Shark Cage: Protecting User Interfaces Against Malware

Author names
Institution
Address
Email

ABSTRACT

Applications need to interface with human users and need to be able to fend off simulated input or eavesdropping by malicious software. We present an analysis of requirements and existing implementations of the trusted path concept in operating systems, transfer the knowledge to a trusted path for individual applications, and demonstrate how it can be implemented for programs running atop the Microsoft Windows operating system. Despite recent growth in the market share of mobile devices, personal computers and the Windows operating system are still used by hundreds of millions of people. Our approach makes use of the APIs available in Windows and does not depend upon adding costly hardware or creating new operating systems.

1. INTRODUCTION

Applications need to communicate with human users and need to be able to fend off simulated input or eavesdropping by malicious software. Despite recent growth in the market share of mobile devices, personal computers are still used by hundreds of millions of people, and most of these machines run the Microsoft Windows operating system. Existing approaches to implement a trusted path for applications often make use of additional hardware or command the introduction of new operating systems. From a technical perspective, the approaches are usually sound. None of the approaches has really succeeded from an economic perspective, owing to cost of hardware, limited usability, or the effort of making the existing investment in applications compatible with new operating systems. The use of mobile phones as a trusted device has been demonstrated to be insufficient, given the observation of two-factor malware combining malware on a personal computer with a malicious app on a mobile phone to trick the user into revealing secret information, cf. [11].

The research questions that we address are as follows: "Given an adversary that is able to execute unprivileged (i.e. non-administrative) code on Microsoft Windows, can a trusted path between the user and a benign application in the same operating system session be established? If so, how? Can the trusted path be achieved at reasonable costs, i.e., without adding hardware?"

Our contribution is to provide a trusted path for individual applications atop the widely used Microsoft Windows operating system, safeguarding investment in existing technology and imposing few limitations on the usability of applications. We call our tool "Shark Cage", because it works similar like a shark cage used in diving: security-sensitive applications run in the same ocean as malware, but are protected from attacks.

In this article, we first scope the applications that require a trusted path to a human user in section 2 and outline the technical background for our work in section 3. We derive a security model for trusted path implementations in section 4. Our design decisions and implementation atop the Microsoft Windows operating system are demonstrated in section 5. We discuss advantages and limitations in section 6 and present previous approaches to implement a trusted path in section 7 before we conclude in section 8.

2. APPLICATION AREAS

There are several situations where an application needs to communicate directly with a human user in the presence of malware on the client device. A trusted path provides integrity, authenticity, and confidentiality.

The concept of a trusted path is at least as old as the TCSEC ([7]; also known as the *Orange Book*). A *Trusted Path* is defined in the TCSEC as:

> Trusted Path – 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 more recent Common Criteria [6] define a trusted path in "Class FTP: Trusted path/channels" like this:

> The TSF (=the target's of evaluation security functions) shall provide a communication path between itself and users that is logically distinct from other communication paths and provides assured identification of its end points and protection of the communicated data.

In the annex, the importance is further emphasised:

> A trusted path provides confidence that a user is communicating directly with the TSF (=the

ACSAC '13 New Orleans, Louisiana USA

> target's of evaluation security functions) whenever it is invoked. ... Absence of a trusted path may allow breaches of accountability or access control in environments where untrusted applications are used. These applications can intercept user-private information, such as passwords, and use it to impersonate other users. As a consequence, responsibility for any system actions cannot be reliably assigned to an accountable entity. Also, these applications could output erroneous information on an unsuspecting user's display, resulting in subsequent user actions that may be erroneous and may lead to a security breach.

Use cases where **confidentiality** of displayed information against malware/spyware is desired comprise bank statements (e.g. in browser banking), health records, sensitive communication or pictures. Since the UIs of applications are not isolated from each other, it is possible for processes to retrieve information that is displayed by other processes, either using an API like *window messages*, by taking a screenshot to see what the user sees, or even by injecting code into other processes. Confidentiality with respect to data entered by a user is relevant when credentials, e.g. PINs, passwords or credit card numbers, are typed that could be re-used by perpetrators on a later occasion. Other processes can retrive typed data by using an API like *window messages*, by hooking into keyboard events, or by querying keyboard state. None of these methods require administrative privileges to be obtained by an adversary. Operating system logons are usually protected by a trusted path while data entered into applications is not.

Integrity needs to be preserved of data that is displayed on which a user bases a decision. Use cases include bank transactions (e.g. in browser banking), verification of electronically signed documents, or state information in building automation systems.

Some situations require **authenticity** of input, e.g. when actions are to be confirmed by a user and the application needs to ensure that input is not controlled by malware. Use cases where this applies involve actions performed by a security token like a smart card on behalf of a user, CAPTCHA-free authentication, confirmation of bank transactions or electronic signatures, elevation of processes similar to the UAC User Account Control feature of Microsoft Windows, or any situation where a MITM man-in-the-middle attack on the communication between the user and an application would be security-relevant. Processes can simulate input using APIs like *SendInput()*, by sending *window messages*, or by employing a `WH_JOURNALPLAYBACK` hook.

3. TECHNICAL BACKGROUND

3.1 Windows UI resource access

Microsoft Windows limits user input/output to a single desktop at a time. A desktop is a securable object contained within a window station; it has a logical display surface. A window station is a securable object that is associated with a process, and contains one or more desktop objects. Threads can be connected to a desktop (they usually are) and they can connect to a different desktop while running, but they cannot be connected to several desktops at the same time. Threads can only use hooks to interfere with user input in the desktop they are connected to. If malware/spyware runs in the active desktop, it can receive user input and can capture output to the user; if it runs in a different desktop than the active one, it cannot. Desktops are create using *CreateDesktop()* and are switched using *SwitchDesktop()* [25].

Desktop managers use additional desktops to provide different views on executing applications or to create a kiosk-like mode. They sometimes provide protection against malware by this approach, if the malware is not designed to work with different desktops. The access control list (ACL) of the additional desktops typically does not prohibit malicious threads from connecting to another desktop, like in [32, 33].

In Windows 8, the *Desktop Window Manager* (DWM, cf. [26]) composes the display that the users sees from several sources: application windows shown on the (traditional) desktop accessed by GDI, and Windows Store ("metro-style") apps. The DWM ensures that apps cannot overlay other Windows Store app's UI, neither in fullscreen mode nor in side-by-side display in snapped mode. Applications on the GDI desktop can still access the contents of the desktop and of windows owned by other applications, but the display of Windows Store apps is sandboxed from other apps and from desktop applications, cf. [27, p. 79]. The Desktop Window Manager only affects the primary Windows desktop (`Default`). While integrity of the display is preserved, confidentiality of output might be breached when a Windows Store app and the GDI desktop are shown side by side in snapped mode, and a screenshot of the whole display is taken and stored in the clipboard where it is accessible to all processes running in the same window station. Input is not affected by the Desktop Window Manager, and, hence, neither confidentiality nor integrity of user input can be guaranteed as malware can still use hooks that affect input processing for all applications on the same Windows desktop.

Exclusive access to the screen can also be achieved by DirectX fullscreen applications, but it does not prevent malware from logging keystrokes [18].

3.2 Winlogon desktop and UAC

The Winlogon desktop is used to collect the user's user name and password, and is also used by UAC (User Account Control, cf. [24]) for confirmation of process elevation. The access control list (ACL) for the `WinSta0\Winlogon` desktop of the interactive user session (owned by the group `BUILTIN\Administrators`) is "so that only Winlogon can access that desktop" ([34]). This was verified by a call to `GetSecurityDescriptorDacl()` and documented in table 1. The ACL prevents non-system and non-administrative processes from accessing the Winlogon desktop.

The Default desktop is used for all user processes unless a different desktop is specified when creating a new process. The ACL for the `WinSta0\Default` desktop of the interactive user session (owned by the group `BUILTIN\Administrators`) is identical for the `SYSTEM` SID. The permissions to change owner and ACL are not included for the `Administrators` group, but `GENERIC_READ`, `GENERIC_EXECUTE`, and `GENERIC_WRITE` are added without the three access rights related to hooks. reading and modifying window objects on the desktop are added to the access control entry. The ACL also allows `GENERIC_ALL` access for *Restricted Code* and for all processes having the logon session SID in their token. The full ACL

Subject	Access mask
BUILTIN\Administrators (SID: S-1-5-32-544)	_DELETE DESKTOP_ENUMERATE READ_CONTROL WRITE_DAC WRITE_OWNER
NT AUTHORITY\SYSTEM (SID: S-1-5-18)	_DELETE DESKTOP_READOBJECTS DESKTOP_CREATEWINDOW DESKTOP_CREATEMENU DESKTOP_HOOKCONTROL DESKTOP_JOURNALRECORD DESKTOP_JOURNALPLAYBACK DESKTOP_ENUMERATE DESKTOP_WRITEOBJECTS DESKTOP_SWITCHDESKTOP READ_CONTROL WRITE_DAC WRITE_OWNER

Table 1: ACL for the WinSta0\Winlogon desktop

is documented in table 2; additional entries compared to WinSta0\Winlogon are printed in *italics*.

3.3 User notification and awareness

Window personalisation has been proposed, among others, by [42], showing a user-chosen logo as a means of authenticating a server to a user based on a shared visually-recognisable secret. That concept has been developed further into personal images as window backgrounds by [8]. In the commercial world, MasterCard's SecureCode [21] uses a "Personal Greeting" as a shared secret between the credit card company and an account holder. Coloured window frames controlled by a window manager are a popular choice to convey information about the trust level of a window, as done, e.g. by extensions to browsers [44], the window manager in Solaris [14], or the Qubes OS [17]. In text-based UIs, the Perseus micro-kernel reserves the topmost line of the screen for a visual indication of the OS/application having console access [30]. Web browsers show a green address bar when the page that is displayed stems from a server authenticated by an "Extended Validation" SSL certificate [4].

4. SECURITY MODEL

In the following explanation, we will use the term *user's computer* for the computer that the physically present user uses, e.g., a personal computer, and we will use the term *dedicated device* or *device* that implements a trusted path as shown e.g. in figure 4.

4.1 Adversary model

The adversary that we address is malware/spyware that is executed with ordinary user privileges, i.e. does not require privileges of members of the *Administrator* group to function. This is a releveant threat, as we see e.g. with the SpyEye trojan [38]. Operating system vendors have made it increasingly harder to exploit weaknesses in an operating system's design and implementation, so malware has adapted to not requiring privileges it is not able to obtain.

Since subversion of the operating system is out of scope for malware without administrative privileges, the targeted

Subject	Access mask
NT AUTHORITY\RESTRICTED_CODE (SID: S-1-5-12) and Session SID	_DELETE DESKTOP_READOBJECTS DESKTOP_CREATEWINDOW DESKTOP_CREATEMENU DESKTOP_HOOKCONTROL DESKTOP_JOURNALRECORD DESKTOP_JOURNALPLAYBACK DESKTOP_ENUMERATE DESKTOP_WRITEOBJECTS DESKTOP_SWITCHDESKTOP READ_CONTROL WRITE_DAC WRITE_OWNER
BUILTIN\Administrators (SID: S-1-5-32-544)	DESKTOP_ENUMERATE READ_CONTROL *DESKTOP_READOBJECTS* *DESKTOP_CREATEWINDOW* *DESKTOP_CREATEMENU* *DESKTOP_WRITEOBJECTS* *DESKTOP_SWITCHDESKTOP*
NT AUTHORITY\SYSTEM (SID: S-1-5-18)	_DELETE DESKTOP_READOBJECTS DESKTOP_CREATEWINDOW DESKTOP_CREATEMENU DESKTOP_HOOKCONTROL DESKTOP_JOURNALRECORD DESKTOP_JOURNALPLAYBACK DESKTOP_ENUMERATE DESKTOP_WRITEOBJECTS DESKTOP_SWITCHDESKTOP READ_CONTROL WRITE_DAC WRITE_OWNER

Table 2: ACL for the WinSta0\Default desktop

assets are applications executed by the user, data stored in these applications, and data or services accessed by way of these applications.

We first take a look how properties of a trusted path translate to requirements for hardware, and do then derive requirements we pose to software implementations of a trusted path for applications.

4.2 Properties of hardware solutions

Trusted path implementations in dedicated hardware enforce the four properties derived from [7, 6] and mentioned in section 2:

1. TCSEC: Trusted Computing Base/CC: TSF: The code stored and executed on the dedicated device is controlled by its manufacturer and never updated or updated only in a controlled manner. Authenticity and integrity of firmware updates are secured by cryptographic means.

2. TCSEC: Direct communication/CC: Protection of communicated data: Only processes on the device have access to the device's input and display resources. Input is not transmitted to the user's computer, and display cannot be modified from processes on the user's computer.

3. TCSEC: Controlled activation/CC: Logically distinct: The user turns attention from the user's computer to the dedicated device.

4. TCSEC: No imitation/CC: Identification of end points: Only trusted software runs on the device and software on the user's computer cannot create or modify hardware.

4.3 Requirements for software solutions

The properties of a trusted path as defined by the TCSEC/CC and as found in existing secure hardware implementations need to be achieved by software-only techniques. Translated to software, the requirements are as follows.

1. TCSEC: Trusted Computing Base/CC: TSF: The code used to implement the trusted path tool must be controlled by its manufacturer and never be updated or be updated only in a controlled manner. Authenticity and integrity of tool software updates are secured by cryptographic means.

2. TCSEC: Direct communication/CC: Protection of communicated data: Only the processes selected for the trusted path have access to input and display resources of the user's computer during the lifetime of the protected process. Input is not transmitted to other processes on the user's computer, and the display cannot be modified by other processes on the user's computer.

3. TCSEC: Controlled activation/CC: Logically distinct: The trusted path tool can establish exclusive interaction mode with the user.

4. TCSEC: No imitation/CC: Identification of end points: A protected process using the trusted path appears in a way that the user recognizes and that cannot be forged by other processes on the user's computer.

5. DESIGN AND IMPLEMENTATION

The trusted path tool – Shark Cage – combines several existing concepts to an innovative implementation of a hardware-less, inexpensive, and legacy-compatible trusted path for applications.

5.1 Design choices

5.1.1 Trusted Computing Base/TSF

Secure software engineering will not be discussed here in detail, since the use of best practices is seen as sufficient.

5.1.2 Direct communication/Protection of communicated data

Exclusive communication between the user and a protected process is achieved by using a separate desktop with a modified access control list (ACL), allowing only processes with the capability to use the trusted path to access that desktop. The capability is simulated by creating a group at run-time with a unique security identifier that has not been in use before. The ACL is limited to entries containing this security identifier. Protected processes are created with a modified token that includes the security identifier; only system processes have the ability to modify a token in that manner, preventing malicious or accidental interference with the UI of the trusted path.

5.1.3 Controlled activation/Logically distinct

Only processes that have the aforementioned capability to access the trusted path desktop can switch to the trusted path desktop. This limits activation of the trusted path desktop to modules of the Shark Cage tool. Non-protected processes need to explicitly request activation of the trusted path by engaging with the Shark Cage tool. There is no method for the user to start communicating via the trusted path. The trusted path is an offer to applications that need to determine when it is applicable to switch away from the traditional shared desktop and to the trusted path UI.

5.1.4 No imitation/Identification of end points

Part of the screen is reserved to display a shared visual secret. That secret is a picture captured by the integrated camera of the personal computer, showing the user and a gadget unique to the application associated with the picture. In case of a banking application, e.g. the user takes a picture with himself and a bank card as shown in figure 1. Access to the picture file is limited to the Shark Cage tool processes and to the protected application itself. It is only shown while the trusted path is activated. If the user sees the picture, he knows that the trusted path is activated. If the user does not see the picture or sees a different picture, he knows that the trusted path is not working properly and can hence abort the use of the application.

5.2 Implementation

The *Shark Cage* tool is composed of four modules: `Cage Service`, `Cage Manager`, `Cage Labeller`, and `Cage Admin`. The Cage Admin is used to configure the Shark Cage tool, selecting the applications that have access to the trusted path, and recording a shared secret between the user and an application. The other three modules are used at runtime to create exclusive access to UI resources and to enable caged

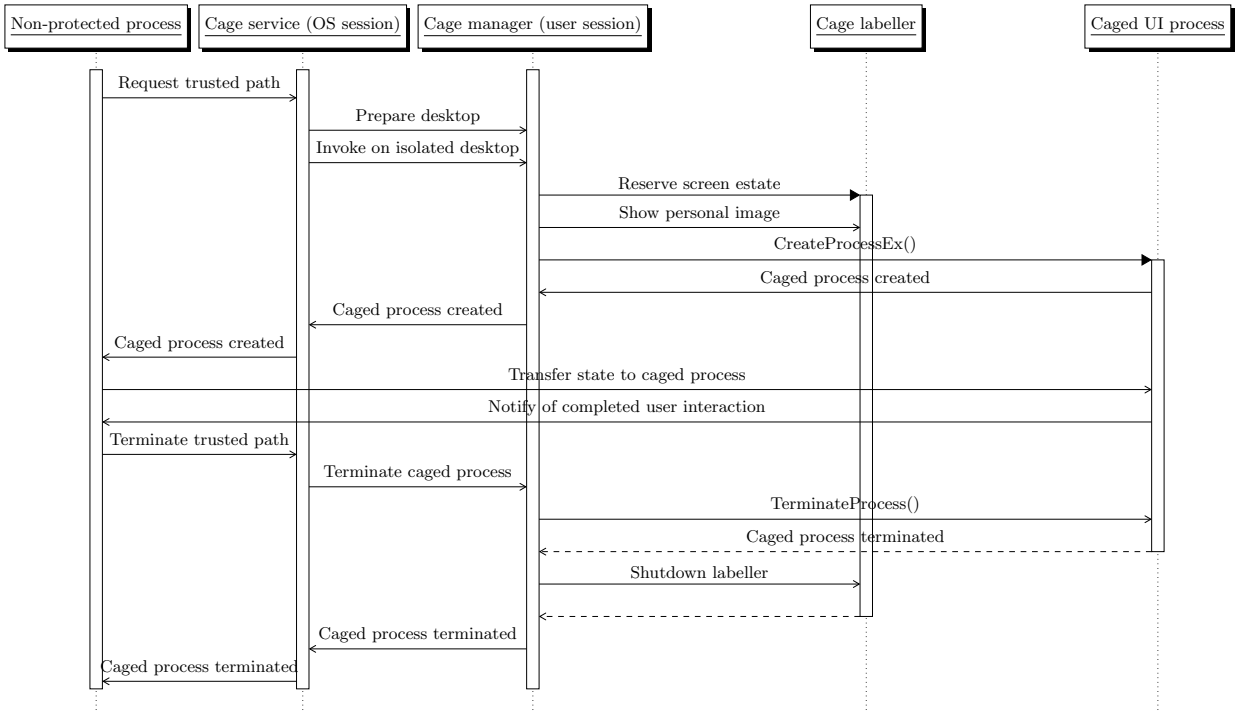

Figure 2: Control flow between *Shark Cage* tool modules

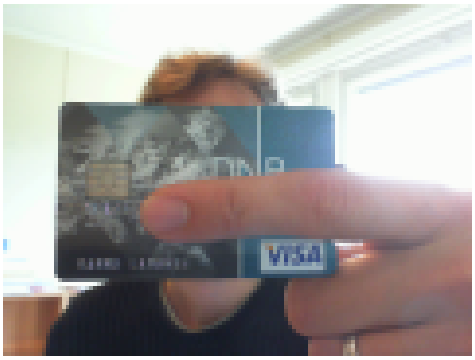

Figure 1: Sample application logo used for online banking

applications to use the trusted path. The usual sequence in which the modules are invoked is shown in figure 2.

A non-protected process sends a SOAP request to the Cage Service; it does not make a difference whether the request is sent from a locally installed application or from an applet in a web browser. Depending on firewall rules, the Cage Service might also be invoked remotely (we currently do not allow that, but there might be scenarios where this is desirable). The Cage Service runs in session 0 where all services are run by default. The request to the Cage Service contains the identifier of the process that is requested to be invoked with a trusted path UI. All eligible processes must be configured once in advance. The Cage Service then creates a Cage Manager process in the session of the requesting process. This is necessary, because creation of desktops in a session is only possible for processes in the same session, and the session (of the Cage Service) cannot be changed during run-time.

The Cage Manager creates a new group with no members and an unused unique security identifier (SID). It then creates a new `CagedUI` desktop with an ACL restricting access to system processes and those processes having a token with a security identifier of the newly created group, cf. table 3. (Alternatively, the Cage Manager reuses an existing `CagedUI` desktop by modifying the ACL and terminating all processes attached to that desktop.) The ACL for the `WinSta0\CageUI` desktop (owned by the `BUILTIN\Administrators` group withing the interactive user session) is similar to the ACL of the `WinSta0\Winlogon` desktop used by the operating system for credential entry.

The Cage Manager creates a Cage Labeller process on the CagedUI desktop. The task of the Cage Labeller process is to show the pre-configured display name of the caged application and the pre-configured application logo as shown in figure 3. It also reduces the work area of the desktop so that a caged application can maximise its window without eclipsing the display name and the logo drawn by the Cage Labeller. The application logo can be a picture taken with the integrated webcam of a laptop. This ensures that it is harder to guess and easier to remember than choosing a logo from a limited set.

On the prepared desktop the Cage Manager creates a process of the requested application. This process is created with a modified token that includes the unique group SID listed in the ACL of the CagedUI desktop. The caller of the Cage Service is informed about creation of the trusted path for the process created in the “cage”. The Cage Manager switches the active desktop from the current desktop to the CageUI desktop.

Subject	Access mask
BUILTIN\Administrators (SID: S-1-5-32-544)	_DELETE DESKTOP_ENUMERATE READ_CONTROL WRITE_DAC WRITE_OWNER
NT AUTHORITY\SYSTEM (SID: S-1-5-18)	_DELETE DESKTOP_READOBJECTS DESKTOP_CREATEWINDOW DESKTOP_CREATEMENU DESKTOP_HOOKCONTROL DESKTOP_JOURNALRECORD DESKTOP_JOURNALPLAYBACK DESKTOP_ENUMERATE DESKTOP_WRITEOBJECTS DESKTOP_SWITCHDESKTOP READ_CONTROL WRITE_DAC WRITE_OWNER
<host>\CagedUI-<x> where <x> is a random number (SID: S-1-5-21-<dom1>-<dom2>-<dom3>-<RID>)	DESKTOP_READOBJECTS DESKTOP_CREATEWINDOW DESKTOP_CREATEMENU DESKTOP_HOOKCONTROL DESKTOP_JOURNALRECORD DESKTOP_JOURNALPLAYBACK DESKTOP_ENUMERATE DESKTOP_WRITEOBJECTS

Table 3: ACL for the WinSta0\CagedUI desktop

The non-protected process can choose to open an inter-process communication channel with the caged process to e.g. transfer state information. The caged process performs its interaction with the local human user, undisturbed from malware/spyware. Only the active desktop receives user input and conveys visual output to the user. Hence, malicious processes attached to other desktops cannot interfere with user input/output. Trying to attach to the CageUI desktop requires a compatible token. Non-administrative processes cannot acquire membership in the unique group and cannot arbitrarily modify tokens, excluding them from the active desktop.

When the need for the trusted path no longer exists, the caged process notifies the non-protected process. The non-protected process then informs the Cage Service that the trusted path is no longer necessary. As a consequence, the Cage Manager terminates the caged process and the Cage Labeller and switches the active desktop back to the original desktop, usually "WinSta0\Default".

6. DISCUSSION

The research questions that we address are as follows: "Given an adversary that is able to execute unprivileged (i.e. non-administrative) code on Microsoft Windows, can a trusted path between the user and a benign application in the same operating system session be established? If so, how? Can the trusted path be achieved at reasonable costs, i.e., without adding hardware?"

We show that a trusted path can be implemented using only APIs provided by the Windows operating system. The main method is the creation of a separate desktop protected by an ACL. The ACL is similar to that of the Winlogon desktop used by UAC. Our ACL differs in that it contains an entry with a dynamically created unique group. Adding that group to the token of processes created on that separate desktop functions like a capability. Since only software is needed, development and deployment effort are much lower than required by alternative solutions like dedicated hardware or new operating systems or hypervisors. While the *clean slate* approach is seen attractive in the academic community, it has not yet succeeded in the marketplace and we need improvements in UI security at least in the transition period when new secure systems are not widely available.

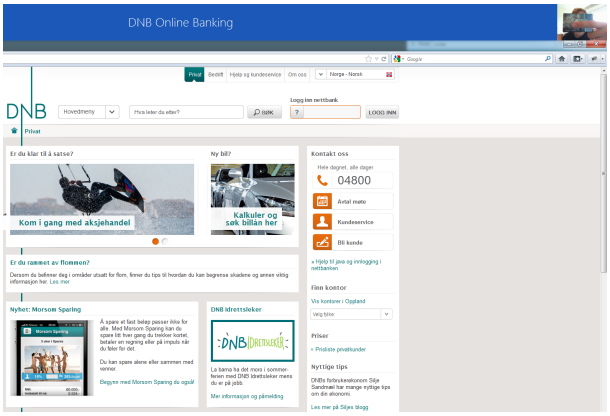

Figure 3: Caged online banking application with display name and logo at top of the screen

Applications can be re-used without any modification if they do not require that processes they interact with are connected to the same desktop. In case that only part of the functionality is to be connected to the desktop representing the trusted path, the modification effort is similar to adapting applications to use UAC for elevated portions of the code.

An issue with transferring applications to a separate desktop is that applications might be composed of modules not under control of the application vendor. These modules would be able to interfere with the UI if they were not trustworthy. Examples include web browser plug-ins. It might be hard to disable plug-ins completely, like in the case of plug-ins assisting users with disabilities. Exploiting possible vulnerabilities in a caged process is confined to capabilities of that process and does not affect the operating system, the *Shark Cage* tool or a trusted path established for other processes.

We did not subject the *Shark Cage* tool to a comprehensive user test. Hence, we do not know whether users would observe when control was switched back to the default (untrusted) desktop.

7. PREVIOUS AND RELATED WORK

7.1 Linux UI resource access

Our implementation is based on APIs for the Microsoft Windows operating system. Linux organises access to input and output resources differently. A Linux kernel driver that allows user land processes to simulate input is Uinput [20]. Managing framebuffer access in Linux is discussed by [12].

In OS X, secure text entry can be activated by applications calling `EnableSecureEventInput()` [2].

7.2 Window managers

Labelling windows has been proposed and implemented for several window managers, e.g. Solaris Trusted Extensions [28] and the EROS trusted window system [36]. An early approach is discussed in [9].

7.3 Custom operating systems

Several new operating systems have been created that offer a trusted path feature or offer the ability to isolate the user interfaces of different applications from each other, e.g. Qubes [17] and Turaya [5]. Modifications of existing operating systems include Apiary [31] andWindowBox [3] which works similar to *Shark Cage*, but does not offer visual indication of active compartments and requires more complex advance configuration. A mobile OS modification to support separate working environments on a mobile phone is demonstrated by [35].

7.4 Sandboxing

An overview of sandboxing techniques is provided by [29]. Operating systems for mobile devices typically allow only a single app at a time to gain access to the main user interface and they restrict inter-process communication and access to other processes.

Adobe Reader [1] and Google Chrome browser [40] use an internal sandbox to render data from untrusted sources. The techniques used are similar to what we use for *Shark Cage*: a desktop with a modified ACL. The difference is that these applications confine potentially malicious content, while *Shark Cage* creates a desktop to keep potentially malicious processes out.

7.5 Trusted computing technology

Trusted computing technology adds a trusted platform module (TPM, [41]) to a device. The TPM can perform measurements of the integrity of a platform, store measurement results, and can report measurement results on request. Modification of software used in a boot process, e.g. boot loader, operating system kernel etc. can be detected and applications could decide not to perform security-relevant functionality in a modified environment. Static integrity measurements of a platform might not be sufficient to detect malware – if the underlying platform is unmodified, it might still be vulnerable. Trusted computing usually only assures authenticated boot (i.e. a certain set of software has been executed), it does not assure secure boot (i.e. only trusted/trustworthy software has been executed, i.e. using a whitelist) [15].

Intel Trusted Execution Technology (TXT, cf. [16]) comprises hardware extensions to CPUs and chipsets. It aims to protect against software-based attacks, shielding execution and memory so that sensitive data can be processed without being manipulated by untrusted software. Sealed storage limits access to sensitive data to the same hardware/software configuration with which the data was stored, and attestation reports whether a system correctly invoked TXT. One of the use models is "local verification" that uses the measurement capability of TXT to allow the user to have confidence that the platform is executing in a known state. TXT's hardware ability measures the launched configuration and stores the measurement in the platform's TPM (Trusted Platform Module). User input can be protected between input devices (keyboard/mouse) and an input manager shielded by TXT. This requires the input devices to encrypt the input in compliance with TXT. Output that is written to the framebuffer can be kept confidential from other processes. An implementation of a trusted display using the small Flicker TCB ([22]) in combination with Intel TXT is presented in [13].

The concept of using trusted computing technology to establish integrity and authenticity of a hypervisor and a device driver used to actually implement a trusted path has also been written down in a patent application [23].

7.6 Dedicated hardware

In many cases where software is perceived as being hard to protect, dedicated devices are proposed as a solution. Since the code on these devices cannot be changed (or can only be updated in a controlled way), and since the issuer of the device controls which code is run on the device, malware/spyware cannot be executed on dedicated devices. Hardware approaches include smart card readers with a PIN pad and a display, an example of a so-called class 3 card reader is shown in figure 4. The SICCT specification (Secure Interoperable ChipCard Terminal, [39]) describes smart card terminals with input/output functionality. The mode of operation is signalled to the user, i.e., the user is informed when input/output is transferred directly to/from the smart card and is not made available to a possibly untrusted environment beyond the card terminal. Moving away from class 1 readers (simple contact unit without UI) and from class 2 readers (card reader with PIN pad) incurs a loss of information, because the PIN is no longer entered in the application, and, hence, the application can no longer prove to the smart card that the user used a specific application to communicate with the card. So-called class 4 card readers have the ability to execute (some) applications on the device; this approach has recently surfaced in the FINREAD standard [10], success has yet to happen in the marketplace.

Figure 4: Hardware implementation of a trusted path: Class 3 smart card reader

At the time of introduction, mobile phones were considered to be trusted devices. Today, mobile phones can execute arbitrary applications, including malware [11]. Other approaches to provide limited UI capabilities with additional dedicated hardware include ZTIC [43], Neb [19] and sCrib [37]. All hardware solutions suffer from similar problems: they

are more expensive than software, they require more effort to deploy to users, and they lack flexibility and the rich user experience of the device that they complement.

8. CONCLUSIONS

Our contribution is to provide a trusted path for individual applications atop the widely used Microsoft Windows operating system. We show that it is possible to protect the UI of applications against malware/spyware running without administrative privileges. We achieve the protection by implementing a trusted path using only APIs of the operating system. Our implementation combines several earlier ideas into a practical, usable, and cost-effective solution. To our knowledge, this is the only implementation for commercial off-the-shelf software without changes to the operating system or application and without the introduction of hardware.

Future work should address compatibility with terminal server and with the RDP remote desktop protocol, applicability of our approach to mobile devices like smartphones and tablets, and evaluation of usability with different user groups and different application scenarios.

9. REFERENCES

[1] Adobe. Inside Adobe Reader Protected Mode – Part 1 – Design. `http://blogs.adobe.com/asset/2010/10/inside-adobe-reader-protected-mode-part-1-design.html`, 2010.

[2] Apple. Carbon Event Manager Reference (Legacy). `http://developer.apple.com/legacy/library/documentation/Carbon/reference/Carbon_Event_Manager_Ref/Carbon_Event_Manager_Ref.pdf`, 2013.

[3] D. Balfanz and D. R. Simon. WindowBox: a simple security model for the connected desktop. In *Proceedings of the 4th conference on USENIX Windows Systems Symposium - Volume 4*, WSS'00, pages 4–15, Berkeley, CA, USA, 2000. USENIX Association.

[4] CA/Browser Forum. About EV SSL Certificates. `https://www.cabforum.org/certificates.html`, 2013.

[5] L. Catuogno, A. Dmitrienko, K. Eriksson, D. Kuhlmann, G. Ramunno, A.-R. Sadeghi, S. Schulz, M. Schunter, M. Winandy, and J. Zhan. Trusted virtual domains – design, implementation and lessons learned. In *Proceedings of the First international conference on Trusted Systems*, INTRUST'09, pages 156–179, Berlin, Heidelberg, 2010. Springer-Verlag.

[6] CCMB. Common Criteria for Information Technology Security Evaluation, Part 2: Security functional components, Version 3.1, Revision 4. `http://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R4.pdf`, 2012.

[7] Department of Defense. *TCSEC: DoD 5200.28-STD Department of Defense Trusted Computer System Evaluation Criteria*. Department of Defense, 1985.

[8] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic Security Skins. In *Proceedings of the 2005 symposium on Usable privacy and security*, SOUPS '05, pages 77–88, New York, NY, USA, 2005. ACM.

[9] J. Epstein. Fifteen Years after TX: A Look Back at High Assurance Multi-Level Secure Windowing. *Computer Security Applications Conference, Annual*, 0:301–320, 2006.

[10] European Committee for Standardization. CWA 14174 Financial transactional IC card reader (FINREAD). `http://www.cen.eu/cen/Sectors/Sectors/ISSS/CWAdownload/Pages/FINREAD.aspx`, 2004.

[11] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, pages 3–14, New York, NY, USA, 2011. ACM.

[12] N. Feske and C. Helmuth. A Nitpicker's guide to a minimal-complexity secure GUI. In *ACSAC*, pages 85–94, 2005.

[13] A. Filyanov, J. M. McCune, A.-R. Sadeghi, and M. Winandy. Uni-directional trusted path: Transaction confirmation on just one device. In *DSN*, pages 1–12, 2011.

[14] S. Gajek, A.-R. Sadeghi, C. Stüble, and M. Winandy. Towards Multicolored Computing – Compartmented Security to Prevent Phishing Attacks. In *1st Benelux Workshop on Information and System Security (WISSec 2006), Antwerpen (Belgium)*, 2006.

[15] J. H. Huh, J. Lyle, C. Namiluko, and A. Martin. Managing application whitelists in trusted distributed systems. *Future Gener. Comput. Syst.*, 27(2):211–226, Feb. 2011.

[16] Intel Corporation. Intel Trusted Execution Technology (Intel TXT) – Software Development Guide. `http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-trusted-execution-technology-software-dev-guide.pdf`, 2011.

[17] Invisible Things Lab. Select Qubes OS Screenshots. `http://qubes-os.org/trac/wiki/QubesScreenshots`, 2013.

[18] H. Langweg. With Gaming Technology towards Secure User Interfaces. In *ACSAC*, pages 44–50, 2002.

[19] B. Laurie and A. Singer. Choose the red pill and the blue pill: a position paper. In *Proceedings of the 2008 workshop on New security paradigms*, NSPW '08, pages 127–133, New York, NY, USA, 2008. ACM.

[20] Linux. Uinput. `http://git.kernel.org/cgit/linux/kernel/git/next/linux-next.git/tree/include/linux/uinput.h`, 2002.

[21] MasterCard. Support SecureCode FAQs. `http://www.mastercard.us/support/securecode.html`, 2013.

[22] J. M. McCune. Flicker: Minimal TCB Code Execution – Version 0.5. `http://sourceforge.net/p/flickertcb/wiki/Home/`, 2011.

[23] J. M. Mccune, A. M. Perrig, A. Datta, V. D. Gligor, and N. Qu. Patent application: Methods and Apparatuses for User-Verifiable Trusted Path in the Presence of Malware. Patent application number 20120198514 (class: Information security policy), 2012.

[24] Microsoft. User Account Control: Behavior of the elevation prompt for administrators in Admin

Approval Mode. http://technet.microsoft.com/en-us/library/dd835564%28WS.10%29.aspx#BKMK_AdminPromptBehavior, 2009.

[25] Microsoft. About Window Stations and Desktops. http://msdn.microsoft.com/en-us/library/windows/desktop/ms681928.aspx, 2012.

[26] Microsoft. Desktop Window Manager (Windows). http://msdn.microsoft.com/en-us/library/windows/desktop/aa969540.aspx, 2012.

[27] Microsoft. Windows 8 and Windows Server 2012 Compatibility Cookbook. http://www.microsoft.com/en-us/download/details.aspx?id=27416, 2012.

[28] Oracle. Solaris Trusted Extensions User's Guide – Chapter 4 Elements of Trusted Extensions (Reference). http://docs.oracle.com/cd/E19082-01/819-7313/6n993dclc/index.html, 2010.

[29] D. S. Peterson, M. Bishop, and R. Pandey. A Flexible Containment Mechanism for Executing Untrusted Code. In *Proceedings of the 11th USENIX Security Symposium*, pages 207–225, Berkeley, CA, USA, 2002. USENIX Association.

[30] B. Pfitzmann, J. Riordan, C. Stüble, M. Waidner, and A. Weber. The PERSEUS System Architecture. Technical Report RZ 3335 (#93381), IBM Research, 2001.

[31] S. Potter and J. Nieh. Apiary: easy-to-use desktop application fault containment on commodity operating systems. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 8–21, Berkeley, CA, USA, 2010. USENIX Association.

[32] D. Reichl. KeePass: Enter Master Key on Secure Desktop (Protection against Keyloggers). http://keepass.info/help/base/security.html#secdesktop, 2013.

[33] M. Russinovich and B. Cogswell. Desktops v2.0. http://technet.microsoft.com/en-us/sysinternals/cc817881, 2012.

[34] M. Russinovich, D. A. Solomon, and A. Ionescu. *Windows Internals, Sixth Edition, Part 1*. Microsoft Press, 2012.

[35] M. Selhorst, C. Stüble, F. Feldmann, and U. Gnaida. Towards a trusted mobile desktop. In *Proceedings of the 3rd international conference on Trust and trustworthy computing*, TRUST'10, pages 78–94, Berlin, Heidelberg, 2010. Springer-Verlag.

[36] J. S. Shapiro, J. Vanderburgh, E. Northup, and D. Chizmadia. Design of the EROS trusted window system. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 12–12, Berkeley, CA, USA, 2004. USENIX Association.

[37] Smart Crib Ltd. Password sCrib. http://www.s-crib.com, 2012.

[38] A. K. Sood, R. J. Enbody, and R. Bansal. Dissecting SpyEye – Understanding the design of third generation botnets. *Comput. Netw.*, 57(2):436–450, Feb. 2013.

[39] TeleTrust. SICCT-Spezifikation V-1.21 vom 17.12.2010. http://www.teletrust.de/uploads/media/SICCT-Spezifikation-1.21.pdf, 2010.

[40] The Chromuim Projects. Sandbox. http://www.chromium.org/developers/design-documents/sandbox#TOC-The-alternate-desktop, 2013.

[41] Trusted Computing Group (TCG). TPM Main Specification Level 2 Version 1.2, Revision 116. https://www.trustedcomputinggroup.org/resources/tpm_main_specification, 2011.

[42] J. D. Tygar and A. Whitten. WWW electronic commerce and java trojan horses. In *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, WOEC'96, pages 243–250, Berkeley, CA, USA, 1996. USENIX Association.

[43] T. Weigold, T. Kramp, R. Hermann, F. Höring, P. Buhler, and M. Baentsch. 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*, Trust '08, pages 75–91, Berlin, Heidelberg, 2008. Springer-Verlag.

[44] Z. E. Ye and S. Smith. Trusted Paths for Browsers. In *Proceedings of the 11th USENIX Security Symposium*, pages 263–279, Berkeley, CA, USA, 2002. USENIX Association.