AppVault: A Flexible and Open Security Companion Device

Skyelar Craver, Steven Pitts, and Collin Fraioli
Electrical and Computer Engineering Department
Wentworth Institute of Technology
Engineering Senior Design 2019

***Abstract*— Security is a growing field as more sensitive data is stored digitally, and more users' daily lives are revolving around computers. With new vulnerabilities and leaks being discovered constantly, there grows a population in need of verifiable and flexible secure computing.**

This project set out to specify and construct a working protype of a device that could keep the most vulnerable targets, holding the most valuable information or working in untrustworthy environments safe. By taking lessons from previous exploits, maintaining open sources, and paring down the attack surface, AppVault could deliver just such a portable secure environment.

I. Introduction

THIS project was influenced by the growing risk associated with many modern digital systems. While it is easier than ever to create highly secure applications with modern software engineering tools, the number of new vulnerabilities discovered over the last few years is record breaking.

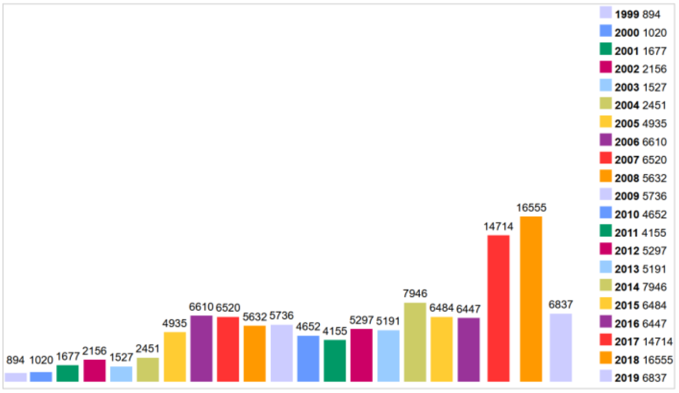

Figure 1: Published CVEs by year [1]

Just as security focused tooling has matured, so have software fuzzing and reverse engineering toolkits. The harm of these attacks' ranges from obscure theoretical threats only to be done in labs, to complete remote execution with a few lines of script. Perhaps the most significant vulnerabilities discovered in recent years have been the hardware level vulnerabilities that exploit the speculative execution and predictive pipelines modern processor manufacturers have used to speed up their architectures. [2]

While a recent push towards open source development has led to many of these vulnerabilities being discovered, obscuring source to hide vulnerabilities could not be more secure. The hardware level vulnerabilities could have been discovered more quickly had the full designs of affected x86 and ARM platforms were available to security auditors and researchers.

II. Theory

The designs theorized to meet the requirements set went through many iterations informed by security literature research. Proposals included: an emulated secure environment that would use address space padding and homomorphic encryption to hide execution from the operating system, a hardened hypervisor that would isolate on a process level and keep all RAM for its running virtual machines encrypted, an embedded co-processor in the vein of ARM's secure enclave, and finally the chosen implementation of a standalone USB connected device capable of arbitrary execution of selected sensitive processes.

This implementation was then further elaborated upon to ensure maximum security at a design level. Embedding encryption keys on the device in the vein of a hardware security module and following design guidelines laid out for HSMs as outlined by bodies such as NIST would allow for long and secure AES keys to be kept away from even a malicious host machine. Another key decision was to forego a traditional operating system. Although highly secure, the sheer size of the Linux kernel all but guarantees a risk of exploitation. Instead a simple firmware layer that would expose most POSIX interfaces would be created, using other vetted code as a basis wherever possible. Finally, one of the largest sources of risk in running applications stems from the plurality of applications running. The odds of one

This article was supervised by Professor Aaron Carpenter for ENGR 5000 and 5550 (Engineering Senior Design) in the Electrical and Computer Engineering Department at Wentworth Institute of Technology.

application's data being intercepted will increase proportionately to the number of processes running at a given time, as each piece of software could be introducing its own vulnerabilities to be exploited. Thus the AppVault would only run a single application at a time, and hold no persistent information in between disparate programs running.

Without an operating system, and without any persistence, there would arise a new difficulty in exposing interfaces for application developers. Some applications can have complex dependency graphs and expect things to be in specific locations. This has been an issue in application development for a long time, and many solutions have arisen in the last few years to accommodate this. The proposed solution would take the application, all its dependencies, and bundle them into a file system that is then archived and encrypted by the AppVault's internal key. This solution is like the relatively popular AppImage format for Linux. The critical differences, however, are the encryption, and the bundling of state and user data into this format. The result is something reminiscent of using a cartridge on an old game console, with all files and save data in one pluggable object.

III. Application

While the full theoretical implementation outlined could not be actualized within the narrow timeframe of the project, a software and later hardware prototype was developed that implemented many of the basic principles of the design.

The software implementation was made in Python to run on a Raspberry Pi Model 3B. This demonstration would clearly lack the open hardware requirement of the project, the small footprint necessary to limit the attack surface, and settle for a simulation of the desired protocols. This higher-level demo did allow for faster iteration using the high-level language and standard Linux environment of the Pi. This environment also allowed for certain concepts that were too complex to write and run well on the microcontroller board used for the hardware implementation, such as using file system images for program dependency and state bundling.

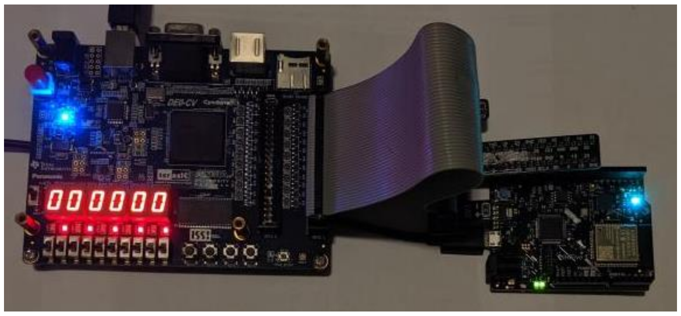

Figure 2: Current hardware prototype of AppVault.
Left: Altera DE0-CV development board
Right: SiFive Hifive-Rev B development board

The hardware accurate implementation utilized consumer development boards to iterate on connectivity and allow for dynamic expansion should requirements change. Although a custom board implementation was worked on, the changing specifications on pinouts and controllers delayed the manufacturing of the custom version infeasible for the time frame.

On the version shown, the FPGA was loaded with a powerful AES core, with an embedded, pseudo-random, erasable 256-bit key. This isolated the key used for encryption and decryption of the program even from the RISC-V core running the user program, granting an additional layer of isolation. The software running on the SiFive board is a minimal firmware used to load in machine binaries over a UART port, decrypt over the SPI connected AES core, execute the program, and re-encrypt the binary with any user data changes to send back over the UART connection. This implementation lacks many of the standard C libraries, or ability to run higher level interpreters, as well as any support for the proposed file system package format. The unit was able to successfully load in, encrypt, decrypt, and execute a simple test program effectively.

IV. Conclusion

The two demos created for this project show the feasibility of creating a true secure system, capable of protecting the likes of even the most sensitive populations. The largest limiting factors in creating the true ideal system, are the maturity of the RISC-V platform, which only hit v1.0 for its base instruction set in 2017, and the time allotted for us to complete the project.

Further refinements in the firmware on the board would allow more user functionality, identifying an open source FPGA capable of holding the AES core, or a RISC-V core with proposed security extension support, would allow significant shrinkage of the system, and speed up general execution. Moving to a higher speed interface between the host and the AppVault would also be advantageous, but would require more advanced USB controller integration.

References

[1] Cvedetails.com. (2019). *Browse cve vulnerabilities by date*. [online] Available at: https://www.cvedetails.com/browse-by-date.php [Accessed 22 Jul. 2019].

[2] N. Abu-Ghazaleh, D. Ponomarev and D. Evtyushkin, "How the spectre and meltdown hacks really worked", *IEEE Spectrum*, vol. 56, no. 3, pp. 42-49, 2019. Available: 10.1109/mspec.2019.8651934.