Poster: SwitchMan: An Easy-to-Use Approach to Secure User Input and Output [1]

Shengbao Zheng*, Zhengyu Zhou*, Heyi Tang† and Xiaowei Yang*
*Duke University, †Tsinghua University
*{szheng, zzy, xwy}@cs.duke.edu, †tangheyi.09@gmail.com

I. Introduction

Modern operating systems for personal computers (including Linux, MAC, and Windows) provide user-level APIs for an application to access the I/O paths of another application. This design facilitates information sharing between applications, enabling applications such as screenshots. However, it also enables user-level malware to log a user's keystrokes or scrape a user's screen output. As an example, between 2013 and 2015, attackers used the Carbanak malware [1] to infect bank computers and then recorded videos of a victim's screen and keystrokes to obtain sensitive banking information. They successfully stole money from around 100 financial institutions and the total financial loss amounted to almost one billion dollars.

Previous proposals to address this challenge fall into four broad categories. Work in the first category protects a user's I/O paths at the hardware level. An example is the Intel Protected Transaction Display (PTD) solution [2]. Work in the second category proposes to use a second mobile device as a trusted input/output device [3]. Work in the third category uses virtual machines to isolate trusted applications [4]. Finally, work in the fourth category proposes to enhance an OS by implementing fine-grained access control for I/O interfaces so that one application cannot access another application's I/O paths by default [5].

Each of the previous solutions requires a unique trusted computing base (TCBs). However, they all require significant user management to achieve the desired level of security. A user needs to decide which data are sensitive and then switch to a trusted hardware (e.g. a mobile device) or a trusted terminal (e.g., one runs inside a trusted virtual machine) or both to input/output sensitive data. We hypothesize that it is challenging for a non-expert user to manage these tasks. Therefore, it is beneficial to explore a design alternative that can automatically manage the switching to sensitive data input/output without user involvement.

In this work[1], we propose SwitchMan, an architecture that enables a server to switch a user to a secure terminal for sensitive user input/output. At the heart of SwitchMan lies a protocol that enables a remote server (e.g. a web server) to embed a secure terminal switching request inside its traffic stream even if the client's software (e.g. a browser) is untrusted. The TCB running on the client will intercept the request and switch the user to a secure terminal.

[1]The poster is associated with an S&P Workshop paper. It will appear at 2019 International Workshop on Privacy Engineering (IWPE19).

II. Assumpsion

SwitchMan's design assumes that a user's OS kernel and the graphical system distributed with the OS can be trusted.

We make this assumption mainly because of our design goals. Trusting the OS makes SwitchMan easy-to-use. By trusting the OS, we can provide a turnkey solution to the user. An OS vendor can distribute SwitchMan with the OS and turn it on by default. We expect that ease of use can increase the chance of user adoption.

Admittedly, trusting the OS has the drawback that when a user's OS is compromised, SwitchMan cannot secure the access to sensitive user input and output data. However, we believe there are a few remedies that can reduce the security risk of this assumption.

First, there exist techniques such as the Integrated Measurement Architecture (IMA) that can measure and attest an OS's integrity. Second, trusting an OS significantly reduces the TCB compared to the status quo. Today, if one application residing in a user's account is compromised, the application can steal sensitive user input/output. We obtain data from the Common Vulnerabilities and Exposures (CVE) dataset [6] for the time period from 2013 to 2018. We find that the percentage of privilege escalation vulnerabilities and root privilege vulnerabilities among all vulnerabilities are in the range of $[3.01\%, 9.34\%]$ and $[0.17\%, 0.65\%]$ respectively. This result shows that the number of OS vulnerabilities is much fewer than the total number of vulnerabilities, suggesting that trusting the OS rather than all applications can significant reduce the security risk of data leakage. Finally, there exists market competition among OS vendors. The OS vendors are accountable for security breaches caused by OS compromises, and accountability can motivate an OS vendor to improve its security, reducing the risk that the OS is compromised.

III. SwitchMan Design

1) SwitchMan Architecture: Figure 1 shows the SwitchMan's architecture. A computer's OS assigns two user accounts to one user. One is a regular account, where the user has the freedom to run any application. The other is a protected account coming with a set of pre-configured software that the OS manufacturer trusts. A main purpose

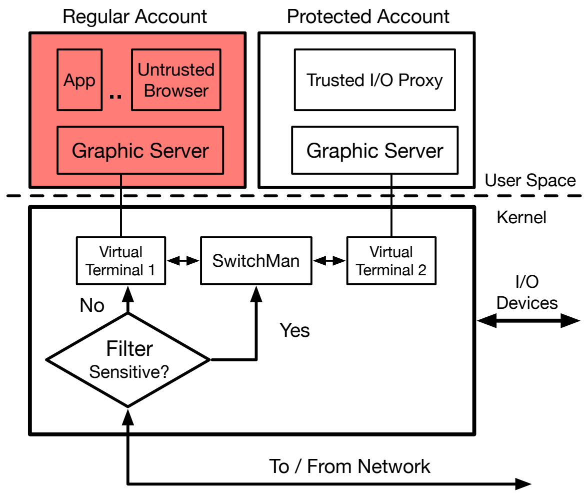

Figure 1. **The SwitchMan Architecture.**

of this account is to provide a trusted terminal for users to input/output sensitive data. Each user account has its own display server. Applications running under the regular account cannot connect to the protected account's display server. Each server uses its own virtual terminal so that their I/O paths are isolated at the software level.

The design of SwitchMan includes four main components: 1) a program called the Trusted I/O Proxy (TIOP) running under a user's protected account; 2) a kernel module called SwitchMan for managing the switching between a user's two accounts; 3) a kernel filter for managing sensitive network input/output data; and 4) a network protocol called SwitchMan's Network Protocol (SNP) that enables a remote server to send a request to a user's OS to switch the user to his protected account for accessing sensitive input/output data.

2) Trusted Input/Output Proxy(TIOP): In the SwitchMan design, a user interacts with sensitive data via TIOP. One can view TIOP as a simple web browser distributed by a user's OS vendor. It displays the sensitive output received from a remote server and takes a user's input. TIOP is the only application connected to the virtual terminal running under the protected account.

3) SwitchMan's Network Protocol (SNP): SwitchMan's design includes an HTTPS-based protocol called SNP. SNP enables a remote server to securely request the SwitchMan OS to switch a user to his protected account. Figure 2 and Algorithm 1 describes how SNP works.

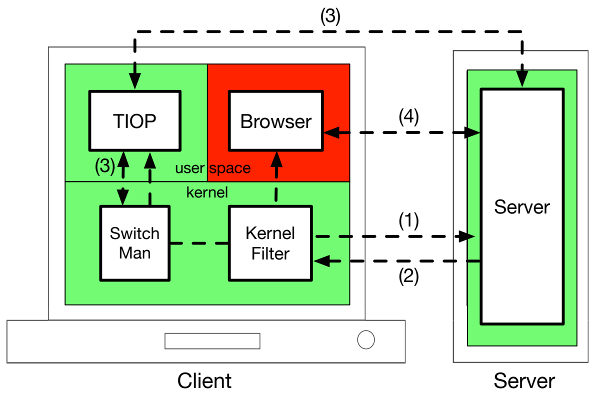

Figure 2. **This figure shows how SwitchMan's network protocol works at a high level. 1) A client connects a server via HTTPS. 2) If a server desires to receive sensitive data or display sensitive data, it notifies the client's kernel. The client's OS intercepts this signal, and switches an eligible server session to a user's protected account. 3) The user uses the trusted TIOP program to interact with the server. 4) The user resumes his previous session in his regular account after the TIOP session finishes.**

Algorithm 1 SNP Protocol

```
// TCP Handshake.
$ClientOS \rightarrow Server : SYN$ w/ $Opt(SM)$
$ClientOS \leftarrow Server : SYNACK$ w/ $Opt(SM\_Echo)$
$ClientOS \rightarrow Server : ACK$
// Server Initiated Switching
$Browser \rightarrow Server : Request$
$Browser \leftarrow Server : Normal\_Data$
// First half of the switching request.
$TIOP \leftarrow Server : TCP\_Option(nonce_1, nonce_{id})$
// Second half of the switching request.
$Browser \leftarrow Server :$
$https(JS(URL_{sensitive}, nonce_2, nonce_{id}, signature))$
// TIOP Connects to the Server
$TIOP \rightarrow Server :$
$https(URL_{sensitive}, nonce_1, nonce_2, nonce\_id)$
$TIOP \leftrightarrow Server : Sensitive\_Data$
```

REFERENCES

[1] Kaspersky Lab, " The Great Bank Robbery: Carbanak APT," https://securelist.com/the-great-bank-robbery-the-carbanak-apt/68732/, 2015.

[2] Intel, "Intel Identity Protection Technology with Protected Transaction Display," https://www.intel.com/content/www/us/en/architecture-and-technology/identity-protection/identity-protection-technology-general.html, 2012.

[3] J. M. McCune, A. Perrig, and M. K. Reiter, "Bump in the ether: A framework for securing sensitive user input," in *USENIX ATC*, 2006.

[4] J. Rutkowska and R. Wojtczuk, "Qubes OS Architecture," *Invisible Things Lab Tech Report*, p. 54, 2010.

[5] M. James, "Secure and Simple Sandboxing in SELinux," https://www.slideshare.net/jamesmorris/secure-and-simple-sandboxing-in-selinux, 2009.

[6] CVE, "Common vulnerabilities and exposures," https://cve.mitre.org.