TruWallet: Trustworthy and Migratable Wallet-Based Web Authentication

Sebastian Gajek
gajek@post.tau.ac.il

School of Computer Science
Tel Aviv University, Israel

Hans Löhr, Ahmad-Reza Sadeghi,
Marcel Winandy
{hans.loehr, ahmad.sadeghi,
marcel.winandy}@trust.rub.de

Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany

ABSTRACT

Identity theft has fostered to a major security problem on the Internet, in particular stealing passwords for web applications through phishing and malware. We present TruWallet, a wallet-based authentication tool that improves previous solutions for protecting web-based authentication. In contrast to other wallet-based solutions, TruWallet provides (i) strong protection for users' credentials and sensitive data by cryptographically binding them to the user's platform configuration based on Trusted Computing technology, (ii) an automated login procedure where the server is authenticated independently from (SSL) certificates, thus limiting the possibility of attacks based on hijacked certificates and allowing less dependency on the SSL PKI model, and (iii) a secure migration protocol for transferring wallet data to other platforms. Our implementation uses a small virtualization-based security kernel with trusted computing support and works with standard SSL-based authentication solutions for the web, where only minor modifications and extensions are required. It is interoperable so that we can re-use existing operating systems and applications like web browsers.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; D.4.6 [**Operating Systems**]: Security and Protection—*Authentication, Security Kernels*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Security*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication, Unauthorized access*

General Terms

Security

Keywords

Identity theft, phishing, password wallet, trusted computing, secure migration

1. INTRODUCTION

Identity theft has become one of the fastest growing crimes on the Internet, leading to huge financial losses and privacy concerns [2, 5] because of rising online fraud [22] and software attacks [45]. Among the most prominent attacks are phishing and pharming, where users are lured to faked sites and asked to disclose their identity credential information. Attackers exploit the fact that the average Internet user is unable to distinguish a legitimate site from a fake one—even though browser vendors introduced improved user interfaces [11]. Moreover, attackers benefit from weak issuing policies of certificate authorities [33] or the use of over-aged cryptographic algorithms to generate SSL certificates [47].

An additional powerful class are cross-site-scripting, request forgery or *malware attacks*. They compromise and infiltrate the user's computing platform with malicious code (e.g., browser scripts, keyloggers or transaction generators) [3, 29, 30, 40]. Commodity operating systems do not mitigate the impact of these attacks appropriately. They suffer from various conceptual shortcomings. Beside architectural security problems and the inherent vulnerabilities resulting from high complexity, today's operating systems require careful system administration skills that ordinary users typically do not have.

Several approaches have been proposed to address the mentioned attacks. Delegated identity management systems exist, where the user calls a trusted third party in form of a distributed server for hosting and providing identity information. Examples include Google's single-sign on and Microsoft's Cardspace protocols. However, they turned out to have deficiencies [7, 15, 20, 31, 39].

On-board credentials (ObC; cf. [32]) are another recent approach to address these threats. ObC use hardware features (such as Trusted Computing technology) to protect credentials and provide a promising and flexible credential system, in particular for mobile devices. However, the model assumes that secret data is "provisioned" explicitly for this system. It is not obvious how ObC could be used to provide secure password-based authentication for existing web services with legacy general-purpose browsers; this approach seems more suitable to provide a robust security solution for novel web services that are developed explicitly for ObC.

Against this background, wallet-like approaches have gained attention (e.g., [14, 25, 26, 27, 34]). A wallet calls an authentication agent in an isolated trusted environment to separate the handling of credentials from the normal web browsing (see also [42, 51]). Most of the existing proposals counteract specific attacks. They do not provide a general approach to protect against all attacks in the wild (see Section 7 for more discussions). However, a desired property is that the user relies on a self-contained solution that protects against any threat of identity theft—be it a phishing or be it malware attack.

To protect average Internet users against these threats, we present the design and implementation of *TruWallet*, a wallet-based approach for secure web authentication. TruWallet consists of (i) a trusted wallet acting as web proxy to perform the user login at web sites, and (ii) a security kernel that provides a secure environment for the wallet and a secure user interface as trusted path between the user and the wallet. As our main contribution, we address the mentioned objectives as follows:

- We propose a method to achieve an SSL-PKI-independent login, assuming trustworthy SSL certificates only during the registration phase (Section 3). Our protocols establish a shared secret between TruWallet and server during registration which is then used for server authentication during login.
- We present an efficient and secure migration protocol for the wallet data using trusted computing functionality based on Trusted Platform Modules (TPMs) [50] and security services interfacing the TPM (Section 4). Our protocol allows the user to securely transfer the secrets to a wallet on another platform in order to access web sites from there.
- As a proof of concept, we describe our reference implementation based on a microkernel architecture that supports virtualization and trusted computing functionality (Section 5). We are able to re-use existing operating systems (Linux in our prototype) and applications like commodity web browsers.

Our approach has the advantage that the user may install arbitrary software in the same security context the browser is executed (because the wallet runs in an isolated, i.e. protected execution environment). TruWallet works with standard SSL-based authentication solutions for the web except that minor modifications and extensions on the server-side are necessary. TruWallet, including the security kernel, is based on open source and can be installed on x86 PCs equipped with a TPM, which are already included in many computers of various vendors.

2. SYSTEM OVERVIEW

In this section we describe the threats and security objectives, and give an overview of TruWallet's architecture and its usage.

2.1 Threats and Security Objectives

TruWallet addresses the following threats:

T.Phishing Classical phishing and pharming attacks lure the user to faked web sites;

T.WeakPol Weak issuing policies of SSL certificates or choice of weak cryptographic algorithms allow the attacker to retrieve a valid SSL certificate;

T.XSS Cross-site scripting or cross-site request forgery attacks exploit server vulnerabilities in order to inject malicious script code into the browser;

T.Malware1 Malicious software on client system compromises system components and applications;

T.Malware2 Malicious software makes unauthorized use of system components and applications; For example, the malware could wait for the user to log in at a web site and subsequently generate fake transactions.

T.Access Unauthorized access of the client system where the attacker impersonates a legitimate user by misusing active browser sessions or stealing credentials.

To address the threats, we need a comprehensive solution that protects both the web authentication mechanism and the software running on the client system. Our main objectives are therefore:

O.ProtectPW Passwords must be unique for each site, resistant to dictionary attacks, and protected from unintentional disclosure;

O.SecExec Secure execution environment for security-critical components and a trusted path to the user to prevent malware attacks;

O.SecStorage Secure storage environment for credentials when the system is offline (i.e., powered off);

O.LessCert Reduced dependencies from SSL certificates to mitigate weak issuing policies of certificate authorities;

O.SecMigrate Secure migration of credentials among different platforms, where the migration mechanism must ensure that each platform complies to the user's security policy.

2.2 Introduction to TPM and TCB

The main component of trusted computing as specified by the Trusted Computing Group is the TPM chip [50]. The TPM provides a secure random number generator, non-volatile tamper-resistant storage, key generation algorithms, cryptographic functions like RSA encryption/decryption, and the hash function SHA-1. It protects a variety of keys. Two of its main asymmetric keys are the *Endorsement Key* (EK), an encryption key that uniquely identifies each TPM, and the *Storage Root Key* (SRK) uniquely created inside the TPM. The private SRK never leaves the TPM, and it is used to encrypt all other keys created by the TPM. The TPM state contains further security-critical data shielded by the TPM. Amongst them is a set of registers called *Platform Configuration Registers* (PCR) that can be used to store hash values. During the platform boot the hardware and software constellation of the platform is hashed and securely stored in specific PCRs which constitute the platform configuration (integrity measurement). Moreover, the TPM allows to report the configuration of a platform to a remote party (*attestation*), and to cryptographically bind data to a certain system configuration, i.e., a subset of the PCRs (*sealing*). Sealed data can only be accessed (unsealed) if the corresponding system can provide the specific configuration for which the data has been sealed. During attestation the recorded PCR values are signed by an *Attestation Identity Key* (AIK) of the TPM and send to the remote party. The AIK plays the role of a pseudonym of the TPM's identity EK. To be authentic, the AIK must be certified, e.g., by a trusted third party called privacy-CA. It is important to note that this privacy-CA is conceptually simpler than a SSL-certificate authority. The TPM ist trusted to function correctly, and to never disclose secrets, such as the private EK, SRK, and AIKs. A TPM works in conjunction with software components that are critical to its security. We call the combination of all trusted components in the system the *trusted computing base* (TCB). Modern operating systems strive to reduce the size of the TCB so that an exhaustive examination of its code base (by means of manual or computer-assisted software audit or program verification) becomes feasible.

2.3 Architecture

Our system model consists of several parties (see Figure 1): A *user* interacts with a computing platform through a secure graphical user interface *secure GUI*. A *browser* is used to render web pages that it gets from the *wallet*, which is acting as a proxy. The wallet obtains the requested pages from the server, blinds security-sensitive fields (e.g., password) on the pages presented to the browser, and fills in login credentials when logging into a website. For this, TruWallet has to handle two different SSL sessions: one between wallet and browser, and one between wallet and server. The secure GUI controls the input/output devices and multiplexes the screen output of the browser and of the wallet. Moreover, it always indicates the name of the application the user is currently interacting with via a reserved area on the screen, hence providing a *trusted path* between user and application.

The TruWallet architecture is based on a *security kernel*, which is a small trusted software layer belonging to the TCB, providing *trusted services* and *isolated compartments*. Thus, the security kernel ensures runtime security of the system. Compartments contain arbitrary software, e.g., a complete legacy operating system (Linux in our case), and may communicate only via well-defined interfaces. In particular, a malicious compartment cannot read arbitrary memory of other compartments. In our solution, browser and wallet run in different compartments, and we assume that arbitrary software (including malware like Trojan horses and viruses) may be running in the browser compartment. Hence, the browser is assumed to be untrusted and any security-enhancing tools based on browser plugins may be modified or deactivated. Therefore, our solution is based on a trusted component (wallet) that is executed in a separated compartment. In our implementation, we realize the compartmentalization by using the isolation property of virtual machines combined with the resource sharing control of the underlying microkernel. The wallet compartment is trusted, which is motivated by the fact that the complexity of the wallet is much lower than that of a web browser. Moreover, the user cannot install arbitrary software (which may be malicious or flawed) in the wallet compartment. To prevent unauthorized access by other users to the platform and, hence, the sensitive data, the security kernel requires an overall user authentication (e.g., a user password) to login into the whole system. In this way, the credentials stored by the wallet are bound to the corresponding user.[1]

Trusted Computing (TC) hardware and TC-enabled software is used to provide *trusted boot*, i.e., based on a "chain of trust", the integrity of the software stack including the TCB can be verified during data migration. Moreover, TC hardware can be used for secure storage, i.e., encryption keys protected by the hardware can only be used if load-time integrity of the system is maintained. As already mentioned before, our implementation uses a TPM as TC hardware (see Sections 4 and 5). The credentials stored by the wallet are bound to the TCB to prevent an adversary from gaining access to the data by replacing software (e.g., booting a different OS).

An alternative way of using TC hardware would be to execute the trusted software via hardware-based dynamic root of trust, using hardware support of modern CPU architectures [4, 21]. Instead of measuring the whole software stack at boot time, the CPU allows to execute trusted code in a special mode where integrity measurements are taken dynamically and stored in the TPM until the trusted code execution ends [37]. In principle, such an approach reduces the amount of needed TCB code. However, when the trusted code is executed, the remaining system code (the operating system, browser, etc.) is halted until the trusted code terminates. This mechanism is intended to (repeatedly) execute small trusted code pieces and resume again. We did not choose dynamic root of trust in our design because the trusted code does not only need to insert passwords (which would be a relatively small functionality), but we also need to authenticate the server, verify the authenticity of the SSL channel, and provide a trusted GUI so that the user can always be sure about the application interacting with. Thus, significant parts of the system need to execute very often or in parallel, which would introduce a noticeable performance overhead when using dynamic root of trust. This becomes of even more importance when the wallet is enhanced with additional functionality like transaction confirmation, where trusted code needs to inspect the network traffic continuously.

[1]In fact the security kernel has to provide comprehensive user access control as in typical operating systems, including system login and screen lock functionality, in order to prevent unauthorized access to the wallet. However, the details of those mechanisms are out of scope of this paper.

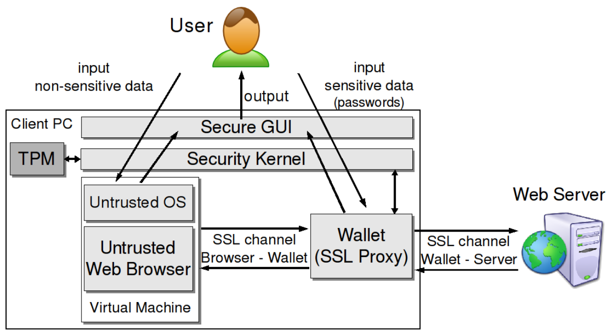

Figure 1: Architecture of TruWallet

2.4 Assumptions

Our solution is based on the following assumptions: First, the user is trained to enter credentials only into the wallet (single credential store mechanism). In our implementation, the user can establish a trusted path to the wallet by pressing a secure attention key. Second, the wallet can rely on a PKI during registration at a server (minimal PKI). A PKI is needed to prevent attacks like DNS-spoofing during the setup/registration process. The wallet relies on a correct SSL certificate to identify a remote site. However, we aim at minimizing this assumption: to protect sensitive user data during a later session, it is important to highlight that we do not rely on a trustworthy SSL-PKI.

2.5 Usage Overview

Registration of a new account at a website with the wallet works almost the same way as in existing approaches, with two differences: first, the user enters sensitive data (e.g., passwords) only into TruWallet (never into the browser); second, TruWallet generates a high-entropy password that is unique for this account. To log into a previously registered site, the user just opens the login page in the browser and clicks on the "login" button. TruWallet is acting as man-in-the-middle proxy and automatically checks if the server is authorized to obtain the credentials before it fills in the login credentials on behalf of the user (for details, see Section 3). Note that, since the wallet is also an SSL proxy, several SSL connections (and hence user logins) can be handled simultaneously, e.g., when the user opens several websites in multiple browser tabs or instances. Each SSL connection between a browser tab or instance has a corresponding SSL connection between the wallet and the respective web server.

To enable the user to access web sites from another computer, we propose a secure migration scheme for the wallet to transfer or synchronize its data with a trusted wallet running on the other device. We use TC functionality to establish a trusted channel ensuring the integrity of the target system to transfer the data (Section 4).

3. SECURE USER AUTHENTICATION

In the following, we describe a scheme for registration and login. It follows the idea of SSL-session awareness [38]. Our schemes rely on SSL certificates only during registration, and require only minimal changes of client and server.

Registration.

When the user registers at a website for the first time, a setup step is needed to enable logins that are independent from SSL certificates later on. Note that this is the case when the wallet first learns about the site and creates a corresponding login/password entry. Hence, this step is completely transparent to the user.

We require that a SSL connection is used for registration, and that the wallet verifies the SSL certificate of the remote site.[2] We use the SSL server finished message to infer an additional shared secret $ss := SERVER\_FINISHED$. The server finished message is derived by computing the hash of the protocol transript (provided the protocol did not abort). A transcript $trnscrpt$ includes all the messages the wallet has received from and sent to the server, respectively. It is important to note that the SSL protocol requires to authenticate and encrypt the finished value, using the derived session key. No attacker perceives the finished message in plain text. At the end of the registration, wallet and server store the shared secret ss securely.

Login.

Our login protocol uses SSL only to provide a secure (confidential) channel. For the crucial server authentication, the shared secret ss from the registration protocol is used. The loging proceeds as follows: The server $\mathcal{S}$ authenticates in challenge-and-respond protocol by proving knowledge of the shared secret ss. For ease-of-implementation, we utilize the SSL transcript $trnscrpt$ as a challenge. Our approach makes use of the fact that the transcript includes a randomly chosen nonce from client. In fact, it would be sufficient to use this nonce alone without security loss. The server answers to the challenge by computing the response $R := \mathrm{HMAC}_{ss}(trnscrpt)$, where $\mathrm{HMAC}()$ denotes a keyed-hash message authentication code, ss the shared secret key between wallet and server, and $trnscrpt$ the login transcript of the SSL protocol. Without any modification to the native SSL implementation, the response message R is postponed to the last server message, i.e., the server finished message of the SSL handshake protocol.

The wallet runs the SSL protocol in the normal way and aborts, if the protocol does so. Next, the wallet verifies that R is a valid response. If $R \neq \mathrm{HMAC}_{ss}(trnscrpt)$, it aborts the protocol. Otherwise, authentication proceeds as usual (username and password are transmitted).

Security.

The use of the SSL SERVER_FINISHED to derive the shared secret ss ensures that no adversary can compute ss. It can be shown that computing the finished value is reducible to the security of the SSL protocol. Furthermore, the server's response in the login protocol cannot be forged by an adversary (even if the adversary managed to obtain a valid SSL certificate and acts as a man-in-the-middle) because an HMAC keyed with ss is used to tie the authentication to the SSL sessions. Man-in-the-middle attacks are not possible in the login protocol—even if the adversary managed to obtain a valid certificate—because the check of the server's response performed by the wallet succeeds only if both parties use the same shared authentication secret ss. This is only possible for the two endpoints of the SSL channel. These considerations imply that the wallet only executes the normal (password-based) login with a party that possesses the shared secret which was computed during registration, and hence only the server where the user registered can obtain the user's credentials. Attackers cannot re-compute ss, because for this, they need access to the ephemeral secrets from the SSL handshake during registration, which were known only to the two (trusted) endpoints: TruWallet and the web server. Thus, even a full compromise of the SSL PKI does not help to recover ss.

Discussion.

Essentially, any password-authenticated key exchange (PAKE) protocol where the server proves knowledge of the password could be used instead of our registration and login protocols, as long as it is used in a way that ensures resistance against man-in-the-middle and replay attacks (see, e.g., [9, 24, 52]). PAKE protocols can be used for mutual authentication, i.e., using PAKE, it is not necessary to transmit username and password over the SSL channel during login. However, note that not all authors explicitly discuss and analyze the important (for our case) requirement that the *server* proves knowledge of the password. In particular, the simple remote password (SRP) protocol [52] is well-suited for our purpose, and RFC5054 [49] specifies the use of SRP for SSL authentication. A detailed security analysis (including a proof) is provided in [52].

Our custom protocols are easier to implement than a PAKE protocol because only minor modifications to existing software are needed. Both, the SSL handshake and the login procedure remain unchanged in our protocols, whereas PAKE protocols require a dedicated key exchange protocol. However, a complete implementation of RFC5054 as a standard-compliant authentication solution might be beneficial in many scenarios and could be the preferred approach for a production-quality implementation.

4. SECURE WALLET DATA MIGRATION

To protect the confidentiality of the wallet data on persistent storage, the wallet data is sealed, i.e., encrypted with a key that is bound to the configuration (integrity measurements) of the wallet's TCB and shielded by the TPM. If the user wants to use the wallet on another platform, e.g., in order to switch to another machine or because of hardware replacement, the sealed data would become inaccessible. We also do not want to expose the wallet data to a platform the user does not trust. Hence, we need to check the "trust status" of the target platform before we re-bind the wallet data.

A *trusted channel* is a secure channel (i.e., authenticity, integrity and confidentiality) with the additional feature that it is bound to the configuration of the endpoint(s). The idea is to embed an attestation of the involved endpoint(s) in the establishment of the secure channel [18, 48]. Hence, each endpoint can get an assurance whether the counterpart complies with trust requirements before the secure channel is settled.

Besides the wallet, we consider two trusted components of the security kernel to take part in the migration procedure: (1) the *StorageManager* provides an abstraction of persistent storage to

[2]At this point, we have to trust the certificate. Later on, we are independent from any changes/updates of the SSL certificate. Note, there exist no authenticated protocols without setup assumptions. Yet, this is an open research problem. We stress the fact that in this paper the gooal is to minimize the setup assumptions.

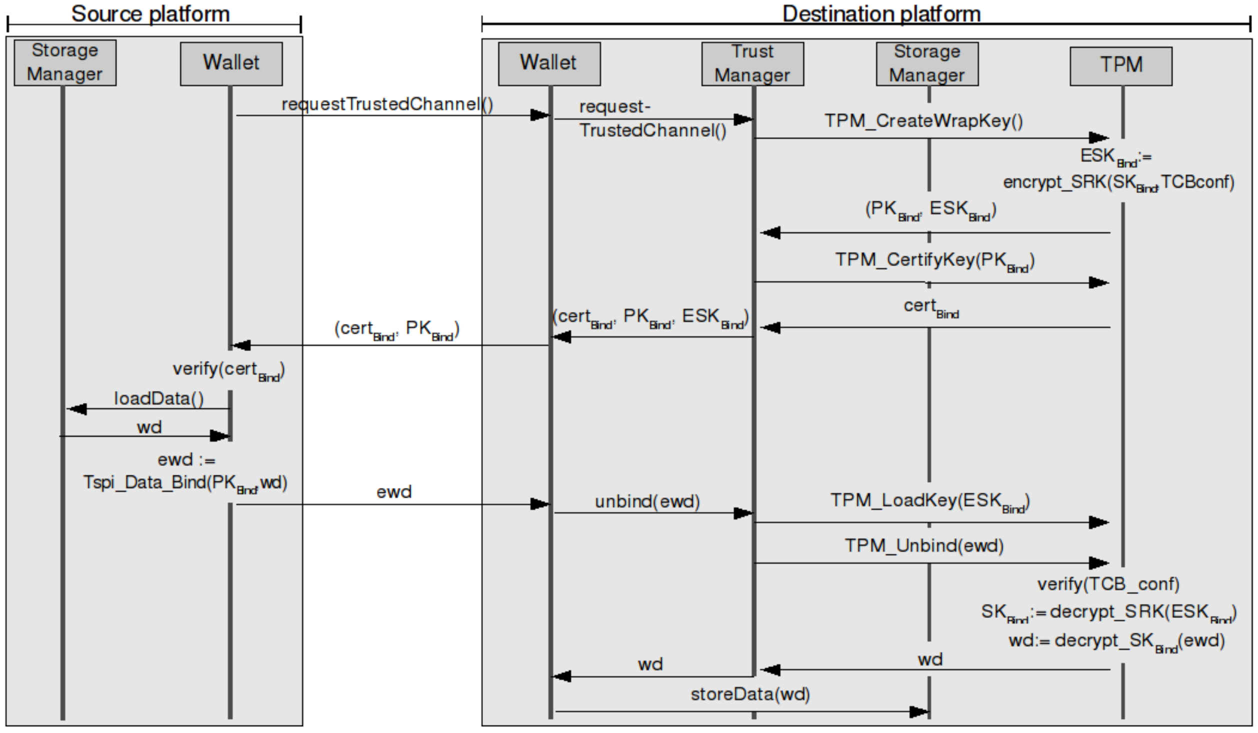

Figure 2: Migration of the wallet data based on a trusted channel

other compartments, here especially the wallet, and enforces additional security requirements to protect confidentiality and integrity by means of encryption and sealing. (2) the *TrustManager* provides an abstraction of the functionality of a TPM and is the only software component that directly communicates with the TPM.

When the user wants to migrate or copy the wallet data from one platform (source) to another (target), then a trusted channel between the two platforms, i.e., between two wallet instances, must be established (see Figure 2). The wallet on the target platform calls the TrustManager of its platform to initiate the trusted channel. The TrustManager uses the TPM to create a new asymmetric key pair as binding key (PK_{Bind}, SK_{Bind}). The TPM creates the key pair, encrypts the private key part with the SRK, and returns PK_{Bind} and the encrypted private key ESK_{Bind}. Note that decryption of the private key is now bound to the configuration of the TCB as measured during the boot process and represented by the values of the PCRs of the TPM, i.e., TCB_conf. Moreover, the TrustManager requests the TPM to sign the public key with an AIK in order to create a certificate $cert_{Bind}$. The TrustManager returns $cert_{Bind}$, PK_{Bind} and ESK_{Bind} to the wallet, which then sends the binding key PK_{Bind} and the certificate $cert_{Bind}$ to the source platform.

The wallet on the source platform can now verify the certificate and decide whether the target platform complies to the wallet's trustworthiness requirements. Therefore, it needs the public key of the AIK of the target platform and the certificate from the Privacy-CA. Then, the source wallet loads its data from the StorageManager and encrypts the data wd with the binding key PK_{Bind}. The encrypted wallet data ewd is sent to the target wallet, which then requests the TrustManager to unbind ewd. Therefore, the TrustManager first calls the TPM to load the key ESK_{Bind} into the TPM, and then requests the TPM to unbind ewd. The TPM in turn verifies with `verify(`TCB_conf`)` whether the TCB configuration (i.e., its integrity measurement) is the same as at creation time of the binding key. If this is the case, the TPM proceeds and first decrypts ESK_{Bind} to retrieve the private part of the binding key, which it uses subsequently to decrypt the wallet data. The decrypted data wd is returned to the TrustManager and the wallet, respectively.

Discussion.

Once a trusted channel is established, it can be re-used for subsequent wallet synchronization between the platforms. The binding of the key pair to the configuration of the TCB guarantees that no other platform and even no modified system booted on the same platform can decrypt the key and, hence, the wallet data. However, for each device the user wants to use TruWallet, the user has to (i) establish a trusted channel, and (ii) if the TCB or the wallet have to be updated, the trusted channel has to be re-established. In principle, it is not necessary that both platforms are online for migration. The binding certificate can be computed beforehand. Of course, the user has to know the potential destination platforms a-priori, but can transfer the data on offline storage, e.g., on a memory stick.

Requiring a certificate for the AIK from the Privacy-CA introduces another PKI dependency, but we expect very few AIK certificates (typically one to four TPMs for the platforms the user wants to use, e.g., laptop, home PC, office PC, mobile device). However, the number of certificates for usual SSL server authentication is much higher (e.g., 10–20 web sites or more used by one user).

5. IMPLEMENTATION

Our implementation is built on top of the Turaya security kernel[3], which is based on an L4 microkernel [35]. Besides virtualization, Turaya provides TC support based on a TPM and a secure GUI. Moreover, using Turaya allows us to re-use and extend the open source Wallet-Proxy developed by Gajek et al.[14]. Figure 3 shows our implementation architecture.

[3]`www.emscb.org/content/pages/turaya.htm`

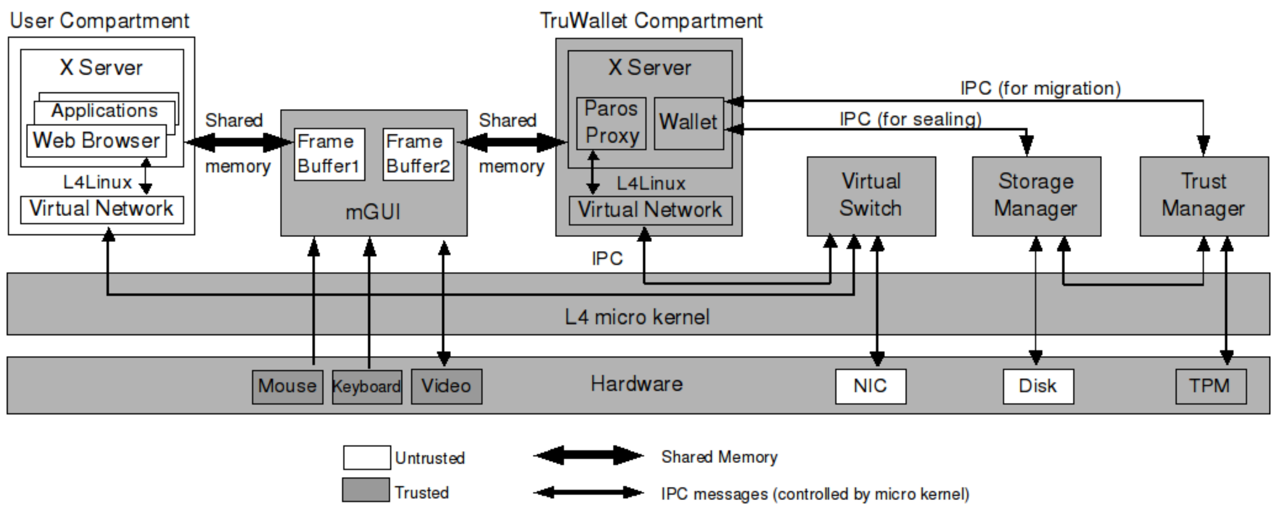

Figure 3: Implementation of our wallet-based architecture

In our prototype, the *user compartment* contains a Linux OS with arbitrary applications chosen by the user. As web browsers, we use Opera and Firefox. The *TruWallet compartment* is realized based on a minimally configured Linux OS. Only the wallet (implemented in Java) and the proxy (using Paros [1], a Java-based HTTP/HTTPS proxy) are running, whereas the Linux system contains only functionality that is needed by these applications (i.e., Java runtime environment, shell scripts, and a simple X server for GUI dialogs). In contrast to the user compartment, the user is not allowed to install applications into the wallet compartment.

The TrustManager is also a minimally configured Linux compartment, which contains a TPM driver and an application providing an interface to the TPM, e.g., used to generate a certificate for the trusted channel of the wallet data migration. The StorageManager is an L4 process which encrypts data from other compartments and seals the data to the integrity measurements of the TCB, using the sealing function of the TPM via TrustManager. In particular, the integrity measurements of the TCB include all trusted software components as depicted in Figure 3.

Our secure user interface is realized with a minimal GUI service *(mGUI)*, which is a native L4 process that has full control over the keyboard, mouse, and video hardware. The mGUI provides a trusted path between the user and the TCB. Implementation mechanisms are out of scope of this paper since related work such as Nitpicker [13] covers details extensively.

TruWallet associates the login data with the URL of the login form of the corresponding web site. This association has to be set up in the wallet. Whenever the wallet detects a password field and has no matching login data, it displays a dialog in its compartment.

Registration.

In practice, different methods are used to register users at a service. Users obtain an initial password via an out-of-band channel (e.g., by email), they might want to set up an existing account for use with TruWallet, or the initial password is entered in a registration form prior to the first login.

To register a new account, the user enters non-sensitive data (login name) in the browser. The wallet recognizes password fields in the registration form and indicates (via the secure GUI) that the user has to switch to the wallet compartment by pressing a secure attention key. In case the user already possesses a password, the user enters the password into a special input form displayed by the wallet. Otherwise, the wallet generates a new password. The wallet submits the complete data to the server, and the user can switch back to the browser and use the web site. If an existing password is used, TruWallet reminds the user to initiate a password change as soon as possible (see Section 5).

In contrast to existing password managers, which in such cases associate the URL of the registration form with the login data, TruWallet offers the user a checkbox to indicate registration forms. At the next login, the user can associate the login data stored at registration time to the actual URL of the login form.

Login.

Now that the user has registered and the wallet setup for this site is complete, the password is filled in automatically whenever the user logs in. The wallet recognizes the URL of the login form, thus, whenever users log in, they just enter their username (the password form should be blocked) and click on the "submit" button. The wallet fills in the password automatically based on the URL and username. Hence, TruWallet supports multiple accounts at one web site.

Password Change.

As proposed in [14], TruWallet must change the user's password to a unique high-entropy secret. The new password should not be known to the user because otherwise the user could be tricked to disclose it unintentionally to a phishing site. For this, TruWallet sets the new password to $pw_{new} := \text{HASH}(pw_{old}, r)$, where pw_{old} is the password chosen by the user and r is a random value. However, a fully automated generic approach to change passwords which works for any website is unrealistic, given the enormous variety of different procedures employed on the Internet.

We conducted a survey on popular websites – including several banking sites, Amazon, eBay, PayPal, Walmart, and others – which led to the conclusion that a simple heuristic for finding out how to change passwords (such as looking for links with a specific text, like "change password") is not sufficient. The difficulties finding the right links and URLs include: HTML-tags enclosing parts of the actual text link (e.g., for emphasis), complex framesets partially generated dynamically using JavaScript, multi-column forms with multiple options for the user, forms where users have to prove their identity by inserting additional information (different from the old password), etc.

In our approach, the user has to initiate the password change on the web site and switch to the wallet compartment. TruWallet displays a special input form offering two options for each password field: "generate new password" and "fill in old password". The user selects the appropriate options, and the wallet submits the completed form to the server. TruWallet stores the new login data.

Migration.

We have implemented the essential steps of the wallet migration and tested it on two PCs with identical configuration (i.e., both running the same security kernel and equipped with a TPM 1.1). Currently, our prototype requires manual initiation of the migration at the target (trusted channel establishment) and manual transfer of the encrypted wallet data (e.g. via USB memory stick).

6. SECURITY OF TRUWALLET

Our solution realizes objective O.ProtectPW by generating new high-entropy passwords not known to the user. Moreover, the wallet verifies the legitimacy of the server before disclosing the password. This mitigates threat T.Phishing.

Assuming SSL certificates were trustworthy during initial registration, we provide security against certificate hijacking during any later login. This means that our login procedure is resistant against T.WeakPol by following O.LessCert. Our approach works without modification of the SSL handshake protocol. Hence, the security properties of SSL remain unchanged.

To address the threat T.XSS, we isolate the usage and storage of credentials from the browser. The wallet performs all tasks related to establish authenticated connection to a web server, including storage and management of session cookies. Hence, attacks such as cross-site scripting and cross-site request forgery, which attack the browser, cannot obtain these credentials.

The security kernel and the use of trusted computing functionality realize O.SecExec and O.SecStorage. During runtime, the security kernel isolates compartments by default and allows only controlled inter-process communication. Thus, malware running in one compartment cannot read or modify the data and configuration of other compartments. The TCB cannot be compromised during runtime because of the assumption that it is small enough to be verified and tested thoroughly, and based on the fact that the user cannot install arbitrary applications within trusted compartments. Hence, the only "entry-point" for malware is the user compartment, which is isolated from the wallet compartment. TruWallet only routes network traffic between the browser and network. If software components are modified in order to compromise the system, the modification of trusted components will result in different integrity measurements taken during the boot process and recorded in the TPM. Thus, the wallet data cannot be unsealed and remains encrypted. The security in this case is based on the security of the encryption and the protected key storage of the TPM. This addresses T.Malware1 on the local platform.

Moreover, our secure migration protocol for copying or transferring wallet data to another device realizes O.SecMigrate to address T.Malware1 on target platforms. TruWallet ensures that the target provides at least the same security properties as the source platform – reflected by the integrity measurements verified during the attestation of the target system. Transaction security can be provided to protect sensitive data after the login process by incorporating transaction confirmation functionality [26, 27, 34] into the trusted wallet compartment. Our modular architecture allows for an easy integration of such extensions that would mitigate T.Malware2. The threat T.Access can be prevented by typical user access control (system login at start-up, screen lock when leaving the computing device).

7. RELATED WORK

Password wallets and trusted virtualization infrastructures are the most relevant related works, which we discuss in this section.

Wallet-Based Web Authentication.

Wu et al. [51] introduce *Web Wallet*, which is a browser extension and 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 she wants to enter sensitive data. The wallet checks the destination site using SSL certificates and other indicators (e.g., site popularity) and if unsure asks the user to explicitly choose the destination site from a list. Although this approach reduces the risk of classical phishing attacks, it does not isolate the wallet from the browser and hence lacks protection against malware that modifies the wallet or fakes its user interface.

Jackson et al. [25, 26] present SpyBlock, a browser extension that requests authentication as well as confirmation of transactions from the user by calling a separated confirmation agent. The browser runs in a virtual machine and, hence, virtualization is used to isolate the agent from the browser and to provide a trusted path to the user. However, they use a type 2 virtual machine monitor (VMM)[4] and the agent runs on the host OS (Windows Vista), and they have no further protection of the agent, i.e., no secure storage like binding the agent data to the platform configuration. Thus, if the host OS gets compromised, malware may be able to manipulate the agent. In our approach, the wallet runs on a small (minor complexity) security kernel and we use a TPM [50] to seal the wallet data to the platform and its TCB. Moreover, SpyBlock does not realize O.ProtectPW because it uses password hashing based on a master password and the domain name of the web site, which a phisher can compute if the user is tricked to enter the master password somewhere. The master password is used to derive all other passwords. Thus, for migrating the authentication data to other platforms, users would need to disclose the master password to all platforms they desire to use, which is a security risk if a platform is compromised. SpyBlock uses a shared secret key between the agent and the web site to compute a MAC of the transaction based on the shared secret which the server can verify, and cryptographically binds the result of a password-authenticated key exchange (PAKE) protocol to the SSL channel, which could be used to obtain a login that is independent from the SSL-PKI (cf. Section 3). However, this does not seem to be a goal of the authors.

Delegate [27] is a web proxy to store credentials and to authenticate to web sites on behalf of the user. The web proxy is running on a different machine and not on the same device the user runs the browser. To realize the trusted path, they use a trusted mobile phone. Thus, they use physical isolation to separate the browser from the authentication part. However, this approach requires users to have an extra device (the phone) besides their PC and an online connection to the proxy for each login request. In contrast to their approach, TruWallet uses a TPM as trusted device attached to the platform the user operates on and a security kernel to isolate browser and credentials on the same platform. Hence, users need to rely only on one platform to perform web authentications.

[4]According to [17], a type 1 VMM executes directly on hardware, whereas a type 2 VMM needs a host operating system to run on. VMWare Workstation is an example for a type 2 VMM, and Xen [12] a type 1 VMM (also called *hypervisor*).

Approach	Strong Passwords	Web Site Verification	Isolation	Trusted Path	Secure Storage	Less PKI Dependency	Trusted Migration
Web Wallet [51]	no	yes (checking certificates etc.)	no	no	no	no	no
Delegate [27]	no	no (user checks servername)	yes (different physical machines)	yes (using mobile phone)	no	no	partly (usable from any machine)
SpyBlock [25, 26]	no	yes (using PAKE)	yes (type 2 VMM)	yes (reserved area)	no	partly (using PAKE)	no
Vault [34]	no	no (user checks servername)	yes (type 1 VMM)	yes (secure attention key)	no	no	no
Wallet-Proxy [14]	yes	yes (using SSL certificates)	yes (type 1 VMM)	yes (reserved area + secure attention key)	yes (TPM sealing)	no	no
Our approach (TruWallet)	yes	yes (proving shared secret)	yes (type 1 VMM)	yes (reserved area + secure attention key)	yes (TPM sealing)	yes (establish shared secret)	yes (trusted channel)

Table 1: Comparison of wallet-based approaches

Kwan and Durfee [34] define a protocol framework for the interoperation between an untrusted compartment and a trusted compartment (Vault) handling sensitive user data. They use virtual machines and a type 1 VMM to isolate the compartments and to provide a trusted path to Vault. However, they do not address classical phishing directly because users have to enter passwords into Vault at each login and verify the legitimacy of web sites on their own.

Gajek et al. [14] present Wallet-Proxy, a trusted wallet to store passwords and to automatically perform the login on behalf of the user. They propose to replace the user-provided password with a hash of the original password concatenated with a random value. However, they do not present details of how the wallet can define a new password for existing or newly registered accounts. In contrast to our approach, they heavily rely on a PKI, i.e., SSL certificates, in order to authenticate web servers during the login process. They also use a TPM to seal the wallet data to the platform configuration (integrity measurements of the wallet and its underlying TCB). While sealing provides a secure storage on one platform, our solution allows to securely migrate the wallet data to another machine.

Trusted Computing and Virtualization.

Binding a key to the configuration of the underlying TCB has been realized with TPMs [36] and secure coprocessors [28, 46]. Asokan et al. [8] describe a protocol for a trusted channel to realize license transfer in a DRM scenario. Our trusted migration protocol is a novel application of this license transfer. We require less components and less protocol steps since our trusted channel is not needed to transfer huge amount of media data like in their DRM scenario, and we do not have a freshness requirement, i.e., replaying wallet data is not a security problem in our scenario.

Since we use sealing to protect the wallet data during persistent storage and migration, respectively, an appropriate integrity measurement mechanism during boot-up of the system is essential. Performing integrity checks during the boot process [6], extending the measurement of loaded modules during runtime [44], and binding secrets to integrity measurements [36] are well explored concepts.

Using virtualization to implement secure browsers has been explored by Cox et al. [10]. They isolate different web applications by running instances of web browsers in separated virtual machines. Other works [23, 41, 19] achieve even more fine-grained control by decomposing the web browser functionalities into single processes and applying process protection mechanisms of the operating system. However, browser security is complementary to our work because it does not protect against classical phishing attacks. Our wallet architecture adds a trusted component *besides* the browser to handle web authentication.

Xen [12] is a prominent hypervisor architecture. sHype [43] adds policy-controlled isolation enforcement and resource sharing to Xen. Terra [16] is another VMM architecture using trusted computing functionality to provide attestation of VMs to remote parties. In contrast to these hypervisors, an L4-microkernel-based VMM has the advantage of running small native processes besides VMs, which we use to implement security services like the StorageManager and TrustManager. This reduces the code size of the TCB one has to trust compared to full VM installations.

8. CONCLUSION

We have presented TruWallet, a wallet-based architecture for secure web authentication. TruWallet allows users to automatically authenticate to web sites without revealing their long-term secrets to untrusted applications or faked web servers. It prevents the user from being tricked into revealing credentials by generating high-entropy passwords that even the user does not know. Moreover, we have presented secure registration and login protocols requiring only minimal changes to existing server software. Furthermore, we propose a secure migration protocol to be able to use wallet data on different platforms. Our implementation based on trusted virtualization technology is a proof of concept on PC platforms. Only minor changes in user behavior are required, e.g., entering sensitive data into the wallet instead of the web interface.

In contrast to existing solutions, we (i) achieve the additional requirements of secure storage and trusted migration bound to the state of the wallet and its underlying TCB using trusted computing functionality available in many computing platforms, and (ii) reduce the high dependency on SSL certificates for server authentication. We demand a trusted PKI only for registration when creating a new account at a server, and for establishing trusted migration.

9. REFERENCES

[1] Paros website. `http://www.parosproxy.org`.

[2] ID theft ring hits 50 banks, security firm says, 2005. `http://news.cnet.com/2100-7349_3-5823591.html`.

[3] New Trojans plunder bank accounts, 2006. `http://news.cnet.com/2100-7349_3-6041173.html`.

[4] AMD. AMD64 architecture programmer's manual volume 2: System programming. Technical Report Publication Number 24593, Revision 3.14, AMD, Sept. 2007.

[5] Anti Phishing Working Group. Phishing Activity Trends Report(s), 2005-2008. `http://www.antiphishing.org`.

[6] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *IEEE Symposium on Security and Privacy (S&P'97)*, pages 65–71, Oakland, CA, May 1997. IEEE Computer Society.

[7] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. T. Abad. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In *6th ACM Workshop on Formal Methods in Security Engineering (FMSE'08)*, pages 1–10, Hilton Alexandria Mark Center, Virginia, USA, 2008. ACM.

[8] N. Asokan, J.-E. Ekberg, A.-R. Sadeghi, C. Stüble, and M. Wolf. Enabling fairer digital rights management with trusted computing. In *10th Information Security Conference (ISC'07)*, LNCS vol. 4779, pages 53–70. Springer, 2007.

[9] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symposium on Security and Privacy (S&P'92)*, pages 72–84, 1992.

[10] R. S. Cox, J. G. Hansen, S. D. Gribble, and H. M. Levy. A safety-oriented platform for web applications. In *IEEE Symposium on Security and Privacy (S&P'06)*, pages 350–364. IEEE Computer Society, 2006.

[11] R. Dhamija, J. D. Tygar, and M. Hearst. Why Phishing Works. In *Conference on Human Factors in Computing Systems (CHI'06)*, pages 581–590. ACM, 2006.

[12] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the art of virtualization. In *ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 164–177. ACM, October 2003.

[13] N. Feske and C. Helmuth. A Nitpicker's guide to a minimal-complexity secure GUI. In *21st Annual Computer Security Applications Conference (ACSAC'05)*, pages 85–94. IEEE Computer Society, 2005.

[14] S. Gajek, A.-R. Sadeghi, C. Stüble, and M. Winandy. Compartmented security for browsers – or how to thwart a phisher with trusted computing. In *2nd International Conference on Availability, Reliability and Security (ARES'07)*, pages 120–127. IEEE Computer Society, 2007.

[15] S. Gajek, J. Schwenk, and X. Chen. On the insecurity of microsoft's identity metasystem cardspace. Technical Report HGI TR-2008-004, Horst Görtz Institute for IT-Security, Ruhr-University Bochum, 2008.

[16] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 193–206. ACM, 2003.

[17] R. P. Goldberg. *Architectural Principles for Virtual Computer Systems*. PhD thesis, Harvard University, 1972.

[18] K. Goldman, R. Perez, and R. Sailer. Linking remote attestation to secure tunnel endpoints. In *First ACM Workshop on Scalable Trusted Computing (STC'06)*, pages 21–24. ACM, November 2006.

[19] C. Grier, S. Tang, and S. T. King. Secure web browsing with the OP web browser. In *IEEE Symposium on Security and Privacy (S&P'08)*, pages 402–416, Washington, DC, USA, 2008. IEEE Computer Society.

[20] T. Groß and B. Pfitzmann. SAML artifact information flow revisited. In *IEEE Web Services Security Symposium (WSSS'06)*, pages 84–100. CERIAS, 2006.

[21] Intel Corporation. Intel trusted execution technology software development guide. Technical Report Document Number: 315168-005, Intel Corporation, June 2008.

[22] Internet Crime Complaint Center. 2008 Internet Crime Report. `http://www.ic3.gov/media/annualreport/2008_IC3Report.pdf`, 2008.

[23] S. Ioannidis and S. M. Bellovin. Building a secure web browser. In *FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 127–134, Berkeley, CA, USA, 2001. USENIX Association.

[24] D. P. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review*, 26(5):5–26, 1996.

[25] C. Jackson, D. Boneh, and J. Mitchell. Spyware resistant web authentication using virtual machines. `http://crypto.stanford.edu/spyblock/`, 2006.

[26] C. Jackson, D. Boneh, and J. Mitchell. Transaction generators: Root kits for web. In *2nd USENIX Workshop on Hot Topics in Security (HotSec'07)*, pages 1–4. USENIX Association, 2007.

[27] R. C. Jammalamadaka, T. W. van der Horst, S. Mehrotra, K. E. Seamons, and N. Venkasubramanian. Delegate: A proxy based architecture for secure website access from an untrusted machine. In *22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 57–66. IEEE Computer Society, 2006.

[28] S. Jiang, S. Smith, and K. Minami. Securing web servers against insider attack. In *17th Annual Computer Security Applications Conference (ACSAC'01)*, pages 265–276. IEEE Computer Society, 2001.

[29] N. Jovanovic, E. Kirda, and C. Kruegel. Preventing cross site request forgery attacks. In *IEEE International Conference on Security and Privacy in Communication Networks (SecureComm'06)*. IEEE, 2006.

[30] E. Kirda, C. Krügel, G. Vigna, and N. Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *ACM Symposium on Applied Computing (SAC)*, pages 330–337. ACM, 2006.

[31] D. Kormann and A. Rubin. Risks of the passport single signon protocol. *Computer Networks*, 33(1–6):51–58, 2000.

[32] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala. On-board credentials with open provisioning. In *4th International Symposium on Information, Computer, and Communications Security (ASIACCS'09)*, pages 104–115. ACM, 2009.

[33] B. Krebs. The new face of phishing. *Washington Post*, 2006. `http://blog.washingtonpost.com/securityfix/2006/02/the_new_face_of_phishing_1.html`.

[34] P. C. S. Kwan and G. Durfee. Practical uses of virtual machines for protection of sensitive user data. In *Information Security Practice and Experience Conference (ISPEC'07)*, pages 145–161. Springer, 2007.

[35] J. Liedtke. On micro-kernel construction. In *15th ACM Symposium on Operating System Principles (SOSP'95)*, pages 237–250. ACM, 1995.

[36] R. Macdonald, S. Smith, J. Marchesini, and O. Wild. Bear: An open-source virtual secure coprocessor based on TCPA. Technical Report TR2003-471, Department of Computer Science, Dartmouth College, 2003.

[37] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri. Minimal TCB code execution. In *2007 IEEE Symposium on Security and Privacy (S&P'07)*, pages 267–272. IEEE Computer Society, 2007.

[38] R. Oppliger, R. Hauser, and D. A. Basin. SSL/TLS session-aware user authentication revisited. *Computers & Security*, 27(3-4):64–70, 2008.

[39] B. Pfitzmann and M. Waidner. Analysis of liberty single-sign-on with enabled clients. *IEEE Internet Computing*, 7(6):38–44, 2003.

[40] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The ghost in the browser analysis of web-based malware. In *First Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[41] C. Reis, S. D. Gribble, and H. M. Levy. Architectural principles for safe web programs. In *6th Workshop on Hot Topics in Networks (HotNets'07)*. ACM, November 2007.

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

[43] R. Sailer, T. Jaeger, E. Valdez, R. Perez, S. Berger, J. L. Griffin, and L. van Doorn. Building a MAC-based security architecture for the Xen open-source hypervisor. In *21st Annual Computer Security Applications Conference (ACSAC'05)*, pages 276–285. IEEE Computer Society, 2005.

[44] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *13th USENIX Security Symposium*, pages 223–238, August 2004.

[45] SANS Institute. SANS Top-20 2007 Security Risks. `http://www.sans.org/top20/2007/top20.pdf`, November 2007.

[46] S. W. Smith and S. Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31(8):831–860, April 1999.

[47] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. MD5 considered harmful today—creating a rogue CA certificate. In *25th Chaos Communication Congress (25C3)*, December 2008. `http://www.win.tue.nl/hashclash/rogue-ca`.

[48] F. Stumpf, O. Tafreschi, P. Rder, and C. Eckert. A robust integrity reporting protocol for remote attestation. In *Second Workshop on Advances in Trusted Computing (WATC'06 Fall)*, Tokyo, December 2006.

[49] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. RFC5054: Using the secure remote password (SRP) protocol for TLS authentication, 2007. `http://www.ietf.org/rfc/rfc5054`.

[50] Trusted Computing Group. TPM main specification, version 1.2 rev. 103, July 2007. `https://www.trustedcomputinggroup.org`.

[51] M. Wu, R. C. Miller, and G. Little. Web Wallet: Preventing Phishing Attacks by Revealing User Intentions. In *2nd Symposium on Usable Privacy and Security (SOUPS'06)*, pages 102–113. ACM, 2006.

[52] T. Wu. The secure remote password protocol. In *Network and Distributed System Security Symposium (NDSS'98)*, pages 97–111. The Internet Society, 1998.