

Towards Multicolored Computing - Compartmented Security to Prevent Phishing Attacks

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

Horst Görts 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 fostered to a major concern of Internet users. Classical 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 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 (i) the multicolored computing (e.g., red for the risky and green for the trusted domain), and (ii) a trusted wallet for storing credentials and authenticating sensitive services. Our solution requires no special care from users for identifying the right web sites while the disclosure of credentials is strictly controlled.

We present the main idea of how to integrate countermeasures against Phishing and malware into one sound security architecture. Our approach establishes compartmented security for mounting isolated applications, provides a secure graphical user interface to configure sensitive applications, and performs secure booting to preserve the system integrity. We also give hints on how to implement this architecture efficiently by utilizing trusted computing functionality and virtualization.

1 Introduction

The Internet and its underlying infrastructure might constitute the most complex IT system ever built. On the one hand, this huge platform offers us many opportunities to access information, gain knowledge and set up a variety of business models with sophisticated functional and security re-

quirements. On the other hand, it also bears many risks that can be exploited by adversaries who misuse this powerful medium driven by various motivations. In this context, the issue of identity theft has become a subject of great concern in the recent years: Since password-based user authentication established on the Internet to grant users access to security critical services, identity theft and fraud attracted attackers [37]. Hence, phishing—a colloquial abbreviation of *password fishing*—has become a prominent attack. Whereas *classical* phishing attacks primarily used rogue emails to lure unwary users to faked web sites where they revealed personal information (e.g., passwords, credit card numbers, transaction numbers), current attacks have become advanced in their number and technical sophistication [2, 16, 20]. This type of phishing 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 more precisely mount classical phishing attacks [10], *pharming* compromises DNS servers to resolve domain name requests to phishing sites [2], and *malware phishing* infiltrates customers' computers to log their password stroking using special malicious programs [24].

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., [18]). Studies point out that ordinary Internet users often do not distinguish legitimate from faked web sites and do not understand security indicators [31]. Thus, they are vulnerable to classical phishing attacks: To reliably authenticate a web site, the user has to verify the domain name, 'https' in the URL, and the server certificate. However, ordinary Internet users are unfamiliar with the meaning of SSL. This is in particular true for common

phishing attacks, as most phishing sites may have been exposed, if users had properly looked at the presence of SSL.¹

More dramatically, the rise of malware phishing attacks is a strong evidence for the lack of appropriate and fundamental protection of common computing platforms in practice. The problem with malware phishing attacks is that they are specifically designed to target certain service providers (e.g., domestic banks) providing tailored functionality to obtain users' credentials [2, 15, 24]. For this, these phishing attacks fake, e.g., security indicators, imitate the browser's (or any security-critical application's) chrome or modify the system configuration. Thus, malware phishing attacks likely circumvent present phishing countermeasures and most anti-virus products that generally support updates of wide-spread threats.

Contribution In this paper, we make the first steps toward the design and implementation of a security architecture with the property to counter both classical and malware phishing attacks. We propose a modular platform that is interoperable to legacy operating systems and 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. To establish a secure execution environment for the wallet, our implementation uses Trusted Computing functionality², and virtualization to efficiently reuse existing software, keeping the development costs low. The novelty of the work is to present a concrete solution idea that combines both aspects of secure operating systems using trusted computing functionalities and secure wallet-based user authentication applied to the problem of phishing attacks. While this paper only describes the main concepts of our approach, a future paper will describe the architecture and its implementation in more detail. Currently, we have implemented a prototype based on the L4 microkernel and Linux.

2 Basic Approach

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

Objective (Confidentiality of Credentials)
The system must protect the user's credentials against any unauthorized access.

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

²as specified by the Trusted Computing Group

An adversary must not gain access to the user's credentials. These credentials must be protected against phishing attacks. This especially means that credentials must only be given to authorized sites. To prevent phishing attacks, our approach relies on the following ideas: We let a trusted component, called *trusted wallet*, (i) authenticate legitimate service sites, and (ii) control the secret data of the user's identity including performing the user authentication procedure on behalf of the user (see Fig. 1). The trusted wallet 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.³ 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.

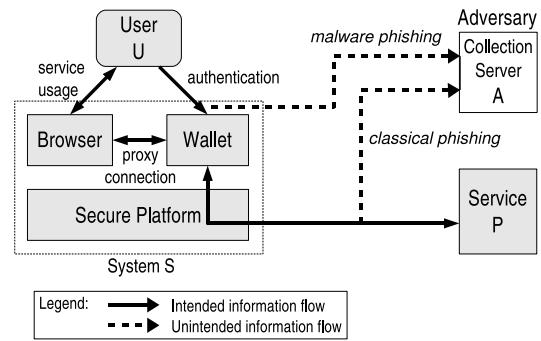


Figure 1: Illustration of the general idea.

To protect the user also against malware phishing attacks, we require a trusted execution environment. We accomplish this requirement by exploring the idea of trusted and untrusted compartments (multicolored computing) relying on Trusted Computing functionality and virtualization technology. The latter allows for an efficient implementation and usage of legacy software. The underlying security kernel provides a variety of security properties, such as strong isolation of compartments facilitating controlled communication channels between certain compartments and remote systems, and a secure user interface to enable the user to clearly distinguish between trusted and untrusted compartments.

Trusted Computing Support Trusted Computing (TC) provides security functionalities which we use for *secure booting* and *sealed storage*. For

³To ensure that the user initially is not tricked by faked (password) dialogs, we utilize a trusted Graphical User Interface (GUI).

this, we deploy TC-enabled hardware that measures the integrity of the BIOS boot code and enables the operating system’s boot loader to establish a secure booting sequence. TC allows to store an image of the current system configuration in a special hardware component, the Trusted Platform Module (TPM).⁴ The TPM can encrypt data using a key that never leaves the TPM. The decryption can be bound to the system 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 the wallet accesses the storage only if the integrity of its inherent compartment is preserved.

3 Model

3.1 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 . 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 virtual 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, $send_{U \rightarrow S}$ is the unilateral channel that U uses to send a message to S .

Identities are security sensitive information and target of phishing attacks. We denote an identity $identity_{sid}$ as the tuple $(sid, cid, attrid)$ where sid indicates a set of unique service provider identifiers (e.g., domain name, server certificate), cid a set of credentials to get access to P , and $attrid$ a set of attributes and claims specific to user and service (e.g., age, address, credit card number). Credentials cid establish the claim that U is in possession of $identity_{sid}$ and are denoted as the tuple (uid, pwd_{id}) , whereas uid and pwd_{id} are username and password.

3.2 Phishing Attacks

The goal of phishing attacks is to obtain $identity_{sid}$. These attacks generally consist of two elementary stages [1]: In the first stage, preliminary attacks are launched to mount the actual illusion. In the second stage, illusion attacks are launched to mimic,

⁴Today, several computer manufacturers already ship their computer platforms equipped with a TPM chip.

e.g., legitimate text, images and windows to deceive the user. In this stage all the stolen identities are gathered in a collection server. Phishers therefore utilize various strategies, which we briefly introduce in the following:

Classical phishing attacks have in common that the divulgement of $identity_{sid}$ 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 $identity_{sid}$ on client side. Some variants of malware phishing attacks alter 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 [41] or attach malicious programs to phishing mails (see [2]).

3.3 Security Assumptions

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

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 sid .*

We already mentioned in the introduction that U has to verify sid 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 [18, 31] point out that ordinary Internet users are bad in distinguishing legitimate from faked web sites and do not understand indicators, which signal trustworthiness. This is in particular true for phishing attacks, as most phishing sites may have been exposed, if users had properly looked at the presence⁵ of SSL. Therefore, $chan_{U \leftrightarrow S}$ is assumed

⁵Note that phishers generally do not endeavor to deploy and/or imitate SSL. We reviewed more than 10.000 phishing sites [14] and remark that no site was protected by SSL.

to be untrusted if U has to verify the authenticity of P .

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; otherwise intruders were able to steal identities from the service provider's database. Moreover, services are resilient against web spoofing attacks [11], which provide adversaries to completely display a faked web, i.e., we do not consider scenarios, where phishers may impersonate a service while U registers to 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), respectively. Nevertheless, if the application is vulnerable to, e.g., buffer overflow or format string attacks, then the security platform only safeguards that the intruder gains no more information than given in the compartment.

3.4 Requirements

To be able to provide the security objective, the system S has to fulfill the following requirements:

Requirement 1 (System Integrity) *The integrity of security-critical components in S is 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 modified and integrity violations are not recognized. Thus, a weaker version of this requirement is to demand a verification of the integrity and to stop the execution of the system when an integrity violation has occurred.

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

Malicious processes must not be able to access the internal state or the persistently stored state of other processes.

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.*

The user must be sure about the integrity and authenticity of the application when he is going to enter credentials. For instance, emulating password input dialogs is a common attack of Trojan horse programs. Thus, the user needs a way to invoke a trusted path to security-critical input dialogs.

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 regarding security indicators (e.g., SSL padlock icon). 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.

Requirement 5 (Interoperability) *The system S should be interoperable to commodity services that third parties provide.*

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. 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. 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. Otherwise, we believe that the system would not be widely deployed.

4 Related Work

In this section, we discuss recent work on counteracting phishing attacks and categorize them in protection mechanisms addressing the browser, the operating system, and the interaction of the user and the server. We also discuss wallet-based solutions, which are tangential to our trusted wallet approach.

Browser-based Countermeasures Methods of this class support unwary users to identify faked web sites and/or not to reveal their credentials. In [5] a heuristic check of web sites is proposed. According to user-defined thresholds, several iterative checks are performed to disclose a site's identity. The shortcomings of this approach are false positives: If the phisher knows how web sites are analyzed, he can modify phishing sites to circumvent

the corresponding checks. Less advanced heuristics deploy whitelisting and respectively blacklisting approaches, recently adapted by prominent web browser vendors (e.g., [13]). There has also been work on fixing flaws of the browser chrome, as some phishing attacks deceive the user from correctly identifying a web site: The authors in [39] propose to render boundaries of browsers' dialogs according to their origin in different colors blinking synchronized to a reference window. In [1, 7], the authors propose to personalize the chrome. Some research was also made to display SSL with respect to the diligence and capabilities of average Internet users: In [40], the author proposes to color the address bar depending on certificates' trustworthiness according to policies of traffic lights. Moreover, the authors in [19] propose the deployment of logo-type certificates and display logotypes (e.g., brand marks) in tamper-resistant regions of the browser chrome. The provision of an interface transparent to users is essential, since a basic methodology of phishing attacks is to fake relevant connection and security indicators of browsers to hide the illicit origin of phishing sites.

However, these approaches do not achieve our security objective. In particular, these approaches do not fulfill requirements 2 and 3: Malware phishing attacks are able to alter the chrome and security indicators respectively, as no integrity check of content and programs is provided in general.

Wallet-based Solutions In [38], the authors 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 sensitive data to the chosen site. Also [18] 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. Moreover, the authors do not provide any means to prevent malware phishing attacks and thus do not meet requirement 1.

Operating System Approaches Lampson [22, 23] 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). 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 [8] provides a policy for labeling windows, showing the security level of the corresponding application. This is accomplished by Trusted X [9], a special X11 window system that enforces the labeling and provides isolation between different security domains. More recently, GUI security was also considered by [33], which is related to the EROS operating system in special. In [12], a framebuffer-based secure GUI server is presented. This allows a more generic application of this technique, since not all operating systems use an X11 like window system.

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 [36]. The authors of [26] show how to use a TPM to implement this approach. The upcoming release of Microsoft Windows "Vista" [27] 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.

Protecting the Interaction of User and Server Work in this category elaborates new approaches on the level of user-friendly security protocols: In [21], the authors propose an oblivious transfer protocol to conjunct password-based mutual authentication and images, i.e., passwords consist of a sequence of images, which are visual shared secrets. The authors in [28] propose the notion of SSL session awareness and augment SSL to couple 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 actually involved parties. Another extension of the SSL handshake is introduced in [29], where a trusted device (e.g., mobile device) instead of the web browser is used to automatically verify a web site's authenticity.

These approaches are appropriate to combat classical phishing attacks, where the user discloses cre-

dentials on a remote system. Nevertheless, they do not fulfill requirements 2 and 3, and thus do not protect against malware phishing attacks. These attacks may latch onto the SSL layer and manipulate the communication. That is also true for mobile devices (see e.g., [6]), which do not provide a secure architecture.

5 Architecture

In this section, we present our security architecture to prohibit identity theft through phishing attacks. We describe the static structure and the dynamic behavior of our architecture along one major use case. We first consider in Section 5.1 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 5.2 how the security properties can be achieved by an efficient secure operating system design and Trusted Computing functionality.

5.1 Wallet-based Solution

Despite the fact that an honest web browser may be applied, the user may be cheated to request a faked web site. Moreover, he may be tricked to disclose credentials, if he does not closely verify the authenticity of the site. To counteract this type of identity theft, we argue that the user's system should be responsible for authenticating the service provider and perform the user authentication on legitimate service sites. Based on this, we propose to utilize the following wallet-based architecture:

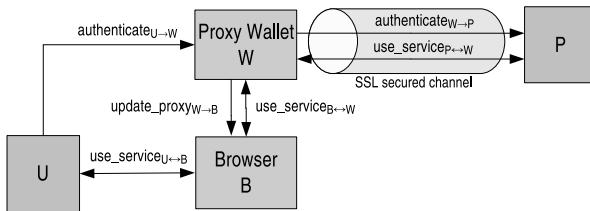


Figure 2: Architecture of the wallet-based Solution.

The architecture basically consists of two modules (see Figure 2): A standard web browser B to access the services of provider P , and a Proxy Wallet W to store credentials $c_{id} := (u_{id}, p_{wd_{id}})$ identifying legitimate service sites, and perform the user authentication. We assume that the user enters security-sensitive data only into W . Then W acts as a local net-

work proxy for the browser B in order to transparently encapsulate the mutual authentication between user U and provider P .

5.1.1 Web Browser

The web browser B provides the user U to use web services. However, connection requests for services that require authentication are redirected to W . The only action of the user U we presume is entering credentials into W instead of B . W configures B such that access to these services is redirected to the proxy. This functionality is supported by common web browsers. Hence, B forwards the connection request to W using the channel $use_service_{B \leftarrow W}$, if a user authentication is required. After W has performed the mutual authentication, the user can use the service within B .

5.1.2 Proxy Wallet

The Proxy Wallet W provides a configuration dialog (channel $authenticate_{U \rightarrow W}[s_{id}, u_{id}, p_{wd_{id}}]$) allowing the user to enter credentials $c_{id} := (u_{id}, p_{wd_{id}})$ belonging to the service identified by s_{id} . Since W should perform the mutual authentication automatically, the user has to enter the sensitive information before they are used to authenticate a session.

When the browser B requests a connection to the service site s_{id} , W actually establishes the connection to the service provider's system P and authenticates P based on a verification of a SSL/TLS certificate (more precisely, the certificate's fingerprint). Note that typically the user has to verify the result of the verification, while in our architecture this is done by W . If the site is not legitimate, W aborts the connection and displays an eye-catching error message. Only if the service site is legitimate, W transfers the corresponding credentials to P by using the secure channel $authenticate_{W \rightarrow P}[u_{id}, p_{wd_{id}}]$. If the login data are correct, P returns an authenticated session denoted by using the channel $use_service_{P \leftarrow W}$, which is forwarded by W to the web browser B . The user U may now use the service as usual.

5.1.3 Security Analysis (Sketch)

Based on the assumptions made about the web browser and Proxy Wallet, we argue that the proposed architecture ensures that the user's credentials only are transferred to legitimate sites and hence protects against classical phishing attacks. To lure the user U to a faked site \widetilde{s}_{id} , two cases are possible:

In the first case, the user U is tricked to request a faked site because of being cheated by an obfus-

cated URL (e.g., in a phishing mail). The attack is detected because W was invoked with an unknown service identifier $\widetilde{s_{id}} \neq s_{id}$ and hence does not authenticate U . Moreover, Proxy Wallet locks the login forms. As the user U 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.

In the second case, the DNS server used by U has been manipulated to resolve legitimate domain names to IP addresses of phishing sites. Although the locally used domain name is correct and B invokes W to perform the user authentication, the attack is detected because W fails to authenticate the site on the basis of server certificates by comparing the fingerprints. However, the comparison is only feasible, if an SSL-protected connection has been established. In the case of unprotected connections, we may not reliably verify the server's identity. But note that in this case, the service provider P does not intend to provide a secure channel and thus identity theft could occur at any node of the Internet.

5.1.4 Discussion of Assumptions

The assumption that the user enters security-critical data only into the Proxy Wallet is in practice more realistic and thus weaker than the assumption that the user always correctly verifies the result of the certificate verification. For unskilled users regarding security issues, cryptographic certificates have a rather complex meaning, whereas the identification of a clear-cut wallet interface should be much easier. If we additionally assume 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 Proxy 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 . Classical phishing attacks do not change or modify system components; instead they relay the communication channel to a remote system.

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. In the following, we are going to consider the most important and well-known shortcomings:

- *No trusted path:* Since legacy operating systems lack support of a secure user interface providing a trusted path, malicious applications may access the authentication data when users enter them into W .
- *No application authentication:* Since 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:* Since applications are not protected from each other, a malicious application may access the configuration data of W .
- *No secure boot:* An adversary could manipulate the binary of 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 5.2 describes a security architecture providing a trustworthy computing base for the Proxy Wallet with respect to reuse of present components.

5.2 Secure Platform for the Wallet

We now show how the security requirements can actually be realized. The wallet-based architecture is therefore slightly modified, and we introduce a secure computing platform as execution environment. This follows the approach of red/green computing [23] by dividing the system into trusted and untrusted parts⁶ as shown in Figure 3 and 4.

Handling Insecure Browsers Since users tend to install additional software (plugins) into their web browsers, probably originating from an untrustworthy source, we assume that manually installed software may be infected by malware and thus be dishonest. For using security-sensitive services, such as online banking, this is a critical issue. Therefore, we additionally use a *trusted browser* TB . The trusted browser is a zero-footprint version of a standard web browser, i.e., any modifications,

⁶Although a division into only two domains, trusted and untrusted, may not be generally adequate, this distinction will suffice for the phishing scenario. 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.

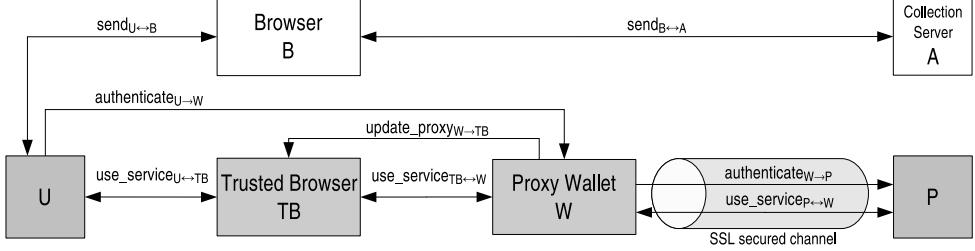


Figure 3: Logical view of the architecture with untrusted components (colored in white).

such as installing plugins, are disallowed. We use TB then to access security-sensitive services that require an authentication of the user.

We modify the wallet-based architecture in the following way (cf. Figure 3): Only the trusted browser TB is authorized to call W and to connect to the service site. Communication channels between an untrusted browser B and W are forbidden. B provides the user a way to use services that do not require an authentication. Modifications of B are allowed, e.g., the user can add extra functionality. Thus, malware may infect B and establish a (covered) connection to adversary A , i.e., a collection server, denoted by the channels $send_{U \leftrightarrow B}$ and $send_{B \leftrightarrow A}$. The channels transfer arbitrary data and the user is not able to identify the endpoint of this communication.

Providing Isolation To protect the different components and to prevent unintentional communication, we have to isolate the processes and only allow controlled communication channels. We therefore execute each component in a different compartment. There can be trusted and untrusted compartments. W is executed in a trusted compartment, whereas the untrusted browser is executed in an untrusted compartment. To efficiently realize the isolation through compartments, we use virtualization [4, 17]. This means, we can execute off-the-shelf applications and their corresponding operating system in a compartment. This allows for us to reuse existing software meeting requirement 5 and minimizes development and maintenance costs.

Realizing a Trusted Path 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. Moreover, communication channels between the user and trusted compartments must be secure, i.e., a trusted path. Therefore, we use the concept of a *Secure GUI*: There is one trusted component solely controlling the input and output to the user, i.e., keyboard/mouse events and the screen. Depending

on the currently used channel by the user, the input data is only passed to the corresponding compartment and vice versa for the output. 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 to display the compartment label and its color. This realization is similar to the one described in [12].

System Integrity Isolation confines malicious modifications to compartment boundaries and thus can help to preserve the system integrity. However, the system behavior changes if the compartment is infected. Best practice is to verify the integrity of components before they are executed. Integrity verification relies on the comparison of current component properties to reference values. The security of integrity verification thus relies on the integrity of the reference values. We use Trusted Computing functionality to protect the initial reference value and establish a secure boot process. During this process, the integrity of all relevant system components, i.e., the trusted computing base (TCB), is measured and the measurement result is stored within the TPM. The concepts of this are borrowed from [26, 36].

We briefly describe the secure boot process: The bootloader is bound to the system configuration of the hardware, i.e., the BIOS; the bootloader loads the basic modules of the security kernel and checks their integrity. This includes a special component we call *Compartment Manager*. The Compartment Manager loads the remaining compartments and measures their integrity. The Compartment Manager uses the measurements to authenticate trusted and untrusted compartments. The compartment authentication allows for us to control the communication through channels as well as to create the corresponding labeling for the SecureGUI.

Sealing Secret Data To protect the confidentiality of credentials, we use the sealing functionality and bind secret data to the configuration of W

and the TCB. This means, the credentials are encrypted using a key that is protected by the TPM, and the decryption is only possible if the state of W and the TCB are the same as at encryption time. Moreover, before W uses the data to perform an authentication, it verifies the properties of TB by using the information provided by the Compartment Manager. Only if TB is unmodified, W will pass the secret data to the corresponding service site and establish a proxy connection to TB .

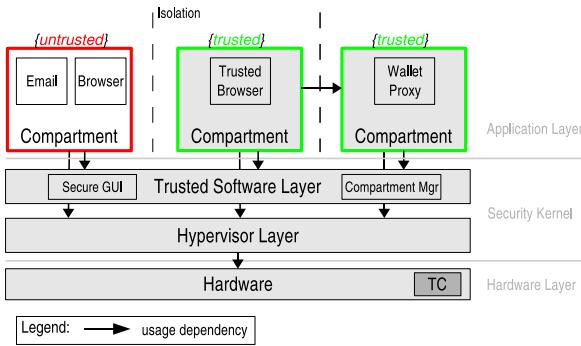


Figure 4: Layered view of the architecture showing untrusted (red) and trusted (green) compartments. The hardware is TC-enabled, e.g., by a TPM.

Figure 4 shows the layers of our security architecture. The secure computing platform is realized by the security kernel and hardware that provides Trusted Computing support. We divide the security kernel into two further layers: The *Hypervisor Layer* is responsible for the actual virtualization of hardware resources, whereas the *Trusted Software Layer* comprises the security services that are needed to fulfill the security requirements.

5.2.1 Security Analysis (Sketch)

We have already argued 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. These attacks try to modify the intended system behavior. They can be classified regarding the target of modification:

1. Modification of channels from the user to the system compartments.
2. Modification of channels between compartments.
3. Modification of a compartment itself.
4. Modification of components of the underlying security kernel.

1) There are two possible cases: In the first case, an attacker may try to modify a channel directly, e.g., to eavesdrop data the user enters into W . However, the SecureGUI controls the input and output, and only the currently displayed compartment receives the user's input. This means, malware running in a compartment cannot obtain data the user enters into another compartment. In the second case, a malicious compartment can try to indirectly change the channel by faking the user interface of a trusted compartment. For instance, an attacker may try to emulate the user interface of W to deceive the user to enter her credentials in the channel $send_{U \rightarrow B}$. The SecureGUI though provides a visual labeling of each compartment, and the user can definitely identify the current communication channel. This prevents the user from entering data into a compartment that claims to be a different one.

2) Malware may try to change the channels between compartments in order to eavesdrop sensitive data. However, the isolation mechanism confines changes to compartment boundaries. Any modification resulting from malware is restricted to that compartment the malware is running in. Hence, only the communication channels for this compartment can be changed. Since the Compartment Manager measures and authenticates each compartment, the channels for trusted compartments cannot be changed. If the integrity of the trusted components is preserved, their channels are trusted, i.e., authentic, confidential, and have integrity.

3) An attacker may try to modify a specific compartment, such as 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 the untrusted compartment, they cannot affect the trusted compartments. If, in the second case, an adversary can mount an offline attack, i.e., when the system is not running, the secure boot process will detect a modification at next start-up. Since the user's credentials are sealed to a specific platform configuration (hardware and software), they cannot be unsealed and thus cannot be accessed after a malicious modification.

4) The integrity of the trusted components is verified during the secure boot process. If the underlying security kernel is maliciously manipulated, the integrity measurement results in the TPM will also change. Because the measurement will differ from the one at sealing time, the user's credentials cannot be unsealed. Thus, the secret data cannot be accessed and their confidentiality is preserved.

6 Implementation

The implementation of our prototype is basically an instance of the PERSEUS secure platform architecture [30]. We use an IA32 based system equipped with a TPM [34] to enable Trusted Computing functionalities. We use the bootloader Trusted GRUB [35] to establish a secure booting process. The implementation of the Hypervisor Layer is based on an L4 microkernel [25], 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. To reuse existing software, we realized the compartments with L4Linux [17], i.e., a para-virtualized⁷ Linux system. We used Linux for our prototype implementation because it is open source software and can be easily modified, which is currently necessary for the virtualization. However, an implementation based on the Windows operating system would also be possible in principle.

Within an L4Linux compartment, ordinary Linux applications can be executed without modification. Our web browser is a standard Firefox browser. The Trusted Browser compartment is realized by using a stripped down Linux system running a zero-footprint version of Firefox. In the first version of the prototype, the Proxy Wallet compartment is also a stripped down Linux system running a proxy application. However, the Proxy Wallet may later be an application running natively on L4. Thus, the Proxy Wallet can be seen as another application program on a PC, but it is protected by the virtualization and strong isolation capability of the underlying security kernel. This is the major difference to existing wallet approaches, which usually integrate the wallet in the browser.

In order to perform the user authentication using the user's credentials, the Proxy Wallet has to examine the HTTP content of the requested service sites for login form data. It does not need to examine the whole network traffic, since the login process is indirectly triggered by the request of the Trusted Browser. Thus, the Proxy Wallet "knows" when it has to scan for login data, since the login sites are usually the first sent by the service provider. Due to the implementation of the Proxy Wallet within a stripped down Linux compartment we can easily reuse existing open-source form data scanning engines. However, since there is still a lack of a standardization for login form data, the implementation of the Proxy Wallet has to be adjusted to new ser-

vice sites if their login forms differ. But once such a standardization exists, the implementation of the Proxy Wallet will be easily made compatible.

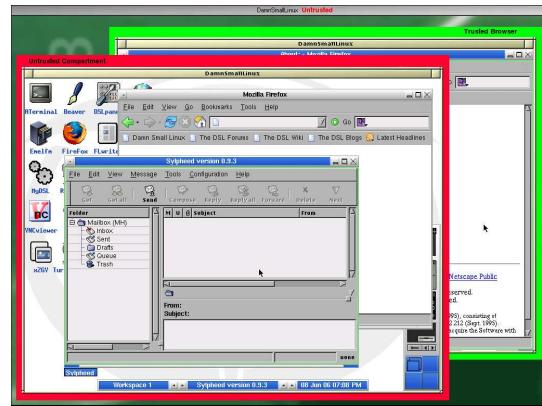


Figure 5: Screenshot, showing compartments differently marked by the SecureGUI.

7 Conclusion and Outlook

We have presented the main idea of 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 shown how to efficiently implement such a secure platform based on Trusted Computing and virtualization technology. The latter enables us 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 guest operating systems, such as Windows. To overcome update problems due to the binary attestation of the system configuration, we will work on the integration of property-based attestation [32] within the integrity measurement. Currently, we are working on a user study to evaluate the usability of our approach and our prototype implementation. Moreover, to store additional attributes (e.g., age, address), which are also identity-related, future work will investigate how to automatically handle such identity claims requested by arbitrary services.

⁷ The guest operating system kernel has to be modified to redirect hardware-critical functions calls to the hypervisor.

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] D. Dagon, T. Martin, and T. Starner. Mobile Phones as Computing Devices: The Viruses are Coming! *IEEE Pervasive Computing*, 03(4):11–15, 2004.
- [7] R. Dhamija and J. D. Tygar. The Battle Against Phishing: Dynamic Security Skins. In *SOUPS '05: Proceedings of the 2005 Symposium on Usable Privacy and Security*, pages 77–88, New York, NY, USA, 2005. ACM Press.
- [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.
- [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. Threatwatch—Trojan hijacking, proxy victims, breaching conflicts of legal interest. *Financial Cryptography*, 18 March 2006. <https://www.financialcryptography.com/cgi-bin/mt/mt-tb.cgi/672>.
- [16] G. Goth. Phishing Attacks Rising, But Dollar Losses Down. *IEEE Security and Privacy*, 03(1):8, 2005.
- [17] 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.
- [18] A. Herzberg. Protecting web users from phishing, spoofing and malware. Cryptology ePrint Archive, Report 2006/083, 2006. <http://eprint.iacr.org/>.
- [19] 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>.
- [20] K. J. Hole, V. Moen, and T. Tjstheim. Case Study: Online Banking Security. *IEEE Security and Privacy*, 4(2):14–20, 2006.
- [21] M. Jakobsson, S. Myers, and M. Augiere. Delayed Password Disclosure, 2005. <http://www.informatics.indiana.edu/markus/stealth-attacks.htm>.
- [22] B. W. Lampson. Accountability and Freedom. <http://research.microsoft.com/~risaacs/blampson.ppt>, Oct 2005.
- [23] C. E. Landwehr. Green Computing. *IEEE Security & Privacy*, 3(6):3, Nov/Dec 2005.
- [24] E. Levy. Criminals Become Tech Savvy. *IEEE Security and Privacy*, 02(2):65–68, 2004.
- [25] J. Liedke. On Microkernel Construction. In *15th ACM Symposium on Operating System Principles*, 1995.
- [26] 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.
- [27] Microsoft Corp. Secure Startup - Full Volume Encryption: Technical Overview. http://www.microsoft.com/whdc/system/platform/pcdesign/secure-start_tech.mspx, April 2005.
- [28] 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).

- [29] B. Parno, C. Kuo, and A. Perrig. Phoolproof Phishing Prevention. In *Financial Cryptography*, 2006. (to appear).
- [30] 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.
- [31] J. D. T. Rachna Dhamija and M. Hearst. Why Phishing Works. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI2006)*, 2006.
- [32] A.-R. Sadeghi and C. Stüble. Property-based Attestation for Computing Platforms: Caring about properties, not mechanisms. In *NSPW*, pages 67–77, 2004.
- [33] 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.
- [34] Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 85, Trusted Computing Group, Feb. 2005.
- [35] Trusted GRUB. <http://trustedgrub.sourceforge.net>.
- [36] J. Tygar and B. Yee. Dyad: A System Using Physically Secure Coprocessors, 1994. Technical Report CMU-CS-91-140R.
- [37] W. Wang, Y. Yuan, and N. Archer. A Contextual Framework for Combating Identity Theft. *IEEE Security and Privacy*, 4(2):30–38, 2006.
- [38] 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.
- [39] Z. E. Ye and S. Smith. Trusted Paths for Browsers. In *USENIX Security Symposium*, pages 263–279, 2002.
- [40] K.-P. Yee. Designing and Evaluating a Petname Anti-Phishing Tool, 2005. <http://cups.cs.cmu.edu/soups/2005/2005posters/23-yee.pdf>.
- [41] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King. Automated Web Patrol with Strider Honey-Monkeys: Finding Web Sites That Exploit Browser Vulnerabilities. In *Network and Distributed System Security (NDSS) Symposium*, 2006.