RUHR-UNIVERSITÄT BOCHUM

Horst Görtz Institute for IT Security

Technical Report TR-HGI-2007-001

Compartmented Security for Browsers

Sebastian Gajek, Ahmad-Reza Sadeghi, Christian Stüble, and Marcel Winandy

Chair for System Security
Chair for Network and Data Security

Ruhr-Universität Bochum
Horst Görtz Institute for IT Security
D-44780 Bochum, Germany

TR-HGI-2007-001
March 30, 2007

Compartmented Security for Browsers

Sebastian Gajek, Ahmad-Reza Sadeghi, Christian Stüble, and Marcel Winandy
Horst Görtz Institute for IT Security
Ruhr-University Bochum
Universitätsstr. 150, D-44780 Bochum, Germany
sebastian.gajek@nds.rub.de, sadeghi@crypto.rub.de, stueble@acm.org, winandy@ieee.org

Abstract

Identity theft through phishing attacks has become a major concern for Internet users. Typically, phishing attacks aim at luring the user to a faked web site to disclose personal information. Various solutions have been proposed against this kind of attack. However, these solutions can hardly counter the new generation of sophisticated malware phishing attacks, e.g., pharming trojans, designed to target certain services.

This paper aims at making the first steps towards the design and implementation of an open source and interoperable security architecture that prevents both classical and malware phishing attacks. Our approach is based on the ideas of compartmentalization for separating applications domains of different trust level, and a trusted wallet for storing credentials and authenticating sensitive services. Once the wallet has been setup in an initial step, our solution requires no special care from users for identifying the right web sites while the disclosure of credentials is strictly controlled. Moreover, a prototype of the basic platform exists and we briefly describe its implementation.

1 Introduction

Identity theft has become a subject of great concern for Internet users in the recent years: Since password-based user authentication has established on the Internet to grant users access to security critical services, identity theft and fraud attracted attackers [39]. Hence, phishing—a colloquial abbreviation of *password fishing*—has become a prominent attack. Whereas *classical* phishing attacks primarily used spoofed emails to lure unwary users to faked web sites where they reveal personal information (e.g., passwords, credit card numbers, transaction numbers), current attacks have become advanced in their number and technical sophistication [2, 17, 21]. The new generation of phishing attacks does not solely address the weaknesses of careless Internet users, but also exploits vulnerabilities of the underlying computing platforms and takes advantage of legacy flaws of Internet technologies: *Hostile profiling* addresses specific email recipients to mount classical phishing attacks more precisely [10], *pharming* compromises DNS-Servers to resolve domain name requests to phishing sites [2], and *malware phishing* infiltrates customers' computers, e.g., to log their password stroking using special malicious programs [25].

The most dominant reason for the proliferation of phishing attacks is that strong assumptions and requirements are made on the ability of ordinary Internet users when accessing sensitive services (see, e.g., [19]). Internet users of average skill often do not understand security indicators and cannot distinguish between legitimate and faked web sites [32]. To reliably authenticate a web site, the user has to verify the domain name, 'https' in the URL, and the server's certificate. However, ordinary Internet users are unfamiliar with the meaning of SSL and DNS. This is in particular true for phishing victims, as most faked sites may have been exposed if users had properly checked for the presence of SSL channels.[1] On the other hand, the rise of malware phishing attacks indicates that common computing platforms lack of appropriate protection in practice. The problem with malware phishing attacks is that they are (i) specifically designed to target certain service providers (e.g., regional banks), (ii) exploit arbitrary operating system characteristics, and (iii) deploy tailored functionalities to obtain users' credentials [2, 16, 25]. It is straightforward for malware phishing attacks, e.g., to fake security indicators, imitate the browser's (or any security-critical application's) chrome or modify the system configuration, and thus to circumvent current phishing (and malware) countermeasures (see Section 4). More importantly, malware phishing attacks are not transparent to the user and hence raise less suspicion of identity theft than its classical variant.

Contribution In this paper, we make the first steps towards the design and implementation of a security architecture that counters phishing attacks while considering the usability aspects. We propose a modular platform that uses a trusted wallet to (i) store user's credentials and (ii) authenticate the sensitive services as a proxy on behalf of the user. Hence, it does not require specific skills from users, e.g., to distinguish between real and faked web sites by identifying security indicators. We discuss how to setup and update credentials that are to be stored in the wallet and how to solve problems that may arise when security-unaware users want to apply the same credentials to different services. But our contribution is much broader. In contrast to existing proposals our solution provides protection measures against the strongest type of phishing attacks, namely malware phishing. To establish a secure execution environment for the wallet and to be compatible to existing software applications, we show that a secure operating system can be efficiently realized by using virtualization technology and we justify why trusted computing functionality is needed.

Outline The remainder sections are structured as follows. In Section 2 we give an overview about phishing attacks and infer an attack taxonomy. Then, we summarize in Section 3 the requirements of a security architecture to prevent phishing attacks and discuss related work in Section 4. We describe our architecture in Section 5 and the details of the realization in Section 6. In Section 7 we show how such a system can be implemented. Finally, we conclude in Section 8.

2 Threat Model

2.1 Terms and Notations

Principals are parties involved in the phishing scenario. These are the user U who is interfaced to a computer system S and the service provider's system P. S is a collection of software components, such as the browsing application B. *Compartments* are isolated

[1]We analyzed several phishing sites and observed that none of them was triggered over SSL (cf. [14]).

logical components in S. We denote the phishing adversary as A and say that A uses a set of collection servers, such as a phishing site, to store and retrieve identities.

Channels are abstractions of communication paths. We distinguish between secure and insecure channels and denote a secure channel as a communication of two principals which is authentic, confidential, and of integrity. For example, $send_{U \to S}$ is the unilateral channel that U uses to send a message to S.

Identities are security sensitive information and are the targets of phishing attacks. We denote an identity $ID_{s_{id}}$ as the tuple (s_{id}, c_{id}, $attr_{id}$) where s_{id} indicates a set of unique service provider identifiers to authenticate P, c_{id} a set of credentials to get access to P, and $attr_{id}$ a set of attributes specific to user and service, such as age, address, or credit card number. The set of identifiers s_{id} are the URL and a server certificate (in case of SSL), which we abbreviate as the tuple (URL_{id}, $cert_{id}$). Credentials c_{id} establish the claim that U is in possession of $ID_{s_{id}}$ and are denoted as the tuple (u_{id}, pwd_{id}), whereas u_{id} and pwd_{id} are username and password.

2.2 Taxonomy of Phishing Attacks

The goal of phishing attacks is to obtain $ID_{s_{id}}$. Therefore, these attacks generally constitute two elementary stages [1]. First, a preliminary attack is launched to mount the actual illusion. Then, an illusion attack is launched to imitate, e.g., legitimate text, images, and windows to bluff the user. We briefly introduce phishing attacks in the following:

Classical phishing attacks have in common that the divulgement of $ID_{s_{id}}$ occurs on a remote phishing site. The phishing site imitates a legitimate service provider and occasionally masquerades security and connection identifiers of the browsing application (e.g., address bar). As mentioned in the introduction, classical phishing attacks presuppose that the user does not authenticate the malicious remote machine. To lure users to spoofed sites, phishing mails containing obfuscated links, cross site scripting (XSS) attacks, or DNS-based attacks are used to name a few. For more details, see [1].

Malware phishing attacks collect $ID_{s_{id}}$ on the client side. Some variants of malware phishing attacks alter the local host's files to resolve a false domain name and redirect users to faked sites. Advanced attacks capture the user's input when a targeted service is requested, or mimic original login sites. However, malware phishing attacks prerequisite that S has been corrupted; more precisely, that weaknesses of the software layer have been exploited. To infiltrate S, phishers anticipate the latency time of unfixed exploits [43] or attach malicious programs to phishing mails (see [2]).

Figure 1 shows the principles involved in the threat model related to the different phishing attacks. To conclude, phishing is a vector of attacks using different mounting, illusion and social engineering strategies. We say

Definition 1 (informal) *Phishing is the set of attacks that reveal a user's identity by establishing the illusion that the user communicates with the original (web) application.*

Note that phishing attacks exclude any physical attacks against the user or the hardware. In other words, we do not consider attacks where the adversary watches the user's key strokes to spy on private information or modifies the hardware invalidating the underlying security functionalities.

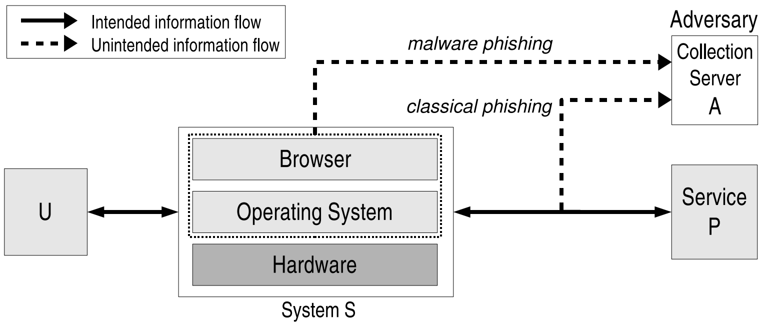


Figure 1: Threat model of phishing attacks.

2.3 Security Assumptions

Based on the diversity of current phishing attacks, we make the following assumptions:

Assumption 1 (Ordinary User) *Let U be an ordinary Internet user, i.e., potentially threatened by phishing attacks, then we assume that U is unable to properly authenticate P according to s_{id}.*

We already mentioned in the introduction that the user U has to verify the service identifiers s_{id} to reliably authenticate a web site of P, i.e., the domain name, 'https' in the URL, and the SSL server certificate. However, recent studies (see [19, 32]) point out that ordinary Internet users do not distinguish legitimate web sites from faked ones and do not understand indicators which signal trustworthiness.

Assumption 2 (Honest Provider) *Let P be a standard service provider, then we assume that P and its services are not corrupted.*

The service provider P fulfills all requirements to protect his services and enforces sound security policies; otherwise intruders were able to steal identities from the service provider's database. This is in particular true for certifying services. An adversary A may gain an original certificate $cert_{id}$ for a phishing site [15]. This is rather a problem of public key infrastructures and not the scope of the present work. Moreover, services are resilient against so called web spoofing attacks (see [11]), where the adversary A initially displays a completely faked Internet and is able to spoof any service. This is crucial because the user would disclose his identity while signing in to any service. Due to their more academic incentive, we neglect web spoofing attacks, and presume that the user U always performs the initial registration at the honest provider P.

Assumption 3 (Sound Browser) *Let B be a standard browsing application running on S, then we assume that the functionalities of B are implemented correctly.*

Browser developers are responsible for the soundness of their software and features (e.g., Javascript, Flash). Nevertheless, if the browser is vulnerable to, e.g., buffer overflow or format string attacks, then the user's system should safeguard that the intruder gains no more information than given in the application boundaries of the browser (see requirements below).

3 Requirements

The main motivation behind our architecture is to fulfill the following security objective:

Objective (Confidentiality of Credentials) *The system S approves that user U and service provider P are mutually authenticated and use a secure communication path.*

An adversary must not gain access to the user's credentials, i.e., credentials must only be given to authorized sites and authorized components of the operating system. The problem is that most web applications provide only entity authentication, i.e., the authentication is based on credentials and does not include all components in the communication path. This opens a gap for the communication of the user to the browser and services, respectively. Thus a service provider may not be able to verify if he is actually related to the claimed user. Analogously, the security-unaware user (recall Assumption 1) may not be able to ensure that he is authenticated to the claimed service.

To be able to provide the security objective, the system S has to fulfill the following requirements. In Section 6.1.5 we debate why only the fulfillment of all requirements protects against phishing attacks.

Requirement 1 (System Integrity) *The integrity of security-critical components in S should be preserved.*

The system is incapable to provide protection mechanisms and meet the other security requirements if its critical components are infected by malicious programs. Therefore, these components must be isolated from non-critical components. Moreover, there must be means to prevent offline attacks, e.g., when a different system is booted on the same hardware device. Otherwise the initial system components may be maliciously modified. Thus, a verification of the integrity at system startup is required (secure boot).

Requirement 2 (Isolation) *The code and data of applications in S have to be protected during runtime and when being stored persistently.*

Malicious processes must not be able to access the internal state or the persistently stored state of other processes. Malware attacks may try to exploit vulnerabilities of the computing platform in order to, e.g., log the user's key strokes or to modify the system configuration. Thus, applications of different tasks should be isolated, e.g., active scripts running in the web browser should not be able to access the credential store of the wallet. Where inter-process communication is necessary, only controlled communication interfaces should be possible.

Requirement 3 (Trusted Path) *The input and output of the application in S in which the user enters his credentials, must be protected from unauthorized access by other applications.*

For instance, emulating password input dialogs is a common attack of Trojan horse programs. Thus, the user must be sure about the integrity, authenticity, and confidentiality of the communication path to the application.

Requirement 4 (Robustness) *Security-critical components of S should be robust against wrong configuration or setup.*

In the context of phishing attacks, we consider average users who are not skilled. This holds especially for the configuration of security-critical applications. Thus, any configuration or setup that the user must perform and which are needed to fulfill the security objective must be easy to understand and robust against mistakes.

A further requirement would be that the system S should be compatible to commodity services that third parties provide. Although this is not a security-critical requirement, the issue is necessary to give the system a realistic chance for being deployed and commercially used. Therefore, we do not make any requirements concerning the service providers, i.e., the user's system should not presume adjustments and modifications of service infrastructures and software. Whenever changes to a system are demanded (e.g., attesting the client's system configuration), they should not require high costs for the provider and client.

4 Related Work

In this section, we discuss recent work on protection mechanisms against phishing attacks. Since executing a digital wallet for passwords on top of a secure operating system is a fundamental approach of our work, we also discuss related wallet-based solutions as well as design approaches of secure operating systems. We retain the discussion on approaches that try to increase user awareness or prevent the mounting of phishing attacks (e.g., secure DNS or digitally signed emails).

4.1 Phishing Countermeasures

Protecting the User. Boneh et al. [5] propose heuristic checks of web sites. According to user-defined thresholds, several iterative checks are performed to disclose a site's authenticity. Other heuristics deploy whitelisting and respectively blacklisting approaches, recently adapted by prominent web browser vendors (e.g., [13]). These approaches depend on the report of phishing sites. As long as a phishing site has not been reported, phishers may steal personal data on phishing sites.

There has also been work on fixing flaws of the browser's chrome, as some phishing attacks trick the user in verifying a web site's identity: Ye and Smith [41] render boundaries of browser dialogs according to their origin in different colors blinking synchronized to a reference window. Adelsbach et al. [1] propose to personalize the chrome.

Since SSL server authentication is a reliable method to authenticate web sites, some research has been done to display SSL to non-experts or to strengthen the user authentication. Yee [42] proposes to color the address bar depending on the trustworthiness of server certificates following the policies of traffic lights. Moreover, Herzberg and Gbara [20] propose to augment X.509 certificates with logos being displayed in tamper-resistant regions of the chrome. Ross et al. [33] propose to hash a user-typed password and domain name to provide stronger user authentication. This is an appropriate countermeasure against classical phishing, assuming DNS-based attacks are not present. We will make use of this idea, which we slightly modify and discuss in Section 5.

Nevertheless, none of the approaches achieves our security objective. In particular, they do not fulfill requirements of isolation and trusted paths. Malware phishing attacks are able to alter the chrome and falsify security indicators, as no integrity check of content and programs is provided in general.

Protecting the Interaction of User and Server. Work of this category proposes more user-friendly, password-driven security protocols: First work has been made by Steiner et al. [35]. The authors propose a password-based extension of SSL. Oppliger et al. [29] propose the notion of SSL session awareness. The authors augment SSL to link users' passwords (or any credentials) to SSL sessions. As a result, servers are able to thwart Man-in-the-Middle attacks, as passwords contain information about the actually involved parties. Parno et al. [30] introduce another extension of SSL. Instead of the web browser, a trusted device (e.g., mobile device) is used to automatically verify a web site's authenticity. In addition, Jakobsson et al. [22] propose an oblivious transfer protocol to conjunct password-based mutual authentication with images, i.e., passwords are linked with a sequence of images, which are visual shared secrets.

These approaches are appropriate to combat classical phishing attacks, where the user discloses credentials on a remote system. Nevertheless, they do not fulfill requirements 2 and 3, and thus do not protect against malware phishing attacks, which latch onto the augmented SSL handshake (or any other protocol) and manipulate the communication. That is also true for mobile devices (see e.g., [7]), which do not provide a secure architecture.

4.2 Wallet-based Solutions

Wu, Miller and Little [40] introduce a web wallet, which distinguishes between input of sensitive data and service usage by strictly deactivating login forms in the browser. The user has to press a special security key whenever he wants to enter sensitive data. The web wallet verifies the security properties of the web site and asks the user to explicitly choose the destination site for the sensitive data from a list. The list contains apart from the current site also sites for which an identity has been previously stored. The wallet passes the sensitive data to the chosen site. Also Herzberg [19] discusses a single-click approach storing passwords in a wallet that may be cryptographically protected by keys saved on hardware tokens. To defend against malicious content (which the author considers as the main reason of transporting malware through web browsers), he proposes a browser sandbox model, in which unapproved web objects (e.g., unsigned content) are strictly blocked. Although these approaches reduce the risk of classical phishing attacks, they do not prevent attacks that fake the user interface of the wallet and thus do not meet requirement 3.

4.3 Operation System Approaches

Although operating system approaches do not address phishing attacks specifically, they are essential in building a secure execution environment for application-specific solutions. In the following, we give some references to operating system concepts that are particularly useful against malware in general and thus can be used against malware phishing.

Lampson [23, 24] proposed a security architecture for ordinary computing and ordinary users based on two colors: A red one to perform potentially untrusted tasks (e.g., downloads, adding plugins to browsers from untrustworthy sources), and a green one to perform security-critical tasks (e.g., online banking, taxes). The simultaneous execution of red and green tasks on one platform while preserving isolation can be efficiently achieved by using virtualization technology. Task-specific applications are executed together with their own operating system environment in a virtual machine, and a hypervisor manages the sharing of hardware resources. Hypervisors can be realized based on, e.g., a virtual machine monitor like Xen [4], or microkernels like L4 [26].

Malware may try to mimic user interfaces and the appearance of security-critical applications, e.g., faking password input dialogs. Secure graphical user interfaces enable the user to clearly identify the application he intends to send input to. Moreover, the input and output channels of different applications can be isolated and, hence, this provides a protection against malware trying to eavesdrop data, e.g., keyloggers. Epstein et al. [8, 9] introduced Trusted X, a special X11 window system that enforces secure labeling and authentication of application windows and provides isolation between different security domains. More recently, GUI security was also considered by [34], which is related to the EROS operating system. Nitpicker [12] is a framebuffer-based secure GUI server process on top of the L4 microkernel and controls the physical display while aggregating the virtual screens of client applications. The server also adds labels and border colors to the virtual screens, which enables the user to authenticate the application currently used and displayed.

Other work is related to integrity preservation and verification, which can be used to prevent malware attacks in general. AEGIS [3] performs an integrity check during the boot process of the whole operating system. It protects the integrity reference values by building a chain of trust and protecting the root reference value by special hardware [38]. The authors of [27] show how to use a TPM[2] to implement this approach. The upcoming release of Microsoft Windows "Vista" [28] will also provide a similar approach by encrypting the entire system and binding the encryption key to the boot stack, thus ensuring that system files are unmodified.

Cox et al. [6] propose the Tahoma browser operating system for web applications. They use a security kernel that isolates different web applications by assigning to each service site a browser compartment, running an instance of a web browser, and restricting the communication of that browser compartment. Service providers may provide a policy defining to which web sites the browser instance is allowed to communicate. The authors also present an implementation based on Xen. The browser compartments are realized as virtual machines, respectively. The communication of these browser compartments with web sites is controlled by a network proxy within the security kernel.

While the Tahoma approach is effective against a malware-infected browser trying to pass credentials to a different site other than stated in the policy, it provides no means against classical phishing. If the user is tricked to open a phishing site the Tahoma architecture can only guarantee that there will be an isolated browser compartment for this site. But the user still has to authenticate the web site and may be tricked to enter his credentials in the phishing site. Thus, to prevent both classical and malware phishing attacks, a combination of operating system approaches and other phishing countermeasures seems to be necessary. In the following, we present our security architecture showing such a combination.

[2]The Trusted Platform Module (TPM) is the basic building block of Trusted Computing technology as specified by the Trusted Computing Group (TCG), see `https://www.trustedcomputinggroup.org`

5 Architecture

To prevent phishing attacks, our approach relies on the following ideas: We let a trusted component, called *wallet-proxy*, (i) authenticate legitimate service sites, and (ii) control the secret data of the user's identity including performing the user authentication procedure (see Fig. 2) The wallet-proxy acts as a web proxy from the browser's point of view. This allows the system to be interoperable to existing web browsers. The only action users need to perform is to initialize the wallet by storing sensitive data once. Since the wallet performs the authentication on behalf of the user and passes sensitive user data solely to approved service sites, an unintentional disclosure of the user's identity is prevented. This approach protects only against classical phishing.

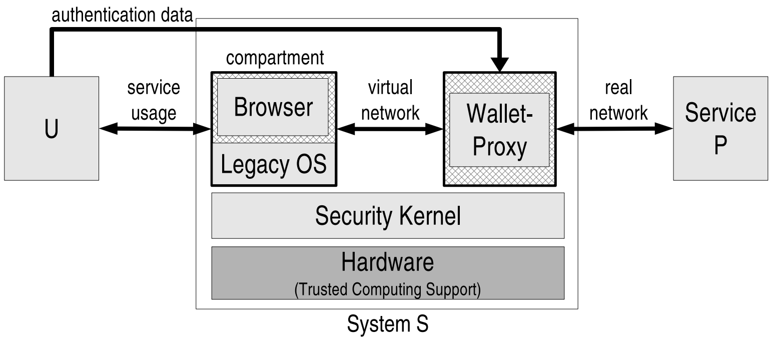


Figure 2: Conceptual view of the architecture.

To protect the user also against malware phishing attacks, we need a secure execution environment. We accomplish this requirement by exploring the idea of trusted and untrusted compartments. The browser is contained within one compartment and the wallet-proxy within another compartment. This is the main difference to existing wallet-based approaches since the wallet functionality is not realized as a browser plug-in or add-on, but it is strictly isolated from the browser except for one communication channel controlled by the security kernel. Thus, malware attacks targeting the browser compartment are confined to this compartment and will not effect the wallet-proxy compartment or other parts of the user's system. Moreover, malware attacks targeting the wallet-proxy compartment must not result in an unauthorized disclosure of the user's secret credentials. Therefore we need an execution environment that guarantees isolation and integrity.

Security Kernel We realize this environment by using the PERSEUS security framework [31]. The PERSEUS framework has a security kernel that provides the following security properties:

1. *Isolation*: Applications can be executed in different compartments. The security kernel enforces strong isolation of compartments while facilitating controlled communication channels between certain compartments and remote systems. To efficiently realize the isolation through compartments, we use virtualization [4, 18]. This means, we can execute off-the-shelf applications and their corresponding operating system in a compartment. This also allows to reuse existing software.

2. *Trusted Path*: The security kernel provides a secure user interface to enable the user to authenticate compartments and clearly distinguish between trusted and untrusted compartments.

3. *System Integrity*: Isolation confines malicious modifications to compartment boundaries and thus can help to preserve system integrity. However, if a critical compartment is affected, the system behavior could change. Therefore, the security kernel facilitates an integrity verification during the boot process as well as at loading and starting time of compartments.

4. *Trusted Storage*: The security kernel provides a storage mechanism that protects integrity and confidentiality for application code and data. Each compartment can have its own isolated trusted storage.

The PERSEUS architecture uses virtualization technology to execute one or more instances of a legacy operating system on top of the trusted software layer. Each virtual machine has its own virtual resources and cannot interfere with the resources of another virtual machine. Virtualization allows for an efficient implementation and usage of legacy software on conventional hardware platforms.

However, virtualization alone is not sufficient to provide a secure operating system. For instance, the integrity verification process must rely on correct integrity reference values. Malware may try to modify these, and offline attacks (e.g., due to booting a different system from a bootable CD-ROM) may maliciously modify critical system components. To confirm the security guarantees of integrity and confidentiality, the PERSEUS security kernel is executed on hardware that supports Trusted Computing functionality, e.g., as provided by a TPM. Since several computer manufacturers already ship their computer platforms equipped with a TPM chip, we can reasonably assume such hardware support.

Trusted Computing Support Trusted Computing (TC) provides security functionalities, which we use for *secure booting* and *sealed storage*. For this, we deploy TC-enabled hardware that measures the integrity of the initial platform boot code and enables the boot loader to establish a secure booting sequence. A measurement is performed by accumulating a cryptographic hash of the binaries in the boot stack. The bootloader is bound to the system configuration of the hardware, i.e., the BIOS; the bootloader loads the basic parts of the security kernel and checks their integrity. The security kernel can then check the integrity of application binaries that are to be executed in compartments (see also [27, 38]).

The TPM can encrypt data using a key that never leaves the TPM. The decryption is bound to the platform configuration stored in the TPM at encryption time (sealing). Hence, the data can only be decrypted if the computing platform has the desired state defined as being trustworthy. We use this functionality to securely store the user's credentials and to ensure that only the wallet can access the storage if the integrity of its inherent compartment is preserved.

6 Realization

In the following, we present details of our security architecture. We describe the static structure and the dynamic behavior of our architecture along major use cases. We first consider in

Section 6.1 a pragmatic approach, where we assume that the underlying computing platform used for a wallet-based solution is an off-the-shelf operating system. We describe a generic architecture based on this and show that the security properties this system provides are insufficient against current phishing attacks. Second, we show in Section 6.2 how the security properties can be achieved by integrating the wallet-based approach into the PERSEUS security architecture framework.

6.1 Wallet-Proxy

Our wallet-based approach basically consists of two modules (see Figure 3): An arbitrary web browser B to access and use services, and a wallet W to store credentials, to identify legitimate service sites, and to perform the user authentication. We prerequisite that the user enters security-sensitive data only into W. Then W acts as a local network proxy for the browser B in order to transparently encapsulate the mutual authentication between user U and service provider P. The authentication information is the tuple (s_{id}, c_{id}, $attr_{id}$), which is kept in a credential store for each service s_{id}.

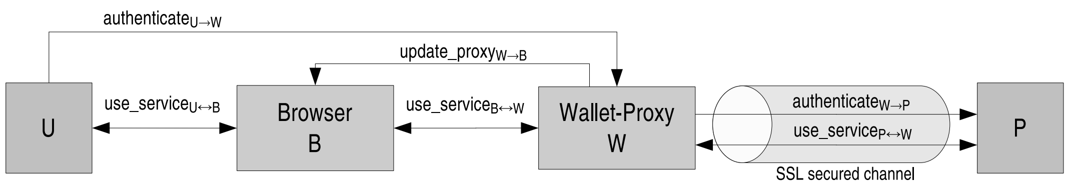


Figure 3: Communication channels of the browser and the wallet-proxy.

6.1.1 Setup

In principle, there are three cases a user authenticates to service P:

Two-Factor Authentication The user receives credentials out-of-band that he uses in an SSL-protected connection. For example, in some European countries banks prefer to send the authentication information by snail mail. Then the authentication is split into two stages: First, the user is instructed to login to site $s_{id} := (URL_{id}, \cdot)$ using username and password denoted as the tuple (u_{id}, pwd_{id}) to get access to his account. Second, he uses an acknowledgment code pwd_{id}^{Ack} to confirm the login. The code may be printed, such as a TAN list, or dynamically generated by a hardware device (token).

In that case, U sets up W manually to store the credentials $c_{id} := (u_{id}, pwd_{id})$ and the service identifier $s_{id} := (URL_{id}, \cdot)$ received out-of-band. To configure W, U uses channel $authenticate_{U\rightarrow W}$. Occasionally, he also deposits some specific attributes $attr_{id}$. When the browser B requests URL_{id} for the first time, an eye-catching dialog pops up informing U that the deposited credentials have been associated with this URL. Then, W saves the server's certificate fingerprint $cert_{id}^{print} \in cert_{id}$, which is used in subsequent requests to identify that site, i.e., if the service identifier s_{id} matches the tuple (URL_{id}, $cert_{id}$), then W performs the login on behalf of the user U.

One-Factor Authentication User and service provider have not agreed on a shared secret before. Therefor, the user negotiates credentials over an SSL-protected web site while signing in to the service. A registration is mandatory for s_{id}.

For this, W looks for forms on the web site which have to be filled out by U, blocks the forms to prevent an unintentional disclosure of credentials and generates a credential profile. Blocking the forms is realized by modifying the HTML code presented to the browser, and this ensures that U enters credentials and attributes only into W. To setup the credentials, U configures W using channel $authenticate_{U \to W}$ by selecting the credential profile and entering the required credentials. W will save them with one slight modification, it will bind credentials to service identifiers. Loosely speaking, W stores random passwords that are linked to cryptographically unique service identifier, such as the fingerprint of the server's certificate $cert_{id}^{print}$. Therefor, W retains the hash value of ${pwd_{id}}^{user}$ concatenated with a random value r instead of the user-typed password ${pwd_{id}}^{user} \in c_{id}$:

$$pwd_{id} := hash\ ({pwd_{id}}^{user} \parallel r)$$

As pointed out, e.g., in [33], we prevent on the one hand that U applies low-entropy passwords to setup the account, on the other hand we ensure that U does not use the same password for different accounts.

Unprotected Authentication The user and service provider negotiate credentials over an unprotected web site. Note that confidentiality and authentication of transferred data is not provided then. However, recall that this case is of particular interest because most phishing-sites use an unprotected connection.

When an insecure channel is established, W shows a warning dialog to inform U that eavesdropping attacks are possible. Anyway, should U decide to register to the site despite the warnings, W proceeds as in the case of one-factor authentication. W blocks the forms and generates a credential profile of the site, which has to be filled out by U. The password pwd_{id} is again modified to the hash of user-typed password ${pwd_{id}}^{user}$ and a random value r. Although the communication is insecure, we show in Section 6.1.4 that pwd_{id} prevents a certain class of phishing attacks anyway.

6.1.2 Login

The user requests a site URL_{id} using channel $use\_service_{U \leftrightarrow B}$. If the wallet-proxy W identifies the service according to s_{id}, W embeds the credentials c_{id} into the site and logs in the user. Channel $useservice_{B \leftrightarrow W}$ is used to attach the credentials c_{id} and attributes $attr_{id}$ to the user's response. All the user sees is being redirected to the original logged-in site in the successful case. Then the service is assumed to be trusted and the user U is allowed to fill out additional forms (e.g., requesting for the acknowledgment code), which are not stored in the wallet. Otherwise, U sees blocked forms requesting for credentials. This keeps the user from revealing personal data to unknown sites and alerts him to enter sensitive data into the wallet only.

6.1.3 Update

An update is important if the user wants to modify some service specific attributes $attr_{id}$ or if the server certificate is invalid. Changing the password should not be necessary, as W uses high-entropy passwords linked to cryptographic identifiers. To update attributes $attr_{id}$, the user U invokes channel $authenticate_{U \leftrightarrow W}$ and selects the corresponding credential profile to configure W. If the server certificate has to be updated, we propose the following security policy. W compares the attributes of the original certificate $cert_{id}$ to the new server certificate $cert_{id}{}^{new}$. In particular, if the issuer is the same and the issuing party is a trusted certificate authority, then W replaces $cert_{id}$ in the credential store; otherwise, a warning message pops up and the user is asked to run the setup. In other words, this policy enforces that any adversary (a) being subject of a trusted certificate may not replace $cert_{id}$ and (b) using untrusted or self-issued certificates obtains credentials that are valid only for the adversarial server.

We argue that the proposed architecture ensures that user's credentials are only transferred to legitimate sites and hence protects against *classical phishing* attacks.

6.1.4 Security Analysis

We first show that the wallet-driven login protects against unintentional disclosure of user's identities. Then we consider security aspects of setting up and updating the wallet. Recall that in a classical phishing attack two cases are possible to lure the user U to a faked site $s_{\widetilde{id}}$.

First, the user is tricked to request a faked site. Then the attack is detected because W was invoked with an unknown service identifier $s_{\widetilde{id}} \neq s_{id}$ and hence does not authenticate U. Moreover, W blocks the login forms. As the user typically does not have to type in the credentials c_{id} to get access to s_{id}, the authentication request therefore attracts his attention. Since we assumed that users enter critical data only into the wallet, the user's identity is not disclosed. Nonetheless, the user could intend to register to the faked site $s_{\widetilde{id}}$. Because $s_{\widetilde{id}}$ is unfamiliar to the wallet, U has to run the setup of W. Then, U initiates W to configure credentials bound to $s_{\widetilde{id}}$, i.e., W generates the password $pwd_{\widetilde{id}}$. Due to the one-wayness of the hash function, it is impossible for a computationally bounded adversary A to gain access to the user-typed password $pwd_{id}{}^{user}$, and hence A is unable to reconstruct pwd_{id}. Thus, the security relies on the collision freeness of the hash function.

Second, the DNS server used by the user has been manipulated to resolve domain names to phishing sites. Then the attack is detected because W fails to authenticate the site on the basis of server certificate $cert_{id}$. More precisely, W compares the digital fingerprints $cert_{\widetilde{id}}^{print} \neq cert_{id}^{print}$. Again, a computational bounded adversary A is unable to compute pwd_{id} (due to the one-wayness of the hash function). This is also true for the update. Consider, for example, the attack in which the adversary A uses self-issued certificates. Then W sets a password $pwd_{\widetilde{id}}$, which is only valid for the faked side identified by an incorrect fingerprint.

Anyway, if credentials have been set up for an unauthenticated service, it is straightforward for the adversary A to spoof[3] URL_{id} and to receive pwd_{id} in cleartext. But note that then identity theft could occur at any node of the Internet. Moreover, due to the randomness in pwd_{id} we prevent that U reveals $pwd_{id}{}^{user}$. Assuming that ordinary Internet users use same passwords for different sites, we deter A from reusing the credentials c_{id} (e.g., A could mount a spear phishing attack to get a list of U's services). Thus, the setup mechanism meets requirement 4.

[3]w.r.t DNS-spoofing attacks; however, aforementioned attacks are prevented because W aborts the user authentication ($s_{\widetilde{id}} \neq s_{id}$)

6.1.5 Discussion

The assumption that the user enters security-critical data only into the wallet-proxy is in practice more realistic and thus weaker than the assumption that the user always correctly verifies the result of the certificate verification. For users unskilled on security, cryptographic certificates have a rather complex meaning, whereas the identification of a clear-cut wallet interface should be much easier. If we additionally enforce that applications behave honestly and do not lie about their identity, they will provide an authentic user interface and will not manipulate the user interface of other applications. Thus, the user will have a trusted path to them. As a very pragmatic solution, we therefore expect that an implementation of the wallet based on existing operating systems, such as Linux or Microsoft Windows, might prevent most of current classical phishing attacks. This assumption is reasonable because these attacks do not impact the integrity of S.

However, the experience has shown that a new form of sophisticated malware phishing attacks can be mounted bypassing these security mechanisms. In practice, legacy operating systems do not provide the desired security properties against this type of attack, i.e., they do not meet the security requirements 1, 2, and 3. In the following, we consider the most important and well-known shortcomings:

- *No trusted path:* Legacy operating systems lack support of a secure user interface providing a trusted path. Malicious applications may then access the authentication data when users enter them into W.
- *No application authentication:* Any application may claim to be another one, users are unable to authenticate applications, and malicious programs (e.g., Trojan horses) may deceive users to reveal sensitive data to dishonest applications.
- *No isolation:* Applications are not protected from each other, a malicious application may access the configuration data of W.
- *No secure boot:* An adversary could manipulate W or the operating system to mount malicious functions.
- *Insecure browser:* An adversary may use scripting and browser plugins to manipulate the behavior of B.

Since we do not expect that the security of the off-the-shelf operating systems will significantly improve in the future, the following subsection 6.2 describes how the wallet-proxy is integrated into the PERSEUS security framework and how it interacts with the core components of the security kernel.

6.2 Secure Platform for the Wallet-Proxy

We divide the system into trusted and untrusted parts following the approach of red/green computing [24]. Although a division into only two domains, trusted and untrusted, may not be generally adequate, this distinction will suffice for the phishing scenario[4]. In section 5 we have already summarized the security properties of the security kernel in PERSEUS. So we only need to show how the wallet-proxy compartment interacts with the security kernel.

[4]For real application scenarios, additional distinct domains may be possible, e.g., one for gaming and one for office work. But this requires more elaborated access policies and will be future work.

6.2.1 Interaction with Trusted System Components

In the following, we focus only on the core components of the security kernel that are of relevance for the wallet-proxy. Figure 4 shows the relevant components and their communication channels.

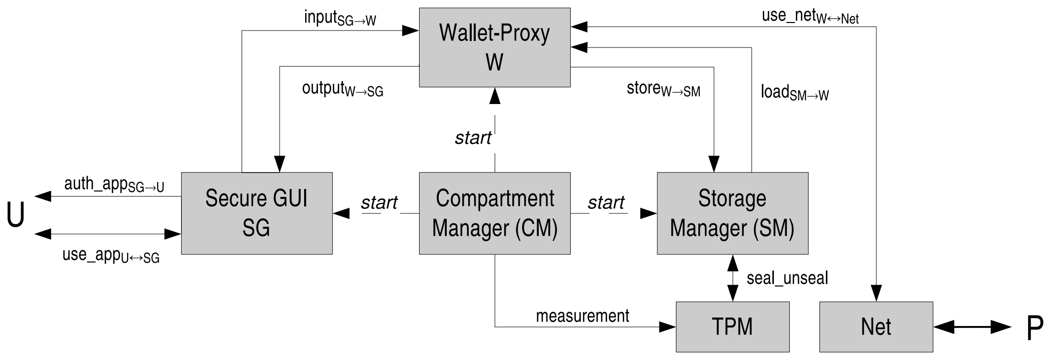


Figure 4: Communication channels of the wallet-proxy with trusted system components).

The user U must be able to clearly authenticate the application currently interacting with, especially when entering secret credentials in W. Thus, U must be able to distinguish between the different compartments. The Compartment Manager CM loads and starts all other components. CM also measures the components and stores the measurement in the TPM, reflecting the integrity of the system.

The SecureGUI component SG solely controls the input and output channels to the user. In order to enable the user to clearly identify compartments, SG provides the channel $auth\_app_{SG \to U}$, which provides the user with the name and color of the compartment that is currently displayed in the channel $use\_app_{U \leftrightarrow SG}$. The input of the user in this channel is passed to the corresponding compartment. In case the wallet-proxy W is currently displayed, U's input is passed through $input_{SG \to W}$ to W, and the output of W is displayed to the user through $use\_app_{U \leftrightarrow SG}$. Each compartment has its own distinct input and output channels to SG. The name and color of a compartment are derived from its measurement, which authenticates the compartment.

To protect the confidentiality of the user's credentials, we use the sealing functionality to bind the secret data to the measurement of W and the underlying security kernel. The wallet-proxy W uses the Storage Manager SM to persistently store the credentials and its configuration. W sends the data through channel $store_{W \to SM}$ to SM, and SM securely stores the data by using the sealing functionality of the TPM and saving the encrypted data. This means, the credentials are encrypted using a key that is protected by the TPM, and the decryption is only possible if the measurement of W and of the security kernel are the same as at encryption time. When W requests to load its credential store, e.g., on system start-up, SM uses the unsealing functionality of the TPM to decrypt the data. Then SM sends the decrypted data through channel $load_{SM \to W}$ to W. W connects to remote network sites through the network driver Net.

6.2.2 Security Analysis (Sketch)

We have already discussed that the wallet-based approach protects against classical phishing attacks. We argue next that our proposed computing platform provides a secure execution environment to also protect against malware phishing attacks, and show that these attacks are unable to interfere the system behavior such that any sensitive data are disclosed. We classify the attacks regarding the targets of modification:

First, A attacks the user-to-compartment channels. He may try to (i) eavesdrop the channel between U and W, (ii) fake the user interface of a compartment to emulate the user interface of W, or (iii) modify the browser compartment B to unblock the forms and deceive U to disclose the credentials. In the first case, the SecureGUI SG controls the input and output and only the compartment currently displayed receives U's input. This means, malware running in a compartment cannot obtain data the user enters into another compartment due to isolation. In the second case, SG provides a visual labeling of each compartment through channel $auth\_app_{SG \to U}$ so that the user can identify the compartment that is currently mapped to channel use_app; U recognizes the faked interface due to the red color of the compartment (see Fig. 7). Thus, SG fulfills requirement 3. In the third case, U fills out the unblocked forms and thus discloses the user-typed password ${pwd_{id}}^{user}$. However, due to the randomness r, which is only known to W, A is unable to reconstruct pwd_{id}.

Second, A modifies the channels between compartments in order to access sensitive data. However, the isolation mechanism confines changes to compartment boundaries, which meets requirement 2. Any modification resulting from malware is restricted to that compartment the malware is running in. Hence, only the outgoing communication of this compartment can be changed. Since the Compartment Manager measures and authenticates each compartment, the integrity of trusted compartments can be verified. If the integrity of the trusted components is preserved, their channels are trusted, i.e., authentic, confidential, and have integrity.

Third, A may try to modify a specific component, e.g., W. There are two possible cases: If the attack is mounted while the system is running, the isolation mechanism prevents a modification across compartment boundaries. Although modifications are allowed in untrusted compartments, they cannot affect the trusted compartments. If, in the second case, A can mount an offline attack, i.e., when the system is not running, the secure boot process will detect a modification of system components at next system start-up, meeting requirement 1. Since U's credentials are sealed by the TPM to a specific measurement of the system, they cannot be unsealed and thus cannot be accessed by A.

7 Prototype Implementation

Although our implementation is only an early prototype, the basic platform is available and executable. The implementation of our prototype is basically an instance of the PERSEUS architecture framework [31]. We use an x86 based system equipped with a TPM [36] to enable Trusted Computing functionalities. We use the bootloader Trusted GRUB [37] to establish a secure booting process. The implementation of the Hypervisor Layer is based on an L4 microkernel [26], which provides isolation of processes and controls inter-process communication (IPC). IPC is used to realize the communication channels between compartments. The Trusted Software Layer is implemented by native L4 applications, which provide the properties of the secure platform. Figure 5 shows the implementation layers of our architecture.

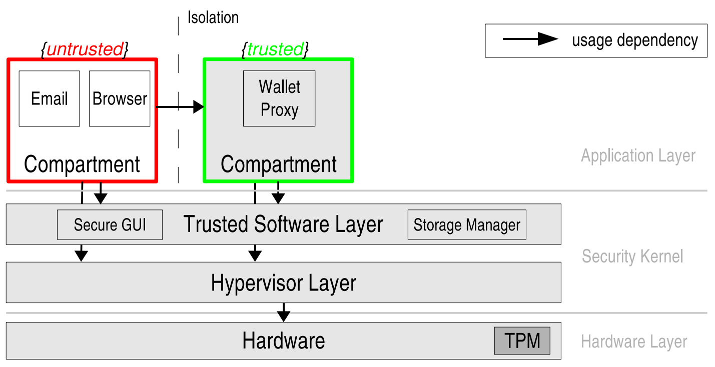


Figure 5: Implementation view of the architecture showing untrusted (red) and trusted (green) compartments. The hardware is TC-enabled by a TPM.

To reuse existing software, we realized the compartments with L4Linux [18], i.e., a para-virtualized[5] Linux system. We used Linux for our prototype because it is open source software and can be easily modified, which is currently necessary for the virtualization. Principally, an implementation based on the Windows operating system would also be possible.

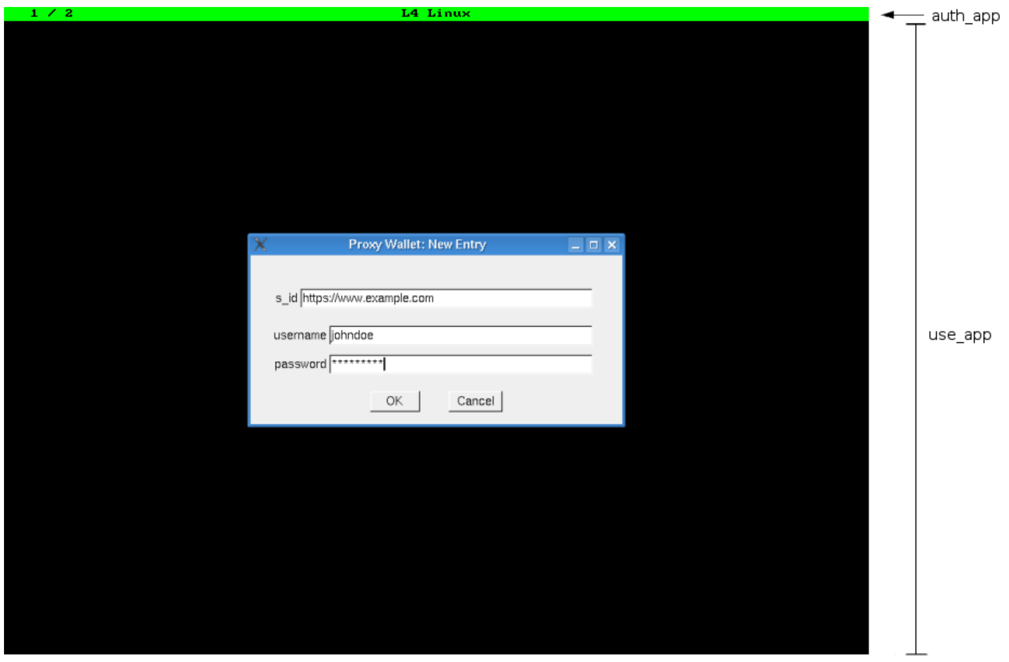
Within an L4Linux compartment, ordinary Linux applications can be executed without modification. Our web browser is a standard Firefox browser. In the first version of the prototype, the wallet-proxy compartment is also a stripped down Linux system. The wallet provides an interface to enter username and password for web sites, see Figure 6 for a screen-shot. However, we have not implemented a web form parser functionality yet. Thus we use a hard-coded version where only the connection to our own test server can be established, which simulates a service provider. The Linux kernel in the wallet-proxy compartment acts as a router for the browser compartment. That means normal Internet network traffic is routed unmodified to the browser compartment. If the browser compartment requests a connection to the test server the wallet-proxy actually establishes the connection, authenticates the user and the test server's SSL certificate, and redirects then the traffic to the browser compartment. The wallet-proxy may later be an application running natively on L4.

The SecureGUI component solely controls the input and output to the user, i.e., keyboard/mouse events and the screen. Each compartment is visually labeled to enable the user to authenticate the currently displayed application. The SecureGUI provides each compartment an isolated framebuffer for drawing GUI elements. There is a reserved area solely controlled by the SecureGUI at the top of the screen to display the compartment label and its color, which implements the channel $auth\_app_{SG \to U}$ (see Figures 6 and 7).

8 Conclusion and Outlook

We have presented a security architecture to protect against different types of phishing attacks. The solution we propose is based on the concept of trusted wallets. It particularly considers the average skilled users, who are the main victims of phishing attacks. If the wallet is executed on a secure platform, malware phishing attacks can be prevented as well. We have

[5] The guest operating system kernel has to be modified to redirect hardware-critical functions calls to the hypervisor.



s_id https://www.example.com

username johndoe

password *********

Figure 6: Screenshot of the wallet-proxy compartment W, showing the green status bar ($auth\_app_{SG \to U}$) indicating a trusted compartment. Channel $use\_app_{U \leftrightarrow SG}$ is mapped to W.

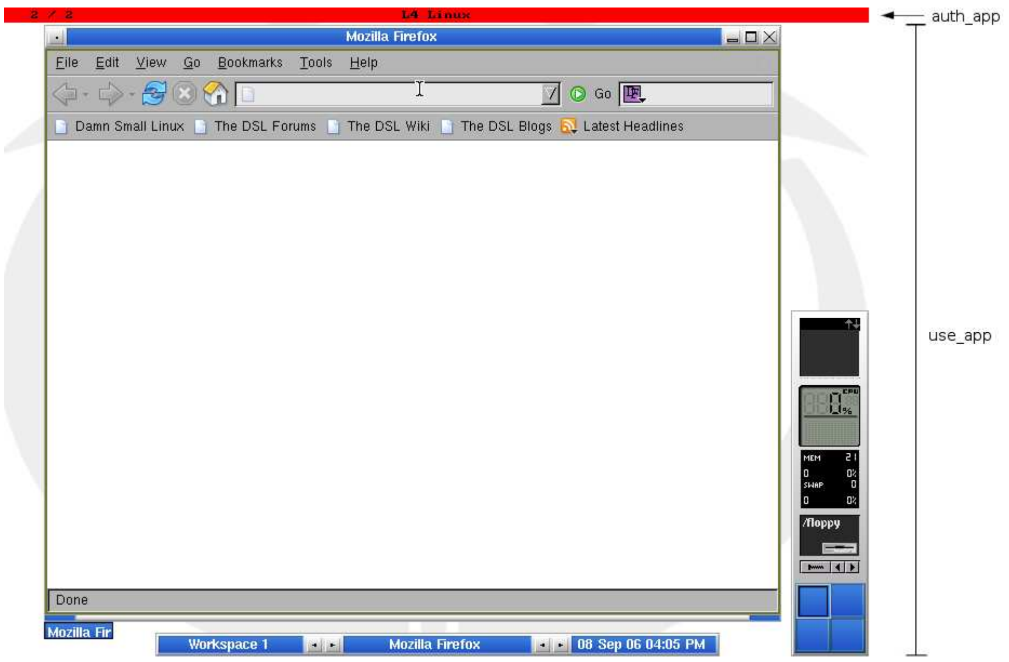


Figure 7: Screenshot of the browser compartment B, showing the red status bar ($auth\_app_{SG \to U}$) indicating an untrusted compartment. Channel $use\_app_{U \leftrightarrow SG}$ is mapped to B.

shown how to efficiently implement such a secure platform based on Trusted Computing and virtualization technology to reuse existing software and keep development costs low. The security architecture can also be implemented on top of a different hypervisor (e.g., Xen [4]). Upcoming processor architectures will provide better support of virtualization, enabling the hypervisor to run unmodified operating systems in compartments, such as Windows. Since many vendors already equip their platforms with a TPM and the upcoming Windows Vista is also going to use it [28], we can reasonably assume the availability of Trusted Computing functionality. Furthermore, the security kernel of our architecture is also used in the Turaya[6] system, which provides a proof-of-concept implementation of a security architecture for various applications.

Our future work aims at augmenting the functionalities of the wallet-proxy, such as allowing to parse forms embedded in emerging web languages (e.g, Ajax or Flash), handling frame-based sites that both download forms over SSL-protected and unprotected sites, or storing and handling additional attributes (e.g., age, address). Finally, we are working on a study to evaluate the usability of our implementation.

References

[1] A. Adelsbach, S. Gajek, and J. Schwenk. Visual Spoofing of SSL Protected Web Sites and Effective Countermeasures. In *Information Security Practice and Experience Conference*, 2005.

[2] Anti Phishing Working Group. Phishing Trend Report(s), 2005-2006. `http://www.antiphishing.com`.

[3] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A Secure and Reliable Bootstrap Architecture. In *IEEE Symposium on Security and Privacy*, pages 65–71, 1997.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[5] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. C. Mitchell. Client-side defense against web-based identity theft. In *11th Annual Network and Distributed System Security Symposium (NDSS '04)*, 2004.

[6] R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen. A Safety-Oriented Platform for Web Applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 350–364, Washington, DC, USA, 2006. IEEE Computer Society.

[7] D. Dagon, T. Martin, and T. Starner. Mobile Phones as Computing Devices: The Viruses are Coming! *IEEE Pervasive Computing*, 03(4):11–15, 2004.

[8] J. Epstein. A Prototype for Trusted X Labeling Policies. In *Sixth Annual Computer Security Applications Conference (ACSAC)*, 1990.

[9] J. Epstein, J. McHugh, H. Orman, R. Pascale, A. Marmor-Squires, B. Danner, C. R. Martin, M. Branstad, G. Benson, and D. Rothnie. A high assurance window system prototype. *Journal of Computer Security*, 2(2):159–190, 1993.

[10] J. Evers. Phishers get personal, 26 May 2005. `http://news.com.com/Phishers+get+personal/2100-7349_3-5720672.html`.

[6]European Multilaterally Secure Computing Base, `http://www.emscb.org`

[11] W. E. Felten, D. Balfanz, D. Dean, and D. S. Wallach. Web Spoofing: An Internet Con Game. Technical Report 540-96, Dept. of Computer Science, Princeton University, 1996.

[12] N. Feske and C. Helmuth. A Nitpicker's guide to a minimal-complexity secure GUI. In *21st Annual Computer Security Applications Conference (ACSAC)*, 2005.

[13] D. Florencio and C. Herley. Stopping a Phishing Attack, Even when the Victims Ignore Warnings. Technical Report MSR-TR-2005-142, Microsoft Research (MSR), 2005. `ftp://ftp.research.microsoft.com/pub/tr/TR-2005-142.pdf`.

[14] German Anti Identity Theft Working Group (a-i3). List of phishing mails and emails seeking for financial agents, 2006. `https://www.a-i3.org/content/category/5/36/84/`.

[15] I. Giang. SSL Phishing, Microsoft Moves to Brand, and Nyms. *Financial Cryptography*, 14 February 2006. `https://www.financialcryptography.com/mt/archives/000654.html`.

[16] I. Giang. Threatwatch—Trojan hijacking, proxy victims, breaching conflicts of legal interest. *Financial Cryptography*, 18 March 2006. `https://financialcryptography.com/mt/archives/000677.html`.

[17] G. Goth. Phishing Attacks Rising, But Dollar Losses Down. *IEEE Security and Privacy*, 03(1):8, 2005.

[18] H. Härtig, M. Hohmuth, J. Liedtke, and S. Schönberg. The Performance of μ-Kernel-based Systems. In *SOSP '97: Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 66–77, New York, NY, USA, 1997. ACM Press.

[19] A. Herzberg. Protecting web users from phishing, spoofing and malware. Cryptology ePrint Archive, Report 2006/083, 2006. `http://eprint.iacr.org/`.

[20] A. Herzberg and A. Gbara. TrustBar: Protecting (even Naive) Web Users from Spoofing and Phishing Attacks. Cryptology ePrint Archive, 2004. `http://eprint.iacr.org/2004/155.pdf`.

[21] K. J. Hole, V. Moen, and T. Tjstheim. Case Study: Online Banking Security. *IEEE Security and Privacy*, 4(2):14–20, 2006.

[22] M. Jakobsson, S. Myers, and M. Augiere. Delayed Password Disclosure, 2005. `http://www.informatics.indiana.edu/markus/stealth-attacks.htm`.

[23] B. W. Lampson. Accountability and Freedom. `http://research.microsoft.com/~risaacs/blampson.ppt`, Oct 2005.

[24] C. E. Landwehr. Green Computing. *IEEE Security & Privacy*, 3(6):3, Nov/Dec 2005.

[25] E. Levy. Criminals Become Tech Savvy. *IEEE Security and Privacy*, 02(2):65–68, 2004.

[26] J. Liedke. On Microkernel Construction. In *15th ACM Symposium on Operating System Principles*, 1995.

[27] J. Marchesini, S. W. Smith, O. Wild, J. Stabiner, and A. Barsamian. Open-Source Applications of TCPA Hardware. In *20th Annual Computer Security Applications Conference (ACSAC)*, 2004.

[28] Microsoft Corp. Secure Startup - Full Volume Encryption: Technical Overview. `http://www.microsoft.com/whdc/system/platform/pcdesign/secure-start_tech.mspx`, April 2005.

[29] R. Oppliger, R. Hauser, and D. Basin. SSL/TLS Session-Aware User Authentication-Or How to Effectively Thwart the Man-in-the-Middle, 2005. (Computer Communications, accepted for publication).

[30] B. Parno, C. Kuo, and A. Perrig. Phoolproof Phishing Prevention. In *Financial Cryptography*, 2006. (to appear).

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

[32] J. D. T. Rachna Dhamija and M. Hearst. Why Phishing Works. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI2006)*, 2006.

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

[34] J. S. Shapiro, J. Vanderburgh, E. Northup, and D. Chizmadia. Design of the EROS Trusted Window System. In *USENIX Security Symposium*, pages 165–178, 2004.

[35] M. Steiner, P. Buhler, T. Eirich, and M. Waidner. Secure password-based cipher suite for TLS. *ACM Transactions on Information and System Security*, 4(2):134–157, May 2001.

[36] Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 85, Trusted Computing Group, Feb. 2005.

[37] Trusted GRUB. `http://trustedgrub.sourceforge.net`.

[38] J. Tygar and B. Yee. Dyad: A System Using Physically Secure Coprocessors, 1994. Technical Report CMU-CS-91-140R.

[39] W. Wang, Y. Yuan, and N. Archer. A Contextual Framework for Combating Identity Theft. *IEEE Security and Privacy*, 4(2):30–38, 2006.

[40] M. Wu, R. C. Miller, and G. Little. Web Wallet: Preventing Phishing Attacks by Revealing User Intentions. In *SOUPS '06: Proceedings of the Second Symposium on Usable Privacy and Security*, pages 102–113. ACM Press, 2006.

[41] Z. E. Ye and S. Smith. Trusted Paths for Browsers. In *USENIX Security Symposium*, pages 263–279, 2002.

[42] K.-P. Yee. Designing and Evaluating a Petname Anti-Phishing Tool, 2005. `http://cups.cs.cmu.edu/soups/2005/2005posters/23-yee.pdf`.

[43] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities. In *Network and Distributed System Security (NDSS) Symposium*, 2006.