, hypervisors, security policies

1 Introduction and Project Description

Computer users in hostile environments face a dilemma: either use familiar computing environments such as desktop operating systems, tablets and smartphones, opening themselves to the risks of criminal and state-sponsored malware and traffic inspection, or constrain themselves to specific secure operating systems such as goal-oriented live Linux distributions. [7,9] Moreover, even when using such an OS, the user may be forced, either legally or via less scrupulous means, to provide decryption keys for the persistent local storage. [2]

The purpose of this project is to allow activists, employees, and regular people to use their familiar computing environments while decoupling all local storage, including the OS, from the computing resource. A tiny bootloader, which is the only permanent storage, downloads and verifies a small safe wrapper OS, which executes a virtual machine that works directly with remote encrypted storage that contains the original operating system and data. This process is transparent to the user, whose existing operating environment can be encrypted and uploaded to a remote server using a separate tool. Access requirements for remote storage can be specified separately, such as using two-factor authentication, or requiring a third party confirmation in order to grant access.

This project avoids the limitations of remote desktop or remote storage implementations. No remote architecture-specific operating systems and resources are needed, and there are no dangers of leaks via local side-channels such as swap. Risk of local exploits and Trojans is also partially mitigated. The bandwidth bottleneck may be reduced via storage compression and local temporary caching. Optional extensions include the ability of several users to take advantage of the same basic storage via personalized snapshots, automatic backups, and other features.

2 Background and Related Work

There are many solutions that try to make computing environments more secure when various computer security aspects are considered. These solutions span, for example, hard drive encryption, anti-viruses, virtual private networks, remote storage, thin clients, and many others. What is common to most of the above is that they work from inside a locally installed general-purpose operating system, which is itself vulnerable to attacks.

Attempts to circumvent this fundamental vulnerability typically come in the form of special-purpose OS distributions, such as TENS [9] (aimed at military personnel and federal employees) and Tails [7] (aimed at privacy-conscious users). There are even hardware solutions incorporating a similar concept. [4] I developed such a system myself [5], and came to a conclusion that regular users will not employ a special-purpose OS on regular basis due to a sharp conflict between security and steep learning curve of using such a system.

However, during development of Liberté, which was the first Linux distribution to ship with a Secure Boot-based trusted boot chain, it became apparent that once several hardware-related capabilities become commonplace, it will become possible to decouple the special-purpose secure OS from the general-purpose user's OS without necessarily sacrificing usability or security. These capabilities, which are commonly available now, are: non-desktop virtualization support, UEFI firmware with reliable Secure Boot support [8], and ubiquitous fast and reliable network connectivity.

3 Decoupling All System Storage

Below, I describe the concept of *System Storage as a Service* (SSaaS), where all storage used by the system, including its operating system, is remote and secure. Here are some evident advantages of the approach:

- Trusted boot chain prevents tampering with the system. Injecting data into local storage provides no benefit to the attacker, as local storage is at most an encrypted cache of primary remote storage. Remote encrypted storage is also protected.
- Network access policies, such as VPN usage, can be defined outside user's operating system. Granularity of such policies is not necessarily limited by what can be inferred from connection metadata, since methods can be devised for real-time introspection of encryption keys inside libraries used by user's OS.
- Hardware access policies can be similarly enforced, which is essential for preventing DMA and other side-channel attacks. [6]
- Ability to enter user's computing environment may be restricted to depend on third-party authorization. This factor has recently become crucial for privacy-conscious users and for employees working with sensitive data, in light of border control policies requiring users to provide access to their devices when they are capable to do so. [2]

- Seamless transition to SSaaS from a regular local setup is possible.
- Backups can be easily managed centrally, and multiple users can work off the same primary disk image using personalized snapshots.
- Virtualization benefits that are now prevalent in the industry, such as easily moving the computation environment to different hardware, become possible at the full system level for the end-user. In principle, it should be possible to load a familiar secure environment on any similar hardware, with the OS and the data immediately accessible.

The apparent disadvantages are:

- Hardware support by the SSaaS OS may be lacking. This limitation can be mitigated by focused testing of approved hardware configurations that are requested by clients.
- Mobile devices often lack virtualization capabilities. However, the situation is improving in light of recently added virtualization support in ARM and Wind River-based systems. [1]
- Network access may be intermittent. This issue can be alleviated by employing local storage as transient encrypted cache with write-back policy that is subject to network conditions quality. Otherwise, as more and more applications rely on Internet access to work, SSaaS requirements should not stand out as peculiar.

4 Architecture and Proof of Concept

The only component of SSaaS that needs to be installed on user's computer is the tiny UEFI bootloader, which does not require frequent updates. Verified by UEFI Secure Boot firmware, the bootloader fetches the SSaaS hypervisor OS from remote storage. Based on experience with [5], the download size should not exceed $\approx 200\,\mathrm{MiB}$.

The SSaaS OS subsequently initializes the hardware, sets up peripheral devices and network access policies, and configures a module for accessing the remote storage, optionally through encrypted local cache. Afterwards, user's OS is started in a virtual machine, subject to the policies above.

Any attempt to tamper with local or remote storage will violate the trusted boot chain. In case when there is a danger that the user may be coerced into providing access to the system, such as at an international border crossing [2], access to remote storage can be temporarily and externally restricted.

Figure 1 show a diagram of the architecture. I am currently developing a prototype of securely porting existing Windows and Linux operating systems on common x86-64 architectures to completely remote encrypted storage. Hopefully, extending the project to Apple operating systems and to non-Intel, mobile architectures will be possible as well, but this ultimately depends on project's timeframe and funding.

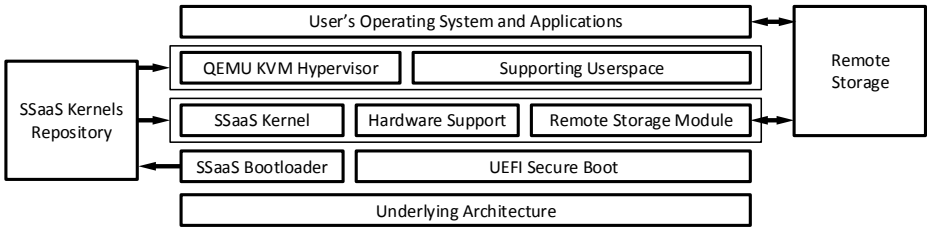

Fig. 1. Representation of SSaaS architecture. Verified bootloader fetches the kernel and the userspace from a remote repository. Once the remote storage module connects to user's data, the guest operating system can be loaded in a hypervisor.

5 Market and Viability

The project as it is described here, once prototyped, should generate interest in privacy-conscious communities due to its novel security focus. I personally experienced this effect with English and Russian-speaking communities when working on [5]. A significant part of this interest comes from developers who are eager to work on open-source security software, which does not only cover the code of the project itself, but also, for instance, related distribution and operating system bugs. Additionally, security projects that do not publish source code attract significant criticism that hurts their feasibility. [3]

Therefore, the most appropriate model for SSaaS development should include open-sourcing most of the code. There are multiple viable business models that rely on open-source code.[1] The following options seem as the most attractive:

- Developing custom solutions per client, similar to what is done internally by TENS [9];
- Selling ready solutions that include managed remote storage;
- Selling enterprise-level support services.

6 Conclusion

I summarized System Storage as a Service (SSaaS) — a concept of abstracting away *all* system storage for the purpose of cybersecurity. In doing so, I drew on my experience with developing Liberté Linux, which is a custom source-based secure Linux distribution with many innovative features that was downloaded and used by thousands of users worldwide.[2] I am convinced that there is real demand for SSaaS, as the approach provides a significant increase in user-level security and privacy. Whether this approach will be taken advantage of at OEM level by market players, or it will be used only as a separate feature by selected users and entities — remains to be seen.

[1] See `https://en.wikipedia.org/wiki/Business_models_for_open-source_software`.

[2] See the following press-conference (in Russian): `https://lenta.ru/conf/kammerer`.

References

1. Dall, C., Nieh, J.: KVM/ARM: The design and implementation of the Linux ARM hypervisor. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 333–348. ASPLOS '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2541940.2541946
2. Electronic Frontier Foundation: Digital privacy at the U.S. border: Protecting the data on your devices and in the cloud (2017), https://www.eff.org/wp/digital-privacy-us-border-2017, [Online; accessed 26-March-2017]
3. Forbes: Pressure increases on Silent Circle to release application source code (2013), https://www.forbes.com/sites/jonmatonis/2013/02/06/pressure-increases-on-silent-circle-to-release-application-source-code, [Online; accessed 26-March-2017]
4. IronKey, Imation, Kingston Digital: IronKey (2005–2012), http://www.ironkey.com, [Online; accessed 26-March-2017]
5. Orlov, M.: Liberté Linux (2010–2013), http://dee.su/liberte, [Online]
6. Stewin, P., Bystrov, I.: Understanding dma malware. In: Flegel, U., Markatos, E., Robertson, W. (eds.) Detection of Intrusions and Malware, and Vulnerability Assessment: 9th International Conference, DIMVA 2012, Heraklion, Crete, Greece, July 26-27, 2012, Revised Selected Papers, pp. 21–41. Springer, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-37300-8_2
7. The Tor Project, Inc: The Amnesic Incognito Live System (Tails) (2009–2017), https://tails.boum.org, [Online; accessed 26-March-2017]
8. Unified EFI Forum: Unified extensible firmware interface specification, Version 2.6, Errata A (2017), http://www.uefi.org/specifications
9. US DOD Software Protection Initiative: Trusted End Node Security (TENS) (2007–2016), https://spi.dod.mil/lipose.htm, [Online; accessed 26-March-2017]