Securing Network Input via a Trusted Input Proxy

Kevin Borders, Atul Prakash
University of Michigan
{kborders, aprakash}@umich.edu

Abstract

The increasing popularity of online transactions involving sensitive personal data, such as bank account and social security numbers, has created a huge security problem for today's computer users. Malicious software (malware) that steals passwords and other critical user input has led to countless cases of identity theft and financial fraud. Client computers remain susceptible to key logging attacks due to inadequate defense against drive-by malware installation. Trusted browsing virtual machines attempt to mitigate this problem, but fail to protect against many runtime and Trojan horse malware attacks. One option for securely acquiring sensitive input is TPM-verified trusted execution. While this method promises to provide the best security, it has serious usability limitations and would require extensive modifications to both the client and the server.

We propose a new approach for securing network input that relies on a Trusted Input Proxy (TIP). The TIP runs as a module in a virtual machine architecture that proxies secure network connections. When a user wishes to enter sensitive data, he or she presses an escape sequence that triggers the TIP to display a secure input dialog. The TIP will automatically generate a *placeholder* value based on the input using regular expression approximation (e.g. 123-45-6789 for a SSN). It will then send key presses for the placeholder to the application. Finally, the TIP will substitute actual data for placeholders as it relays network messages to the server. Although the Trusted Input Proxy approach relies on a slightly larger trusted code base, it requires no modifications to the server, very few to the client, and is far more usable than TPM-verified execution. In this paper, we present the initial design of a Trusted Input Proxy and compare its merits and shortcomings to those of other approaches.

1. Introduction and Related Work

A growing number of people use the internet for sensitive financial transactions. Consequently, fraud and identify theft resulting from stolen credentials has skyrocketed over the past decade. Online credit card and bank account fraud now costs financial institutions billions of dollars every year. Much of this fraud occurs due to weaknesses in client-side security. As an example, a hacker can install malicious software that records keystrokes by getting a user to click on a link to a malicious website in an e-mail. Once recorded, keystrokes including data such the user's name, bank account password, address, social security number, are sent back to server and may be used for identity theft or other fraud.

A number of efforts to improve client-side security, such as personal firewalls, anti-virus software, and automatic security updates, have helped to curb the volume of malware infection. However, these mechanisms do not protect against most "drive-by download" attacks, which might involve tricking a user into downloading malicious software or exploiting a bug in an application. Systems such as Sandboxie [6] and Greenborder [1] go further to protect client security by isolating the web browser. With this approach, a compromised browser will have limited access to the rest of the system. Unfortunately, a compromised browser *will* have access to sensitive keyboard input that it receives during a browsing session and can send that data out over the internet. Furthermore, the underlying system is still vulnerable to a Trojan horse attack where the user downloads and installs a seemingly benign program outside of the sandbox that contains malware.

We first examine a better way of solving this problem based on new Trusted Platform Module (TPM) capabilities for executing a trusted code block to obtain human input. This method is very secure due to an extremely small trusted code base (TCB). We find, however, that this approach is not very portable and requires extensive client and server-side changes. It can be vulnerable to a human security problem similar to standard phishing.

Our main contribution in this paper is the initial design of a Trusted Input Proxy (TIP) architecture that enables secure network input. The architecture consists of three components: keyboard input interception, placeholder generation, and a secure connection proxy. The user's operating system (OS) and applications run inside of a primary virtual machine (VM) and the TIP module runs in a TIP VM. The virtual machine monitor (VMM) sends keyboard and network I/O through the TIP, which mediates and forwards the device I/O to the primary VM. The TIP and VMM are considered to be part of the TCB, and are isolated from attack by the guest OS. The TIP will intercept a secure input keyboard escape sequence and enter into secure input mode. After acquiring sensitive input, it will generate a placeholder and send it to the guest OS. Finally, the TIP substitutes real sensitive data for placeholders in outbound network requests before relaying them to their destination servers.

The Trusted Input Proxy approach requires no server-side modifications or management overhead, and minimal client OS/application changes. Also, we argue that using an escape sequence for securing input is much more effective from a usability perspective than application-initiated secure input. The TIP solution is portable and backwards compatible (non-TIP clients can still log in). One shortcoming of the TIP solution is that it requires the client's main operating system to run inside of a VM, which can be a limitation for users who have scarce hardware resources, run intensive graphical applications, or require compatibility with obscure devices. We expect that many of these limitations will disappear as hardware manufacturers increase virtualization support. Another disadvantage of a TIP compared to TPM execution is a significantly larger trusted code base. However, the TCB is still much smaller than in a standard OS, and we feel that its larger size is more than justified by the increase in deployment practicality and usability provided by the TIP architecture.

Ross et al. describe a method for protecting passwords against *server-side* theft (i.e. phishing) in [5]. Their solution does not help against a completely compromised client. Jackson et al. mention obtaining secure input for web forms [2], but do not discuss how to run a proxy that inserts sensitive data. Jammalamadaka et al. propose a novel method for secure web input on an untrusted client machine using a trusted proxy and trusted mobile device (e.g. cell phone) to send sensitive input to the server [3]. Theoretically, this is very similar to the TIP architecture, with the trusted proxy on the VMM and it acquiring input directly from the keyboard instead of from a cell phone. The TIP solution, however, is much more usable and easy to deploy, requiring the user to have one computing device to send secure input to a server instead of three.

2. TPM-Verified Input Acquisition

One method of protecting against key-logger type attacks is to move input acquisition into a trusted system component. McCune et al. outline a method for executing security-sensitive code in an environment that is completely isolated from the rest of the system by utilizing TPM capabilities [4]. The TPM verifies a small block of trusted code, suspends the rest of the system, allows the code to run, and then restores the system. During this process, the TPM signs a digest of the trusted code, its inputs, and its outputs so that the result can be verified by an external entity. In the case of secure input, the trusted code could obtain the user's password or other sensitive data, ask the TPM to sign it, encrypt it with the server's public key, and then hand it off to the operating system so that it can be sent to the server without the operating system being able to read the data or compromise its integrity.

The TPM-verified execution approach does offer a high level of security due to the very small size of the trusted code base (TCB), which would just include a keyboard driver, a simple graphics driver, a TPM driver, and some setup code. However, it also has some serious limitations. First, if the server is going to verify the code execution and resulting value, it needs to have a public signing key, along with some certification that the key is legitimate, for every client. On the plus side, this would add another authentication factor, making the process more secure. Verifying all these certificates, however, would require a potentially expensive public key infrastructure (PKI). Furthermore, one would only be able to log in from a machine that has been certified by the server, which constrains accessibility.

Another problem with TPM-verified execution is that the operating system, which is not trusted, is responsible for invoking the TPM functionality. Not only does this require changes to the OS, it also allows an attacker to modify the OS so that it does not call the secure input module. Instead, the OS could display some sort of error and allow the user to proceed with divulging sensitive data even though the data is not being sent to the server. This is not only a denial of service, but a denial of *secure* service, which has shown to be a very effective way of getting users to behave insecurely [7].

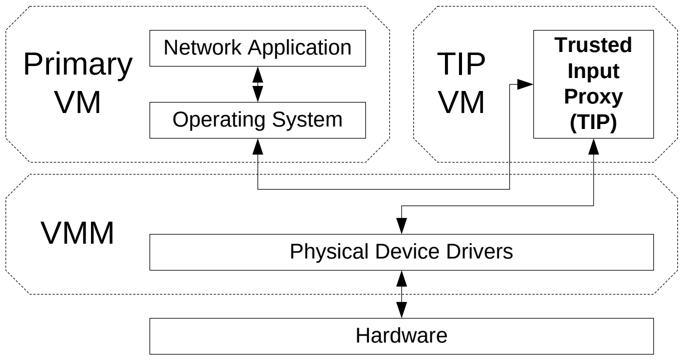

Figure 1. Trusted Input Proxy architecture. The TIP mediates keyboard input and network I/O.

A different way of using a TPM for secure input that does not require a PKI is to have the trusted code access a secret value using TPM sealed storage and display the secret to the user. (The secret could be text or a picture.) This would assure the user that keyboard input is going to the trusted code and is safe from eavesdropping. Next, the trusted code could hash the input (with an optional salt value), or verify the server's certificate and encrypt the input so that it cannot be read by the OS. With this approach, server modification would still be necessary to decrypt or verify individual form fields. Displaying a secret for integrity verification could also enhance the security of the TIP, described next, against malicious VMM modifications.

3. Trusted Input Proxy Design

The Trusted Input Proxy (TIP) is a module that runs in its own virtual machine and mediates some I/O between the physical device drivers and the guest operating system. An illustration of this architecture can be seen in Figure 1. In our design, the TIP module acts as a proxy for secure network connections. It also receives all keyboard input, and can choose whether or not to forward key presses to the guest OS. The size of the trusted input proxy is significant, as it must contain code to do basic protocol processing for at least SSL and HTTP. However, it is still far smaller than the primary virtual machine, which runs all of the user's applications.

3.1 Keystroke Interception

The trusted input proxy receives all physical key press notifications from the keyboard driver in the VMM. It can decide whether or not to forward each keystroke to the guest operating system, and can change, replace, or delete keystrokes. When the user activates an escape sequence on the keyboard, the TIP will drop the key event, go into secure input mode, and display a dialog box on the screen. The escape sequence should be something that generates a single key event, which means any of the ctrl, alt, and shift keys combined with one standard key. It should be something rarely used by default, like ctrl+alt+F12, but should also be easily configurable by the user. The machine will remain in trusted input mode until the user exits out of the dialog by hitting enter (OK) or escape (Cancel). Optionally, it may be desirable in the future to allow mouse input for the dialog box. In this case, clicking on OK or Cancel would also exit secure input mode.

While the user is in secure input mode, all keyboard input goes to the TIP, which does *not* forward it to the guest OS. Depending on the user's preference, the trusted input proxy can either display a dialog with the masked sensitive input (*'s for characters), the unmasked sensitive input (to avoid typos), or nothing at all. Having different escape sequences for each option is recommended. If the user chooses for the TIP to display a dialog box, it should appear on top of anything else in the display, be *unreadable* to the guest OS, and also contain the current automatically-generated *placeholder* value, which is discussed more in the next section. The user should also be able to override the default placeholder value here by modifying it in the dialog box. When the user is finished entering secure input, the TIP will generate keystrokes for the placeholder value and send them to the guest operating system.

3.2 Automatic Placeholder Generation

When the user enters some sensitive input, such as a social security number, the application may reasonably expect that the input passes some simple checks. For example, it may expect a number in the format "XXX-XX-XXXX" where the X's are digits. If the TIP generates a placeholder that does not pass these checks, then the verification will fail and the user will have to manually come up with a placeholder, which may impede usability. There may be some situations where

Table 1. Examples of some regular expressions for placeholder generation. (Perl format)

Regular Expression	Meaning	Examples	Replace With
`([0-9]+[-)( ]*){1,}`	One or more digits, optionally followed by '-' '(' ')' or ' ', repeated *n* times where *n* > 0	135-79-2468, (555) 121 2121, 1123	Randomly replace each digit with another digit (leave other characters)
Upper or Lower case dictionary word	Any word in a password cracking dictionary – include names and common strings like "asdf"	secret, password, asdf, John	Random word of same case and length
`[a-z]+` or `[A-Z]+`	All-upper or all-lower case string	xbwfel, HJERGI	Random string with same length and case
`[a-zA-Z0-9]+`	Alpha-numeric string	AbC456, gOwe23Jikn45p, oioj3249	Random string with same length and at least one character from each set present in string (i.e. upper, lower, or digit)

input validity depends on checksum verification, such as for credit card numbers. These checks are usually performed at the server, but could be done on the client. We plan to investigate this issue in the future, and create replacement algorithms that also match checksums or disable client-side checks. Some websites also have Javascript which will give you a hint about the strength of the password you are choosing. If the user enters a six-letter dictionary word for a password, and the placeholder is a random ten-digit alphanumeric, then the user may get a false sense of security from the client-side checks.

To deal with this problem, the TIP uses a regular expression matching and generation list that generates an equivalent placeholder for given sensitive input. It tries to match the input against an ordered list of regular expressions, with less restrictive regular expressions at the end. Table 1 lists some sample regular expressions, in approximate order from most to least restrictive, and what they should be replaced with to maximize equivalency without compromising private information. In the case where the TIP is unable to come up with an acceptable replacement or the user wants the replacement to be something easy and memorable, he or she can manually specify the placeholder. The goal of placeholders is to make the input proxy process as simple as possible and avoid changes in user behavior.

3.3 Secure Network Connection Proxy

In order to proxy secure network connections, the TIP module needs to effectively act as a man-in-the-middle without compromising security. For SSL connections, our solution is to add a certificate for the TIP to the list of root certification authorities in the VM's browser. This way, the trusted input proxy can copy the contents of the server's certificate (after verifying its authenticity) into a new one, replacing the server's public key with its own. This way, any errors or warnings that may appear as a result of an out-of-date certificate or a certificate that does not match the hostname will still appear, and the trusted input proxy can effectively intercept and modify SSL traffic. Similar approaches can be taken for other secure network protocols. With protocols like SSH that rely on manual key fingerprint verification, the TIP will need a way of displaying the actual key fingerprint to the user. Support for additional protocols would increase the size of the TCB and require writing extensions to the TIP. However, most people only use SSL and possibly one other popular protocol for secure communication, limiting the size of the TCB.

Once the TIP has established a secure connection with both the virtual machine and the remote server, it inspects every message that it gets from the client to search for placeholder strings. The TIP will be more effective if it has some knowledge of the protocol, such as HTTP, so that does not have to rely on a blind search. This is important for protocols where the client sends a lot of random data that has the potential to match a random placeholder. Also, it is necessary for protocols where the placeholders are encoded before being sent over the wire. An example is HTTP authorization, which uses base-64 encoding. With HTTP over SSL, the TIP decodes the authorization field to look for placeholders, and also looks for them in parameters of GET and POST requests according to the specification. When it finds a known placeholder, the TIP then replaces it, rebuilds the request, and forwards it to the server.

4. Benefits and Limitations of a Trusted Input Proxy

The Trusted Input Proxy architecture requires *no* changes to the user's standard operation system and applications other than adding the TIP as a certification authority and migrating the OS into a virtual machine, which can be done efficiently and transparently. The TIP architecture is also transparent to the server, which requires no modification. In terms of practical security, a trusted input proxy is likely to be accepted and used correctly by people who care about their privacy. Entering a two or three-key simultaneous escape sequence then hitting enter afterwards is a minimal perturbation to a user's standard workflow, and users already have a good idea of what data is sensitive and requires secure input (passwords, SSN, account numbers, etc.). Furthermore, the user will become accustomed to entering sensitive data through the TIP, making it hard for malware on the guest OS to trick a user into typing sensitive input without using the escape sequence. (The escape sequence will guarantee confidentiality as long as the TIP is installed.)

The Trusted Input Proxy architecture relies on a trusted code base of moderate size, including network I/O, keyboard drivers, and placeholder generation code. If code base size is a concern, SSL interception and certificate proxying can be done by a network service so that it is not part of the TCB. The TCB is still much smaller than that of a standard operation system, and its size compared to TPM-verified code is justified by its hugely improved usability and reduced deployment requirements. The TIP is also currently limited by its reliance on virtual machine technology. Some users may not want to run their main operating system inside of a VM for performance reasons. Virtual machines require more system resources (especially memory) and can be slightly slower for I/O intensive applications. However, virtual machine performance is usually very good compared to native execution, and it continues to get better as processor manufacturers add features to optimize virtualization.

5. References

[1] Green Border Technologies. GreenBorder's Virtual Session Architecture. http://www.greenborder.com/technology/ architecture.php, 2007.

[2] C. Jackson, D. Boneh, and J. Mitchell. Spyware Resistant Web Authentication using Virtual Machines. http://crypto.stanford.edu/spyblock/spyblock.pdf, 2006.

[3] R. Jammalamadaka, T. Van Der Horst, S. Mehrotra, K. Seamons, and N. Venkasubramanian. Delegate: A Proxy Based Architecture for Secure Website Access from an Untrusted Machine. In *Proc. of the 22nd Annual Computer Security Applications Conference (ACSAC '06)*, 2006.

[4] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri. Minimal TCB Code Execution (Extended abstract). In *Proc. of the 2007 IEEE Symposium on Security and Privacy (SP '07),* 2007.

[5] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. Mitchell. Stronger password authentication using browser extensions. In *Proc. of the 14th Usenix Security Symposium*, 2005.

[6] Sandboxie Transient Storage. http://www.sandboxie.com, 2007.

[7] S. Schechter, R. Dhamija, A. Ozment and I. Fischer. The Emperor's New Security Indicators: An Evaluation of Website Authentication and the Effect of Role Playing on Usability Studies. In *Proc. of the 2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007.