Approved for Public Release: 12-0024
Distribution Unlimited.

SeRPEnT: Secure Remote Peripheral Encryption Tunnel

David Weinstein, Xeno Kovah, Scott Dyer
The MITRE Corporation[*]
{*dweinstein,xkovah,sdyer*}@*mitre.org*

Abstract

Client endpoint systems are a prime target for attackers of every sophistication level. These systems take part in many transactions demanding a degree of trust that cannot be placed in a general-purpose, commodity, computer system. We propose that these sensitive transactions can be made more secure by creating a new kind of trusted path, one that connects a server *directly* to a client's hardware peripherals. This capability has been designed to isolate a compromised endpoint from its peripherals during security sensitive applications. Such connectivity could be made unforgeable, strong against eavesdropping and tied to a user's credentials using end-to-end cryptography.

We present a prototype Secure Remote Peripheral Encryption Tunnel (SeRPEnT) for the Universal Serial Bus (USB). Our device is a small, low-power "cryptographic switchboard" that tunnels connected peripherals to a server with Virtual Machine(VM)-hosted applications. SeRPEnT can also pass-through devices to the client system, allowing normal use of the local system by the user. SeRPEnT enables secure transactions between the user and server applications by only allowing input to these VMs to originate from our portable embedded device. SeRPEnT thus drastically reduces the attack surface currently exposed to an adversary.

1 Introduction

In 1985, the Department of Defense Trusted Computer System Evaluation Criteria[1] defined a "trusted path" as "A mechanism by which a person at a terminal can communicate directly with the Trusted Computing Base. This mechanism can only be activated by the person or the Trusted Computing Base and cannot be imitated by untrusted software."

The explosive growth of the Internet and networked computing is a major difference between the world of 1985 and today. There are many remote services that depend on an ability to securely authenticate a transaction on behalf of a user. Online banking requires the guarantee that transfer of funds be initiated by the legitimate owner of the account. But in all of these cases, due to the lack of a trusted path, compromised endpoint systems have facilitated identity theft, bank fraud, and the theft of user credentials. Despite these vulnerabilities, servers continue to trust the client's operating environment and assume that all requests are initiated by the user. Instead, it must be assumed that the client system is compromised.

A major goal of our work is to re-architect the way users interact with remote applications and services. SeRPEnT facilitates a bidirectional cryptographic tunnel from the user's peripherals, through the compromised host, directly to the remote service. This provides much stronger authenticity of actions initiated by the user than those provided by commodity OSes. To achieve this we refocus trust on an embedded system with a much more restricted functionality than the client's general purpose computer, and a locked-down backend server that mediates user interaction as a focal point of security.

In Section 2 we describe the related work, in Section 3 we will examine the interactions between remote and local systems for different local system architectures. In Section 4 we describe the current implementation of SeRPEnT. We then describe how the implementation can be used in Section 5 and Future Work in Section 6. Finally in Section 7 we give our conclusions.

2 Related work

We discuss prior work on designs of trusted path systems for transactions on commodity hardware and software.

In 2006, McCune et al.[5] proposed a system of trusted input and display paths with an architecture named Bump

[*]DocITS no.: MP120013

© 2012 The MITRE Corporation. All Rights Reserved.

in the Ether (BitE). The original design was based on a number of assumptions: 1) that a mobile phone can be trusted as an input proxy (and usable monitor), 2) the host's OS kernel will not arbitrarily read and modify the memory space of processes running on the system and 3) the host system must be "capable of attesting to its current software state" with the aid of a Trusted Platform Module (TPM). The second assumption was later relaxed by Flicker[4], which introduced a mechanism for isolated code execution by leveraging new hardware support for security on modern x86 processors. In 2009, McCune et al. built upon BitE and Flicker with Bumpy [6], which utilized a USB interposer that provided encryption capabilities to the keyboard/mouse (input devices) in the architecture. The new design inherited the requirement that the user's host machine support Trusted Execution Technology (TXT) or AMD equivalent, in particular a *Dynamic Root of Trust* to ensure the code isolation within the untrusted host for processing trusted input. SeRPEnT follows the tradition of Bumpy's USB interposer but leaves behind a number of complexities–in particular the need to attest to the client system's state. We believe a simpler architecture that shifts the focus of code isolation from client to a professionally managed central SeRPEnT server/embedded device can provide trusted input and be widely deployed sooner. We also believe that mobile phones being used as a trusted monitor reduces usability, and because they lack the widespread adoption of a hardware root of trust, there is no reason to believe mobile systems are any less vulnerable today than host systems were in 2006.

The IBM ZTIC[7] is an embedded system designed for decoupling online banking sessions (in particular SSL) from the user's system. Instead, the HTTPS sessions are proxied through the ZTIC device for approving critical transactions. In this way the user's system need not be trusted with a private key nor trusted to maintain a certificate authority listing. Additionally, approval of banking transactions occurs on an alternate datapath that isn't on the critical path of an adversary. Whereas ZTIC was designed specially for online banking, SeRPEnT is applicable more broadly to additional remote access scenarios and a wider target audience.

3 Symbiote client: thick + thinner-than-thin

In a 2008 position paper, Laurie and Singer suggested that it was no longer realistic to expect an OS to maintain its full functionality and flexibility while also being able to provide a trusted path [3]. Instead they advocated for the extraction of mechanisms necessary to support the trusted path outside of the commodity OS and hardware, into a dedicated software/hardware device. While we agree with this position, unlike Laurie and Singer, we are not concerned with trying to posit what it would take to build something for everyone. Instead, we try to be something to someone and seek a cautious expansion from there. That is why SeRPEnT is designed as a thinner-than-thin-client.

Thin client systems gain trust through their simplicity. They are only meant to provide keyboard, mouse, video, networking, and sometimes a small amount of local storage. They are meant to be dumb terminals which just connect to some central backend server system, where the work is actually done. However, user demands for rich multi-media experience has forced manufacturers to compromise the purity of thin clients. Many units, including the majority of those from Wyse[1], include the capability to offload "rich media" processing for perennially-vulnerable formats such as Adobe Flash to the thin client. Instead, SeRPEnT "thinifies" thick clients by encrypting the user input, and using the thick client as a proxy forwarder to send the opaque data to the SeRPEnT server, as shown in Figure1. In its current form, SeRPEnT does not even accept any input from the untrusted endpoint that isn't encrypted by the server, and retains a very small attack surface.

We believe that when used creatively, trusted input, *even in the absence of trusted displays* is an 80% solution for most use cases requiring trusted remote computing. For instance, let's assume that the user system in Figure1 is completely compromised with the attacker having full privileges. On existing systems the user's keystrokes would follow path a), and go directly into the untrusted machine and be captured by the attacker. Even if the attacker doesn't capture the keystrokes, the authentication session is established between the client application on the untrusted system and the server application on the remote system. The attacker on the untrusted system can then proceed to send fake commands to the application as if he were the user. In path b) when SeRPEnT is being used, the user logs in to their bank with their keystrokes being encrypted from the local SeRPEnT embedded system to the SeRPEnT server. The attacker will not be able to use a keystroke sniffer to capture the credentials being sent to the remote system. More importantly, the attacker will not be able to *act on behalf* of the user on the remote system. This is because the endpoint is just a proxy forwarding along the encrypted traffic with no means to decrypt it. The authentication is actually established between the more trustworthy SeRPEnT server VM and the remote server. Thus the attacker who has compromised the user's system cannot in any way interact with the SeRPEnT server or remote server, since he

[1]http://www.wyse.com

© 2012 The MITRE Corporation. All Rights Reserved.

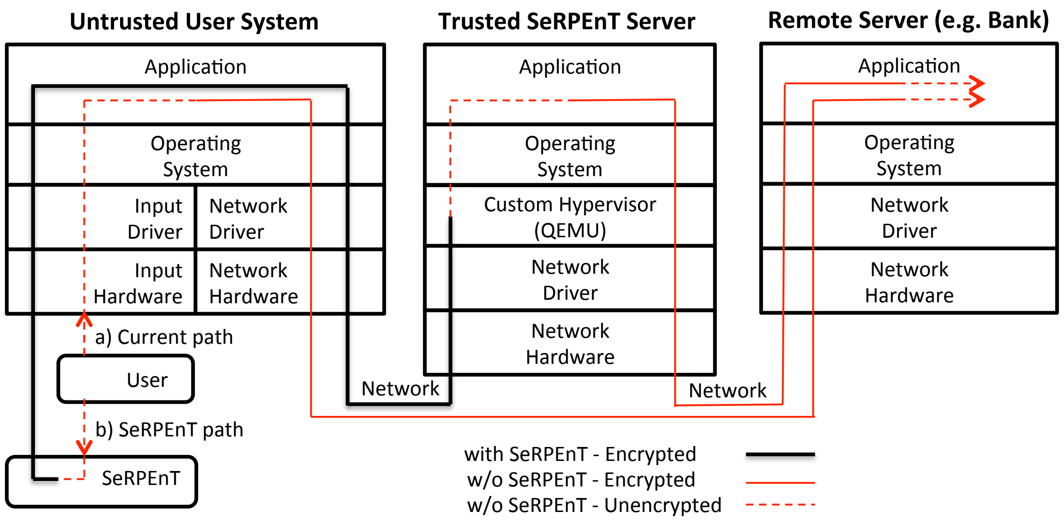

Figure 1: SeRPEnT creates a path for trusted input from the user

does not have physical access to the SeRPEnT embedded system. While the attacker may be able to see how much money the user has, he will not be able to initiate fraudulent transactions to steal the user's money as is common today. Being limited to virtual shoulder surfing in order to get data off of the system puts the attacker at a significant disadvantage because this dramatically increases the amount of data which must be sent to the attacker in order to collect information, potentially making this network traffic a candidate for detection. This situation is a far cry from the capability to completely impersonate a user that an attacker has today when he compromises her system.

It is extremely common to see compromise of user systems leading to the subversion of all authentication. For example, an online banking fraud operation using a trojan called Silentbanker[2] was uncovered in 2008 where criminals were circumventing the use of two-factor authentication. The attackers, having established a foothold (e.g., phishing or rich media vulnerability) on the user's system, had hijacked banking sessions after proper authentication, injecting commands into the communication stream between the web-browser and the server. The ability to interact with the remote system only through a trusted path, as provided by a system like SeRPEnT, can substantially mitigate this type of risk.

4 Implementation

Our working prototype is currently being used by a small group of system administrators and users with a need for enhanced operational security (OPSEC). We've designed it to support whichever USB peripherals must be part of a trusted input transaction, while only using components of the untrusted host when necessary and without sacrificing security. The host CPU, system memory and OS are untrusted. This iteration of the prototype has only implemented a trusted input path, but we believe our design can be extended for trusted display as well.

4.1 Hardware

The current design iteration leverages the ARM based BeagleBoard[3] model xM running a custom embedded Linux kernel and small root filesystem. A 2GB μSD card is capable of storing the OS and keying materials and thus leaving no other filesystem traces on the device. The current cost of the reference prototype (including a flashy case) is \$150. A similar prototype could be created for nearly \$25 using the anticipated Raspberry Pi[4] or any SoC that supports behaving as both a USB host and device simultaneously [5].

The BeagleBoard xM has hardware I/O to act as both a USB host and device, but it has a number of other subsystems that currently add only cost to the system. Ideally a product would implement only USB host and device (e.g., no need for S-video or HDMI display) to reduce cost, size, and further limit the potential for attacks on or misuse of the device. We identify the unused components in Figure 2.

[2]http://www.networkworld.com/news/2008/011408-silentbanker-trojan.html

[3]http://www.beagleboard.org

[4]http://www.raspberrypi.org

[5]We've not yet been able to confirm that the mini USB port on the Raspberry will be able to support our device-mode requirement

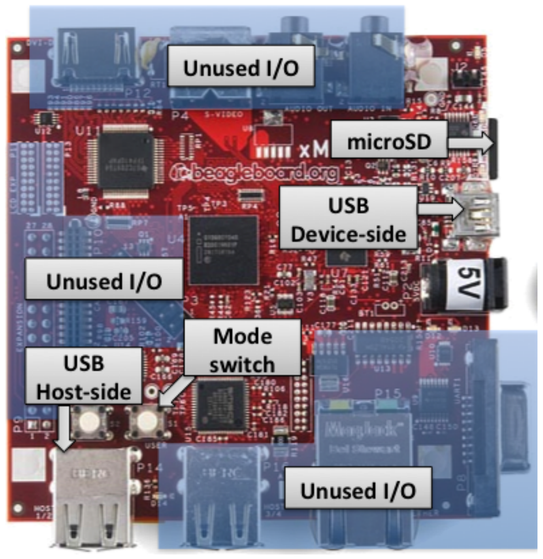

Figure 2: BeagleBoard xM hardware IO usage. The ethernet port is shown as unused, but is currently used only as part of firmware reflashing/development.

4.2 Embedded software

Filesystem and upgrading. A read-only filesystem is used to prevent filesystem corruption due to power loss. To allow for upgradability the board's ethernet hardware can be used. Alternatively, the low cost of the μSD card allows for an upgrade to be done by just sending a new card to the user. Upgrading via ethernet (SSH) allows a testing group to use the device (we refer to SeRPEnT throughout this section as "the device") while also allowing developers the freedom to evolve modifications without needing to be physically present at each device. When being upgraded, the firmware image signature is verified before the local filesystem is remounted with read-write privileges. Once the firmware image is written and after a successful reboot, the filesystem is returned to being mounted read-only. The current design meets the upgradability requirement outlined by Laurie and Singer.

Cryptography. We use standard OpenSSL libraries, currently using self-signed certificates (created at a central/trusted location) for mutual authentication between a SeRPEnT device and a SeRPEnT server. The algorithm providing symmetric encryption by OpenSSL (version 0.9.8o) is AES-256 in CBC mode and is robust against B.E.A.S.T[6] (and other chosen plain-text attacks) because there is no device-side scripting for an attacker to exploit. It hasn't yet been an issue, but for reduced power consumption and filesystem size an SSL implementation designed for embedded systems has been identified.

The current architecture uses two key pairs, one for *input* devices and the other for *output* (i.e., a display path). Before the device is distributed, the input private/public key pair is generated and stored to the μSD card of the device along with the public key of the server. The server stores the public key of each device. The second key pair labeled *output* is generated later and used for viewing the virtual machine's framebuffer from the client. As this key is currently stored on the user's untrusted system, these key pairs bear different levels of trust.

Gadgets. We use the Linux-USB Gadget API Framework[7] to create the appearance of USB devices to the user's host system. This enables the device to be used in pass-through or encrypted mode, described in greater detail in Section 5. The device loads a kernel module that allows it to to appear as a single USB device with two Human Interface Devices (HIDs) (keyboard and mouse) and a virtual serial port when plugged into a host system.

We also build upon USB/IP [2] so that drivers of hosted USB devices can be further isolated. This allows additional flexiblity and support for such USB devices as web cameras, mass storage devices, and audio to be incorporated as part of a trusted transaction. Simple devices like keyboard/mouse load drivers directly on our device, whereas other device drivers (or proprietary drivers) are loaded in an isolated server VM.

SeRPEnT appears as a multi-function composite device (i.e., a multi-interface, single composite device using USB Interface Association Descriptors) to a host system. This physical connectivity to the host is made via the USB device-side port shown in Figure 2. Gadgets can mimic real or common USB devices for which a commodity OS has drivers already loaded, or easily available. This flexibility allows for a mostly "plug-and-play" experience for the user.

We chose to use a CDC-ACM (serial) device as our encrypted transport primarily for simplicity. All the user system really needs is a means of receiving opaque (encrypted) data from SeRPEnT, and to send the packets to our server. This could also have been accomplished using a composite device with a mass storage device interface, which on some operating systems may be better supported than serial ports.

Sources and Sinks. The HID and serial gadgets described above are data sinks from the embedded system's perspective: data is proxied through them depending on the mode SeRPEnT is in as decided by the user's physical pressing of the mode switch (shown in Figure 2). These are static sinks in that they exist while the gadget module is loaded until the device reboots. While gadgets are mostly static, the *sources* on the other hand are dynamic, seamlessly supporting hotplug thanks to udev[8] events.

[6]http://www.cnsuk.co.uk/B.E.A.S.T.PDF

[7]http://www.linux-usb.org/gadget/

[8]udev is the device manager for the Linux kernel and manages nodes in the /dev tree

© 2012 The MITRE Corporation. All Rights Reserved.

Currently keyboards and mice are automatically supported as input devices. When plugged in, the Linux kernel loads the appropriate driver for the device, creates corresponding files in the */dev* tree, and finally a *udev* event is generated. A userspace process listens for these while filtering on keyboard/mouse devices to trigger registration of a new sources. Once registered, the open file handles for input devices are read on-demand using a standard **select** call. Since the prototype supports two basic modes of operation, data is sent either to the gadget HID descriptors (i.e., pass-through mode) or the gadget serial port (encrypted and tunneled mode). This is shown in Figure 3.

4.3 Untrusted system software

In addition to the device drivers loaded for SeRPEnT's HID and serial interfaces, the user system runs a serial-to-TCP *forwarder* agent written in Python. The *forwarder* is unsophisticated, it merely reads the encrypted data from the serial device and encapsulates these frames into a TCP stream to the server. The user endpoint also runs a standard VNC client with SSL[9] to connect to the server and view the current state of the VM's video framebuffer.

4.4 Virtualization

The backend software consists of a modified QEMU-KVM virtual machine with around 200 lines of changes, mainly to the handling of input sources in the VNC server code. The effect is that any input source other than those originating from a SeRPEnT device are dropped silently at the server. The sole purpose of the VNC Server is to manage the efficient distribution of the VM's video framebuffer to the client.

The changes made to QEMU create an alternative input-path (shim) fronted by an SSL decryption process, which decrypts the data sent by the device. QEMU has been modified to open a new (local) UNIX socket, which is opened by a SSL shim for writing and the QEMU instance for reading. If data successfully passes the decryption process it is re-injected in the virtual hardware to generate input events to the hosted operating system, effectively completing the trusted path to the VM. Our system is OS-agnostic requiring no modifications to the OS kernel or specialized drivers.

5 Usage

We will describe the use of SeRPEnT to protect an online banking transaction.

[9]Such as SSVNC or VNC + stunnel

The user's desktop is compromised by malware[10] specifically designed to hijack a browser banking session even though the bank is employing a second authentication factor, up to and including external hardware tokens such as RSA SecurID[11]. The malicious agent is designed to activate as soon as the banking session is initiated.

Without SeRPEnT the attacker can inject commands into the browser session, perhaps using a rogue browser extension that triggers when a particular website is detected by the trojan. The attacker, having targeted a specific bank, modifies outgoing payment account numbers on the fly and redirects payment to their accounts. The attacker also changes transaction records within the user's browser session to give the appearance of validity.

With SeRPEnT the SSL session from the browser to the banking server is no longer used in the same way. Instead, the bank hosts a SeRPEnT server which provides a VNC session, and along with the receipt of a SeRPEnT device the user has been instructed to plug their input peripherals into SeRPEnT and plug SeRPEnT into a USB port on their computer. She has also been instructed on the use of pass-through/trusted input modes, and taught to engage the button when the VNC session is loaded.

Pass-through and trusted input. The pushbutton on the device is used to switch modes between pass-through (unencrypted/local) and trusted path (encrypted/remote). After doing so an LED on the device will confirm the transition from pass-through into trusted input mode. The user will then see a virtual machine session (perhaps limited strictly to a browser window) with the request for user credentials. The user complies and is now exclusively interacting with a webpage otherwise unvailable without SeRPEnT. If she so chooses, the user may switch out of secure input mode by engaging the mode switch again, allowing easy multi-tasking. The mouse cursor now returns to life on the user's desktop. She checks her personal email before turning to secure mode and making a payment. This leaves the attacker dumbfounded as to how the mouse cursor/keystrokes in the VM are entering the banking session. The frustrated attacker finds it impossible to change any information within the session and finding that all that can be done is observe.

On the backend the bank has to make minimal changes to utilize SeRPEnT. They may choose to partner with a third party that hosts SeRPEnT-enabled virtual machines or do it all in-house. Any changes required to the service are only required so that an attacker can't by any other means inject input into banking session–i.e., the only way in must be via a SeRPEnT server and thus controlled only via enabled input devices. The web server that serves banking content/manages transactions should not be routable except to either a virtual tunnel be-

[10]e.g., Zeus, Poison Ivy, SilentBanker

[11]http://www.rsa.com

© 2012 The MITRE Corporation. All Rights Reserved.

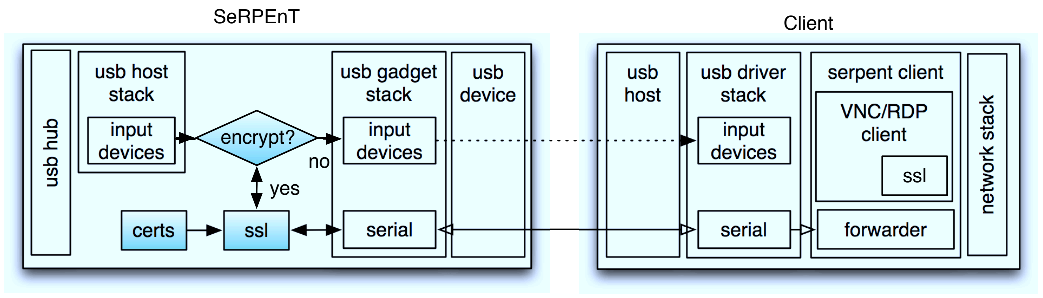

Figure 3: Information flow from input devices to user systems.

tween the SeRPEnT server or through the use of a second physical network interface card (NIC) on the SeRPEnT server with appropriately configured routing tables.

Other use cases. For instance, contrast the preceding banking scenario to that of a system administrator doing remote administration. Today, sysadmins use an IP-KVM, VNC, or Microsoft RDP. In all of these systems, the compromise of the administrator's endpoint means the possibility of compromise for every system the administrator interacts with thereafter. Power users like system administrators are often overconfident that their personal OPSEC behavior is sufficient, despite the fact that they too can fall victim to 0-day exploits if targeted. It is not hard to imagine a trojan like Silentbanker being retrofitted to search for IP-KVM client sessions instead of banking sessions. If instead the administrator's authentication and interaction with these remote systems were taking place over the trusted path provided by SeRPEnT, the risk of an attacker gaining the full privileges of the administrator is drastically reduced. Finally, the SeRPEnT server could co-locate the service and data that a user needs to access, with each user having a separate VM instance and roaming profiles. This would eliminate the need for the secondary Remote Server shown in Figure 1 and any configuration challenges that may introduce.

6 Future work

Future work on SeRPEnT will focus on trusted display paths, alernatives methods of user authentication to SeRPEnT (e.g., smart-cards, DOD CAC), and enabling SeRPEnT capabilities in mobile devices.

7 Conclusion

Decoupling key subsystems of the commodity computer and creating cryptographic compartments in small, portable hardware external to the untrusted system can be used to create trusted paths for critical transactions. Given a limited budget that can be spent by an organization on security, our belief is that this approach could lead to savings on managing intrusions introduced by poorly configured or unpatched client systems. Rather than hardening the vast number of disparate client systems each individually, we could instead focus on hardening the purpose-built subsystems and servers. Finally we believe that by maintaining a high degree of usability for the user and not by depending on thick-client capabilities, we simply allow the user a decision of being in one of two domains untrusted local or trusted remote.

References

[1] DEFENSE, D. O. Standard Department Of Defense Trusted Computer System Evaluation Criteria, 1985.

[2] HIROFUCHI, T., KAWAI, E., FUJIKAWA, K., AND SUNAHARA, H. USB/IP: a peripheral bus extension for device sharing over IP network. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), ATEC '05, USENIX Association, pp. 42–42.

[3] LAURIE, B., AND SINGER, A. Choose the Red Pill and the Blue Pill. In *NSPW '08* (Lake Tahoe, CA, September 2008).

[4] MCCUNE, J. M., PARNO, B. J., PERRIG, A., REITER, M. K., AND ISOZAKI, H. Flicker: an execution infrastructure for tcb minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008* (New York, NY, USA, 2008), Eurosys '08, ACM, pp. 315–328.

[5] MCCUNE, J. M., PERRIG, A., AND REITER, M. K. Bump in the ether: a framework for securing sensitive user input. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference* (Berkeley, CA, USA, 2006), USENIX Association, pp. 17–17.

[6] MCCUNE, J. M., PERRIG, A., AND REITER, M. K. Safe Passage for Passwords and Other Sensitive Data. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)* (Feb. 2009).

[7] WEIGOLD, T., KRAMP, T., HERMANN, R., HÖRING, F., BUHLER, P., AND BAENTSCH, M. The Zurich Trusted Information Channel — An Efficient Defence Against Man-in-the-Middle and Malicious Software Attacks. In *Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies: Trusted Computing - Challenges and Applications* (Berlin, Heidelberg, 2008), Trust '08, Springer-Verlag, pp. 75–91.

© 2012 The MITRE Corporation. All Rights Reserved.