HyperPass: Secure Password Input Platform

Michael Kiperberg[1] and Nezer J. Zaidenberg[2]
[1]*Software Engineering Department, Shamoon College of Engineering, Beer-Sheva, Israel*
[2]*School of Computer Science, The College of Management, Academic Studies, Israel*

Keywords: Virtual Machine Monitors, Hypervisors, Trusted Computing Base, Security Indicators.

Abstract: Confidential information, like passwords and credit card numbers, travel from the user's local machine to a remote server via secure communication protocols. Whereas remote servers are serviced by security specialists, local machines are more vulnerable to memory attacks and keyloggers. To counter these attacks we propose a secure password input platform, called HyperPass, which is based on a thin hypervisor. The thin hypervisor acts as an isolated and secure environment for entering and encrypting user's confidential information. HyperPass uses the keyboard's scroll lock LED as a security indicator. Our evaluation shows that the performance overhead of HyperPass is insignificant ($\approx 2.79\%$).

1 INTRODUCTION

Confidential information, like passwords and credit card numbers, travel from the user's local machine to a remote server via secure communication protocols, like HTTPS (Rescorla et al., 2000). Remote servers are serviced by security specialists and have many protection layers, e.g., firewalls (Sharma et al., 2014), antiviruses (Gandotra et al., 2014), intrusion detection systems (Axelsson, 2000). On the other hand, the user's local machine, which usually lacks some of these protection mechanisms, is more vulnerable to attacks.

The confidential information is at risk between the moment of pressing a keyboard key until the moment of encryption by a secure communication protocol. Keyloggers (Hussain et al., 2016) can intercept the confidential information while it is being typed by the user. Software attacks can fetch the confidential information from the memory of the target process, e.g., a web browser. In order to protect the confidential information, it is required to: (a) safely acquire user's input, (b) store the input in an isolated environment, and (c) encrypt the input before it leaves the isolated environment.

Requirements (a) and (b) can be easily satisfied on ARM processors using ARM's TrustZone (Pinto and Santos, 2019). A software module executing in TrustZone's isolated environment, the "secure world", can prevent "normal world" software from accessing to its internal data.

In addition, ARM's SoC devices are divided into two categories: secure and non-secure (Rosenbaum et al., 2019). A special bit, the "NS bit", on the system bus is used to tag the security of each transaction. Transaction issued by secure devices set the NS bit to 0, whereas non-secure devices set this bit to 1. Secure devices will not handle transactions whose NS bit is set to 1. The CPU issues a transaction the value of the NS bit is determined by the current *world* of the CPU: a normal world sets the bit to 1 and a secure world sets it to 0. For example, a security LED, being a secure device, can be turned off and on only by a software module executing in the secure world.

On Intel CPUs, comparable memory isolation can be achieved using virtualization (Neiger et al., 2006) and enclaves (SGX) (Costan and Devadas, 2016). However, Intel platforms are lacking built-in mechanisms for secure communication with external devices. Fortunately, virtualization can be used to overcome this deficiency. BitVisor (Shinagawa et al., 2009) was the first attempt to use virtualization for securing a hardware device. Specifically, BitVisor is a thin hypervisor that intercepts the communication between the operating system and a hard disk in order to implement full disk encryption transparently.

In order to secure user's input, we propose to use a thin hypervisor, similarly to BitVisor. Unlike BitVisor, our hypervisor intercepts the communication between the operating system and the keyboard. When the user is required to enter a password, the hypervisor turns on the scroll lock LED (Chapweske, 2003),

Kiperberg, M. and Zaidenberg, N.
HyperPass: Secure Password Input Platform.
DOI: 10.5220/0010301905800587
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 580-587
ISBN: 978-989-758-491-6

indicating that the input of the user is safe. All attempts to turn on the scroll lock LED from the operating system are circumvented by the hypervisor. The scroll lock LED can be seen by the user as a security indicator.

There are several ways to satisfy requirement (c): encrypting the confidential information before it leaves the isolated environment. The most common approach is to move the communication stack from the normal environment to the isolated (secure) environment (Liu and Cox, 2014; Rubinov et al., 2016), thus significantly increasing the size of the trusted code and decreasing the overall security. Another approach attempts to move only the encrypting functionality to the isolated environment (Bugiel et al., 2010; Li et al., 2014). This approach usually requires cooperating between the code in normal and secure environments (Ying et al., 2018).

Following the idea of T-PIM (Hirano et al., 2009), we propose to use an external proxy-server connected to the isolated environment using a second communication line. Specifically, the isolated environment encrypts confidential information using the proxy's public key and sends it via the second communication line. The proxy decrypts the confidential information and inserts it in the communication flow. The user "marks" the place of insertion by entering a predefined code-word.

From the user's perspective the process of password entering is very natural. The user opens his browser and navigates to the login screen of some portal, which asks for a username and password. In response the user enters his username and a predefined code-word. A window pops up on the screen asking for the actual password. The scroll lock LED is turned on, signifying that it is now safe to enter the password. The user enters his password and presses enter. The password is encrypted and sent to the proxy server, which replaces the code-word with the actual password and sends the resulting form to the remote web server. The login succeeds.

The main contributions of this paper are as follows. We present a novel method for secure input from a conventional keyboard using the scroll lock LED as a security indicator. We describe a hypervisor-based system, called HyperPass, which takes advantage of this input method. HyperPass allows the user to enter his password and other confidential information naturally and securely. Unlike previously described approaches, HyperPass does not require any modifications to existing software nor does it require execution of additional VMs on the user's machine. We present and evaluate a prototype implementation of HyperPass. Our results show that the performance overhead of HyperPass is insignificant ($\approx 2.79\%$).

2 BACKGROUND

Intel introduced virtualization extensions to their CPUs more than a decade ago (Neiger et al., 2006). The main reason for introducing these extensions was enabling the execution of multiple operating systems simultaneously in an isolated environment, called a "Virtual Machine" (VM). A special software component, called a "Virtual Machine Monitor" (VMM) or "hypervisor", manages the virtual machines. The hypervisor can configure interception of various events that occur in the VMs, e.g., execution of privileged instructions, access to IO ports, triggering of an exception, access to Model-Specific Registers (MSRs), etc. Upon the occurrence of a pre-configured event, a "VM-exit" occurs, which transfers the control to the hypervisor. The information about the occurred event is stored in a special data structure for the hypervisor's inspection.

The hypervisor inspects the information about the occurred event and handles it accordingly. After completion of the event handling, the hypervisor performs a "VM-entry", which transfers the control to the VM.

The hypervisor manages the memory via a mechanism called "Extended Page Tables" (EPT). ETP acts as a secondary page table, which translates VM's physical addresses to real physical addresses. In addition, the hypervisor can use the EPT to make certain pages inaccessible by the VM. An attempt to access these pages trigger and EPT-violation, transferring the control to the hypervisor.

A hypervisor can intercept a wide range of events. While interception of some events, e.g., access to IO ports, can be disabled, others, e.g., execution of the `CPUID` instruction, are intercepted unconditionally.

HyperPass uses three types of events: (1) access to IO ports, (2) execution of the `CPUID` instruction, and (3) EPT-violations. The hypervisor can select the ports that need to be intercepted. Attempts to access the selected ports will trigger a VM-exit, whereas accessing other ports will not incur any overhead.

The `CPUID` instruction is used by many hypervisors, e.g., Xen, KVM, Hyper-V, as a hypercall instruction (Microsoft, 2012), i.e. an instruction that allows a software module executing inside a VM to invoke a function of a VMM. The `eax` register is used to select the required function, whereas other registers act as arguments.

An EPT-violation occurs when the virtual machine attempts to access a page that was marked as

inaccessible by the hypervisor. In response to such access, the control is transferred to the hypervisor. The hypervisor can respond by modifying the access rights and performing a VM-entry.

2.1 PS/2 Keyboards

The current implementation of HyperPass supports only PS/2 keyboards. This type of keyboards was selected for the simplicity of its communication protocol. The communication with the PS/2 keyboards is managed by a PS/2 controller (Chapweske, 2003). The PS/2 controller uses two IO ports for its operation, 0x60 as a data port and 0x64 as a command and status port. Port 0x64 is used for communication with the controller itself, e.g., reading and writing its internal RAM, and for selection of the destination device. Port 0x60 is used to send and receive data from the previously selected destination device.

Assuming that the selected destination device is a PS/2 keyboard, the act of reading from port 0x60 reads the last scan code of the keyboard. A scan code represents a keyboard event, like pressing or releasing a key. The scan codes can then be translated into a character according to a conversion table. Some scan codes, e.g., presses and releases of shift keys, affect the translation of other scan codes. For example, in order to type a capital 'A', one can produce the following sequence of events: (1) pressing the left shift key, (2) pressing the 'a' key, (3) releasing the 'a' key, (4) releasing the left shift key. The presses left shift key causes the pressed 'a' key to be translated into the capital 'A' character.

The act of writing to port 0x60, sends a command to the keyboard. The commands configure various aspects of a keyboard's behavior. Some commands are single byte, others are followed by an argument. For example, the 0xFF command is a single byte command, which resets the keyboard. The 0xF3 command defines the rate at which a pressed key produces events. The rate is defined by the byte that follows the 0xF3 byte. Another example is the 0xED command, which defines the LEDs state. The lower three bits in the byte that follows the 0xED command define the state of scroll lock (bit 0), number lock (bit 1) and the caps lock (bit 2) LEDs.

3 THREAT MODEL

We assume that the attacker has full control over the software of the local machine. In particular, we assume that the attacker is able to execute random code in kernel-mode. We impose only a single limitation on the attacker: we assume that the EFI boot order cannot be changed, i.e. we assume that HyperPass' EFI application is always the first application that is loaded by the EFI firmware.

4 SYSTEM DESIGN

HyperPass consists of three components: (a) a thin hypervisor, (b) a user-mode agent application and (c) a proxy server. The hypervisor and the agent run on the user's local machine, whereas the proxy server runs on a remote server. Several users can share a single proxy server.

Figure 1 depicts the interactions between the different components of HyperPass. HyperPass requires a proxy server between the local machine and a remote server. To understand these interactions, consider the following example, in which a user attempts to log in to the web portal of his organization. The remote server redirects the user to a login screen (steps ① and ②), which requests a username and a password. Normally, the user enters his credentials and they are sent to the remote server. With HyperPass, the user enters a special code-word in the password field, thus indicating his desire to enter the password securely. The username and the code-word, as any other outgoing communication, are sent to a preconfigured proxy server (step ③). The proxy server detects the code-word and sends a password request to the user-mode agent (step ④). The user-mode agent displays the URL of the web portal and the name of the password field. Then, the agent issues a hypercall (step ⑤), which causes the hypervisor to turn on the scroll-lock LED and begin intercepting keyboard strokes. The user verifies that the scroll-lock LED is on and enters the password. The password is consumed by the hypervisor one letter at a time until the enter key is pressed, which signifies the end of the password. The hypervisor encrypts the password using the proxy's public key and transmits it to the user-mode agent (step ⑥), which sends it to the proxy (⑦). The proxy replaces the code-word with the decrypted password and sends the resulting HTTP request to the remote server (step ⑧).

4.1 Proxy Server

HyperPass uses *mitmproxy*, an extensible HTTP proxy server (Boneh et al., 2007). *mitmproxy* provides a Python API for add-ons, which can perform different monitoring activities on the communication flows. When a browser is configured to use a proxy server, the communication is encrypted using the proxy's

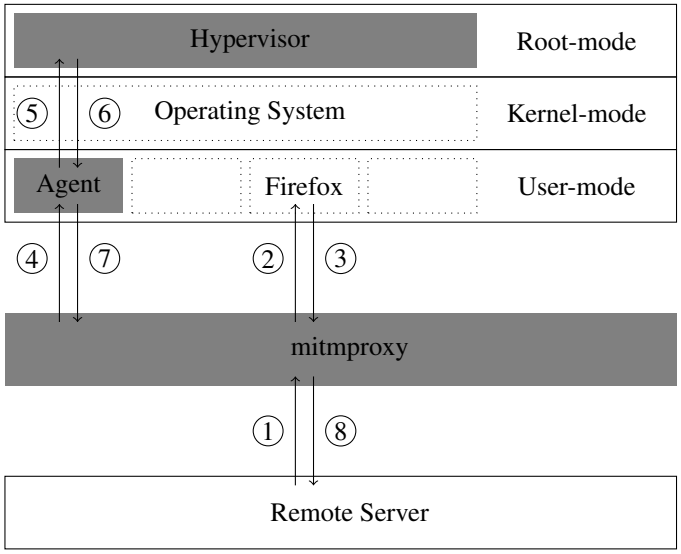

Figure 1: Interactions between HyperPass components and other software. The three components of HyperPass, the hypervisor, the user-mode agent and the proxy, are shaded.

public key, thus allowing it to decrypt and analyze the data.

HyperPass includes an add-on for *mitmproxy* that intercepts HTTP requests and analyzes them. The goal of the analysis is to find password fields that include a predefined code-word. When such a password field is found, the add-on contacts the user-mode agent, running on the local machine. The add-on provides the agent with the name of the password field and the URL to which it was submitted.

The user-mode agent replies with the user-provided password. The password is encrypted with the add-on's public key (which may differ from *mitmproxy*'s public key). The add-on decrypts the password and replaces the code-word with the decrypted password. Then, the HTTP request is sent to its destination.

4.2 Agent

The user-mode agent is implemented as a console application, which listens on a UDP socket. When the proxy server connects to the UDP socket, it sends a pair consisting of the name of the password field and the URL to which this password field should be sent. The agent displays these the name of the field and the URL to the user and issues a hypercall.

The hypercall requests the hypervisor to enter secure mode. While in this mode, the hypervisor reads the keyboard strokes and replaces them with the asterisk key, which is then displayed by the user-mode agent.

After entering the password, the user presses the enter key, causing the agent to issue another hypercall. The hypervisor reacts to this hypercall by provided the agent with an encrypted version of the user password. The encryption is performed using the proxy's public key. The agent sends the encrypted password to the proxy server and continues listening on the UDP socket.

4.3 Hypervisor

The hypervisor is embedded in an EFI application (Zimmer, 2009). The application boots before the operating system, initializes the hypervisor and invokes the operating system bootloader.

The hypervisor intercepts two types of events: (a) input-output on port 0x60 and (b) execution of the `CPUID` instruction. The `CPUID` instruction is used for hypercalls. The user-mode agent sets the `eax` register to the hypercall index and executes the `CPUID` instruction. The instruction is intercepted by the hypervisor, which examines the value of the `eax` registers and acts accordingly. Other registers can be used to pass parameters.

In the current implementation, the hypervisor implements two hypercalls: `HC_BEGIN` and `HC_END` and intercepts input and output on port 0x60, the PS/2 data port. Algorithm 1 presents the pseudo-code of the hypervisor's VM-exit handler.

Any attempt to set the scroll lock LED is prevented by the hypervisor (lines 10–18). In PS/2 keyboards, in order to change the state of the LEDs, the operating system transfers the 0xED command byte,

followed by a LED configuration byte. The hypervisor intercepts the 0xED byte (lines 12–13) and clears the scroll lock bit in the following configuration byte (lines 14–16). In any case, the hypervisor must emulate the intercepted operation (line 18).

The password capturing process begins with the `HC_BEGIN` hypercall, continues with a series of inputs on port 0x60, and terminated with the `HC_END` hypercall. In response to `HC_BEGIN` (lines 1–4), the hypervisor turns on the scroll lock LED, clears the password and sets the `SecureMode` variable.

Any attempt to read a key from the keyboard is intercepted and analyzed by the hypervisor (lines 19–35). The hypervisor reads the input byte from the keyboard (line 20). If the byte corresponds to a press or a release of the shift key, the `UpperCase` variable is updated accordingly (lines 21–24). If the byte corresponds to a release of the backspace key, the last character of the password is removed (lines 25–26). If the byte corresponds to a release of a letter key (line 27), the byte is converted to a character (line 28) and its case is fixed, if necessary (lines 29-30). Then the hypervisor concatenates the character to the password (line 32) and sets the byte that was read from the keyboard to a scan code of an asterisk (line 33). Finally, the hypervisor emulates the intercepted operation (line 35).

In response to `HC_END` (lines 5–9), the hypervisor turns off the scroll lock LED, clears the `SecureMode` variable (lines 6–7). Then, the hypervisor encrypts the password using the proxy's public key and copies the encrypted password to the user-mode agent.

In the current implementation the public key of the proxy server is hardcoded in the hypervisor.

Algorithm 1: Hypervisor's VM-exit Handler.

```
 1: if CPUID and eax = HC_BEGIN then
 2:     TurnOn(ScrollLockLED)
 3:     Password := ""
 4:     SecureMode := True
 5: else if CPUID and eax = HC_END then
 6:     TurnOff(ScrollLockLED)
 7:     SecureMode := False
 8:     Cipher := Encrypt(Password)
 9:     MemCpy(ebx, Cipher)
10: else if output on port 0x60 then
11:     o := output byte to port 0x60
12:     if o is LED control then
13:         LedControl := True
14:     else if LedControl then
15:         LedControl := False
16:         Turn off scroll lock bit in o
17:     end if
18:     EmulateOutput(0x60, o)
19: else if input on port 0x60 then
20:     i := input byte from port 0x60
21:     if i is shift press then
22:         UpperCase := True
23:     else if i is shift release then
24:         UpperCase := False
25:     else if i is backspace release then
26:         Password := Password[1,|Password|-1]
27:     else if i is letter release then
28:         j = ToLetter(i)
29:         if UpperCase then
30:             j := ToUpper(j)
31:         end if
32:         Password := Password + j
33:         i := ToScanCode('*')
34:     end if
35:     EmulateInput(0x60, i)
36: end if
```

5 EVALUATION

5.1 Code Size

An important measure of system security is its code size. HyperPass is based on a thin hypervisor with about 2 KSLOCs equipped with an RSA implementation of 0.3 KSLOCs. In order to show how small HyperPass is and how large can be a full hypervisor equipped with HTTPS (Rescorla et al., 2000) support, we present the sizes of some open-source projects. The lwIP library, which implements the TCP/IP stack, has 146 KSLOCs. The mbedTLS library, which implements secure communication protocols, is 127 KSLOCs. The XEN hypervisor has 600 KSLOCs. The sizes of the projects were obtained using the GitHub Gloc extensions (Solovev, 2020) for the Google Chrome browser.

5.2 Security Evaluation

We assess the security of the proposed system from two perspectives:

- the inability of an attacker to turn on the scroll lock LED and
- the inability of an attacker to obtain the password in its decrypted form.

The keyboard LEDs are controlled by the keyboard and can be changed only by sending an appropriate command on port 0x60. The command consists of two bytes: the command byte 0xED followed by a flag byte that specifies the setting of each of the three LEDs. The hypervisor intercepts the outgoing communication on port 0x60. When the 0xED byte

Table 1: Testing environment configuration.

Host CPU	Intel(R) Core(TM) i7-10610U
Host memory	16 GB
Host OS	Ubuntu 20.04.1 LTS
VMM	VMware Workstation 15.5.6
Guest CPU	Intel(R) Core(TM) i7-10610U
Guest memory	8 GB
Guest OS	Windows 10 (19041)

is detected, the hypervisor clears the scroll lock bit (bit 0) of the following byte. Therefore, even an attacker with kernel-mode random code execution abilities cannot turn on the scroll lock LED.

Similarly, the hypervisor intercepts all the input communication on port 0x60 and masks the scan codes with the scan code of an asterisk when the scroll lock LED is on. The original scan codes are translated to characters and are stored in the hypervisor's memory region, which is protected by EPT. Specifically, the hypervisor makes its memory regions inaccessible by the operating system. Any attempt to access these regions induces an EPT-violation. The hypervisor responds to this event by clearing the password and resetting the machine.

5.3 Performance Evaluation

HyperPass uses a thin hypervisor whose existence may degrade the overall system performance. In order to assess this performance degradation, we ran a benchmarking tool called PCMark (Sibai, 2008) in three configurations: (a) without a hypervisor, (b) with HyperPass' hypervisor, (c) with Oracle VirtualBox (6.1.14) as an example of a full hypervisor, running the same version of Windows with 8GB of RAM. The exact configuration of our testing environment is summarized in Table 1. Table 2 presents the scores that were given by PCMark in each configuration for every category and Figure 2 presents the overhead in percent.

A thin hypervisor and Hyperpass' hypervisor incur a minor performance degradation ($\approx 1.35\%$ and $\approx 2.79\%$ on average, even without taking into account the Video Conferencing results). The only difference between the thin hypervisor and HyperPass' hypervisor is the interception of input and output on port 0x60. Because this port is accessed by the operating system only upon an interrupt from the keyboard, which is a relatively rare event, our tests show a minor performance degradation due to this interception. A full hypervisor, represented by Oracle VirtualBox, showed a much higher impact on the overall system performance ($\approx 38.6\%$ on average).

6 RELATED WORK

The idea to use an isolated environment for securing passwords and other sensitive information is not new. ARM TrustZone (Pinto and Santos, 2019) is an example of such an isolated environment, that is widely used for security applications in the industry and academic research. Samsung pay (Samsung, 2020) and Alipay (Alipay, 2020) use TrustZone to secure users' credit card numbers. Bitcoin Ledger (BitCoin, 2020) uses TrustZone for transaction confirmation. While in TrustZone a security LED is turned on thus indicating to the users that their input is safe.

After receiving the sensitive information, the security module running in an isolated environment, needs to send this information to a remote server, without revealing this information to the normal environment. Typically, this problem is solved by moving some of the communication logic to the isolated environment (Liu and Cox, 2014; Rubinov et al., 2016). For example, VeriUI (Liu and Cox, 2014), a login page protection system, moves the entire communication stack and UI framework to the isolated environment.

Another approach, which is adopted by TruWalletM (Bugiel et al., 2010), Droidvault (Li et al., 2014), KNOX (Knox, 2013), TruZ-Droid (Ying et al., 2018) and HyperPass, attempts to lower the size of the code executing in the isolated environment. This is achieved by keeping the communication stack and the UI framework in the normal environment but encrypting the sensitive information before it leaves the isolated environment.

Unfortunately, TrustZone with its exclusive access to certain hardware components, e.g., security LEDs, is not available on x86 processors. However, the Intel VT-x (Neiger et al., 2006) technology can be used to construct an isolated environment in a form of a hypervisor. Commonly, an existing hypervisor, e.g., Xen or KVM (Deshane et al., 2008), is extended to provide additional security. The security applications range from system call tracing (Pfoh et al., 2011) and DRM (Kiperberg et al., 2019) to preventing unauthorized code execution (Seshadri et al., 2007; Leon et al., 2019).

BitVisor (Shinagawa et al., 2009) is a thin hypervisor that intercepts accesses to ATA hard disks and enforces storage encryption. HyperPass uses a similar method of IO interception for implementing a security indicator.

T-PIM (Hirano et al., 2009) is most closely related to HyperPass. T-PIM extends Xen to provide secure password input. It uses two virtual machines: a working VM and a trusted VM. The browser runs in the

Table 2: PCMark scores in four configurations.

Category	No Hypervisor	Thin Hypervisor	HyperPass' Hypervisor	VirtualBox
App start-up	7277	7110	6842	4008
Video conferencing	2129	2307	2301	1197
Web browsing	2067	2012	2007	1390
Spreadsheet	4473	4420	4354	2688
Writing	5005	4928	4871	2870
Photo editing	850	852	847	524
Video editing	903	887	856	650

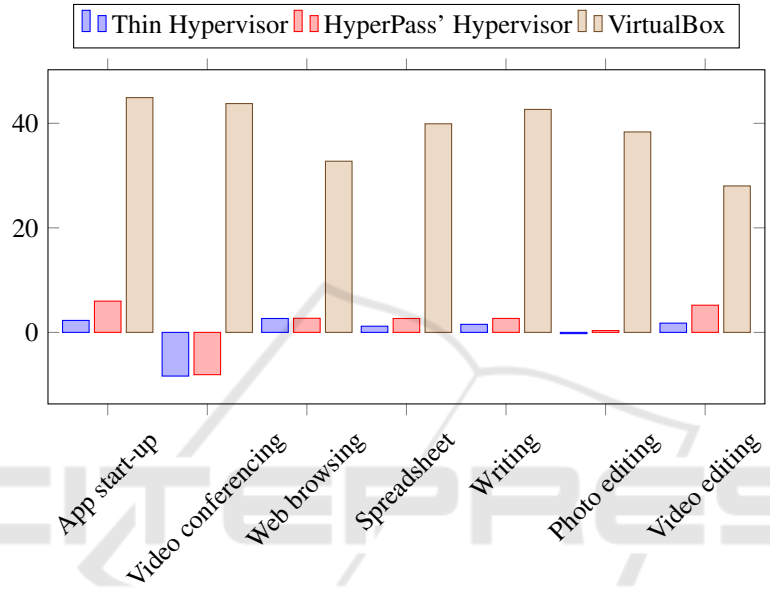

Figure 2: Performance degradation in percent in three configurations with respect to the "No Hypervisor" configuration.

working VM and when a password needs to be entered it is entered in the trusted VM. From the user's perspective, the trusted VM acts as a security indicator, when the trusted VM is active, the user's input is safe. Similarly to HyperPass, T-PIM uses *mitmproxy* to intercept and manipulate the communication between the working VM and an external server.

HyperPass differs from T-PIM in two aspects.

- The size of the code executing in an isolated environment is much smaller in HyperPass than in T-PIM. Whereas in HyperPass only the encryption module is part of the isolated environment, in T-PIM the isolated environment includes the entire Linux operating system. As a result, HyperPass is much more secure than T-PIM.
- Unlike T-PIM, HyperPass does not require another VM for password entering, thus improving the CPU and memory consumption. Moreover, HyperPass is implemented as a thin hypervisor, which further improves its performance compared to T-PIM.

HyperPass is also advantageous from the perspective of user experience, since it does not require the user to switch to another VM to enter a password.

7 CONCLUSIONS

The design of ARM's TrustZone allows software modules running in an isolated environment to control hardware devices in an exclusive manner. Specifically, turning on the security indicator is possible only from the isolated environment.

We have shown that a comparable control of the keyboard LEDs can be achieved using virtualization. Although the performance degradation due to interception of PS/2 communication is negligible, it is unclear what will be the impact of controlling a USB keyboard.

We believe that the presented approach can be used in general for guaranteeing certain invariants in the state of hardware devices.

REFERENCES

Alipay (2020). Alipay. https://intl.alipay.com/.

Axelsson, S. (2000). Intrusion detection systems: A survey and taxonomy. Technical report, Technical report.

BitCoin (2020). BitCoin Ledger. https://www.ledgerwallet.com/beta/trustlet/.

Boneh, D., Inguva, S., and Baker, I. (2007). SSL MITM Proxy. http://crypto.stanford.edu/ssl-mitm/.

Bugiel, S., Dmitrienko, A., Kostiainen, K., Sadeghi, A.-R., and Winandy, M. (2010). TruWalletM: Secure web authentication on mobile platforms. In *International Conference on Trusted Systems*, pages 219–236. Springer.

Chapweske, A. (2003). The PS/2 mouse/keyboard protocol. *electronic file available: http://www.computer-engineering.org/ps2protocol*.

Costan, V. and Devadas, S. (2016). Intel SGX Explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118.

Deshane, T., Shepherd, Z., Matthews, J., Ben-Yehuda, M., Shah, A., and Rao, B. (2008). Quantitative comparison of Xen and KVM. *Xen Summit, Boston, MA, USA*, pages 1–2.

Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*, 2014.

Hirano, M., Umeda, T., Okuda, T., Kawai, E., and Yamaguchi, S. (2009). T-pim: Trusted password input method against data stealing malware. In *2009 Sixth International Conference on Information Technology: New Generations*, pages 429–434. IEEE.

Hussain, M., Al-Haiqi, A., Zaidan, A., Zaidan, B., Kiah, M. M., Anuar, N. B., and Abdulnabi, M. (2016). The rise of keyloggers on smartphones: A survey and insight into motion-based tap inference attacks. *Pervasive and Mobile Computing*, 25:1–25.

Kiperberg, M., Leon, R., Resh, A., Algawi, A., and Zaidenberg, N. J. (2019). Hypervisor-based Protection of Code. *IEEE Transactions on Information Forensics and Security*, 14(8):2203–2216.

Knox, S. (2013). White paper: An overview of samsung knox.

Leon, R. S., Kiperberg, M., Zabag, A. A. L., Resh, A., Algawi, A., and Zaidenberg, N. J. (2019). Hypervisor-Based White Listing of Executables. *IEEE Security & Privacy*, 17(5):58–67.

Li, X., Hu, H., Bai, G., Jia, Y., Liang, Z., and Saxena, P. (2014). DroidVault: A trusted data vault for Android devices. In *2014 19th International Conference on Engineering of Complex Computer Systems*, pages 29–38. IEEE.

Liu, D. and Cox, L. P. (2014). Veriui: Attested login for mobile devices. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, pages 1–6.

Microsoft (2012). Requirements for Implementing the Microsoft Hypervisor Interface. *MSDN, [Online]. Available: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/tlfs .[Accessed Oct 2020]*.

Neiger, G., Santoni, A., Leung, F., Rodgers, D., and Uhlig, R. (2006). Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal*, 10(3).

Pfoh, J., Schneider, C., and Eckert, C. (2011). Nitro: Hardware-based system call tracing for virtual machines. In *International Workshop on Security*, pages 96–112. Springer.

Pinto, S. and Santos, N. (2019). Demystifying ARM TrustZone: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 51(6):1–36.

Rescorla, E. et al. (2000). Http over tls.

Rosenbaum, A., Biham, E., and Bitan, S. (2019). *Trusted Execution Environments*. PhD thesis, Computer Science Department, Technion.

Rubinov, K., Rosculete, L., Mitra, T., and Roychoudhury, A. (2016). Automated partitioning of android applications for trusted execution environments. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 923–934. IEEE.

Samsung (2020). Samsung Pay. http://www.samsung.com/us/samsung-pay/.

Seshadri, A., Luk, M., Qu, N., and Perrig, A. (2007). SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 335–350.

Sharma, R. K., Kalita, H. K., and Issac, B. (2014). Different firewall techniques: A survey. In *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pages 1–6. IEEE.

Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., et al. (2009). Bitvisor: a thin hypervisor for enforcing i/o device security. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 121–130.

Sibai, F. N. (2008). Evaluating the performance of single and multiple core processors with PCMARK® 05 and benchmark analysis. *ACM SIGMETRICS Performance Evaluation Review*, 35(4):62–71.

Solovev, A. (2020). GitHub Gloc. https://chrome.google.com/webstore/detail/github-gloc/kaodcnpebhdbpaeeemkiobcokcnegdki.

Ying, K., Ahlawat, A., Alsharifi, B., Jiang, Y., Thavai, P., and Du, W. (2018). Truz-droid: Integrating trustzone with mobile operating system. In *Proceedings of the 16th annual international conference on mobile systems, applications, and services*, pages 14–27.

Zimmer, R. (2009). Hale,"UEFI: From Reset Vector to Operating System," Chapter 3 of Hardware-Dependent Software.