Using SafeKeeper to Protect Web Passwords

Arseny Kurnikov
Aalto University, Finland
arseny.kurnikov@aalto.fi

Klaudia Krawiecka
Aalto University, Finland
kkrawiecka@acm.org

Andrew Paverd
Aalto University, Finland
andrew.paverd@ieee.org

Mohammad Mannan
Concordia University, Canada
m.mannan@concordia.ca

N. Asokan
Aalto University, Finland
asokan@acm.org

ABSTRACT

Although passwords are by far the most widely-used user authentication mechanism on the web, their security is threatened by password phishing and password database breaches. SafeKeeper is a system for protecting web passwords against very strong adversaries, including sophisticated phishers and compromised servers. Compared to other approaches, one of the key differentiating aspects of SafeKeeper is that it provides web users with verifiable assurance that their passwords are being protected.

In this paper, we demonstrate precisely how SafeKeeper can be used to protect web passwords in real-world systems. We first explain two important deployability aspects: i) how SafeKeeper can be integrated into the popular WordPress platform, and ii) how ordinary web users can use Intel SGX remote attestation to verify that SafeKeeper is running on a particular server. We then describe three demonstrations to illustrate the use of SafeKeeper: i) showing the user experience when visiting a legitimate website; ii) showing the encryption of the password in transit via live packet-capture; and iii) showing how SafeKeeper performs in the presence of phishing.

ACM Reference Format:
Arseny Kurnikov, Klaudia Krawiecka, Andrew Paverd, Mohammad Mannan, and N. Asokan. 2018. Using SafeKeeper to Protect Web Passwords. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3184558.3186968

1 INTRODUCTION

Passwords are the most widely-used mechanism for user authentication on the web. Password-based authentication outperforms competing solutions in terms of usability and deployability: it is easy to learn, easy to use, simple and cheap to deploy, and compatible with virtually all servers and browsers.

However, phishing and password database breaches are two major security concerns [12, 14, 15]. Although password databases usually store only password hashes, theft of a password database often obliges users to change their passwords, since offline brute force attacks can be efficient [4]. Since users often re-use the same password across multiple services, a password leaked from one site could likely be used to compromise other accounts [7].

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3186968

We previously described SafeKeeper [9], a system that protects web passwords against phishing and password database breaches. It consists of two components: a server-side password protection service and a client-side web browser addon. The password protection service replaces the server's existing password hash function with a *keyed* Cipher-based Message Authentication Code (CMAC) performed inside a Trusted Execution Environment (TEE), implemented as an Intel SGX *enclave* [8]. Thus even if the password database is breached, the adversary cannot perform an offline guessing attack without knowing the CMAC key. The web browser addon uses *remote attestation* to verify that the server is running the SafeKeeper password protection service in an SGX enclave. Through this protocol, the browser addon also establishes a shared key with the enclave and uses this to encrypt the password (even within a TLS connection). This ensures that the password can only be read by the SafeKeeper enclave, *even if the web server is compromised*. To defend against spoofing by malicious websites, the browser addon shows the user which input fields on the web page will be encrypted.

In this demonstration, we show how SafeKeeper can be used to protect web password in real-world systems. In addition to describing the demonstrations, this paper also explains two important deployability considerations that arise from integrating SafeKeeper into real-world systems.

Firstly, integrating SafeKeeper into existing server-side systems, like WordPress, requires interaction between a low-level hardware-based security technology (Intel SGX) and a high-level web runtime (e.g. PHP). This raises subtleties that may be overlooked in a naive integration. For example, although it is possible to implement SafeKeeper's server-side password protection service as a simple PHP extension, this will not work in all deployment models because in some cases, the PHP process can be forked to achieve load balancing, but an SGX enclave cannot be forked. We explain how our implementation overcomes this challenge in a deployment-agnostic manner (Section 3.1).

The second deployability consideration arises from SafeKeeper's use of Intel SGX remote attestation. To verify an SGX remote attestation message (i.e. a *quote*), the verifier must send the quote to the Intel Attestation Service (IAS). Intel requires all verifiers to register a certificate in order to establish a mutually-authenticated TLS connection with the IAS. In SafeKeeper, every ordinary web user must be able to verify the password protection service running on the server, but it is unrealistic to require every user generate and register a certificate for this purpose. To overcome this challenge, we implemented an SGX attestation proxy that allows ordinary web users to use remote attestation securely (Section 3.2).

2 BACKGROUND

The key enabling technology for SafeKeeper's server-side password protection service is Intel Software Guard Extensions (SGX) [8]. SGX is an implementation of a TEE that allows applications to isolate security-critical regions, called *enclaves*, such that they are not accessible by any other software running on the machine. The enclave memory is encrypted and access control is enforced by the hardware. When an enclave is created, SGX allocates protected memory pages and loads the enclave into the allocated region. After the enclave is initialized, it is not possible for any code outside enclave to access the protected memory pages. To perform a call to the enclave, a special `ECALL` CPU instruction is used. An enclave defines a set of ecalls that it exposes.

An important feature of TEEs is remote attestation – a process through which a remote party can verify the integrity and authenticity of the code running in the TEE. In SGX, remote attestation is performed by retrieving a so-called *quote* from a special enclave on the machine that holds CPU-specific signing keys. The quote contains a *measurement* of the enclave that uniquely identifies the code inside the enclave. To verify the quote, a remote party sends it to the Intel Attestation Service (IAS). The IAS responds with the status of the quote, and signs the response message with the IAS signing key. The response signature can then be verified by the requesting party and if the response states that the quote is valid, the remote attestation is successful, meaning that the enclave is running on a genuine Intel CPU and the enclave is running the code it claims.

Additionally, during the attestation process, a Diffie-Hellman (DH) key exchange protocol is executed. When generating a quote, the enclave includes its public DH key in the quote. Successful quote verification assures that by following the DH protocol (i.e. generating a key pair and obtaining the shared secret key), data encrypted with the agreed key will only be accessible by the enclave. Remote attestation and secure channel establishment are two main features that allow SafeKeeper to securely transfer users passwords directly from the browser to the SafeKeeper password protection service running on the server.

3 SYSTEM OVERVIEW

In this section, we provide a brief overview of SafeKeeper from the perspective of a web user, and we explain two important deployability aspects: i) how SafeKeeper can be integrated into the popular WordPress platform, and ii) how ordinary web users can use Intel SGX remote attestation to verify that SafeKeeper is running on a particular server.

SafeKeeper consists of two parts: the browser addon and the server-side password protection service. The addon serves the role of verifying that the server actually supports SafeKeeper, and notifying the user that it is safe to enter the password in a particular input field. The user downloads the addon from a trusted browser addon repository such as Chrome Web Store or Firefox Add-ons. It is open-source and it can be inspected by the users to make sure that its functionality is as expected.

When a browser loads a web page, the addon checks for the header field that contains a quote from the server SGX enclave. If this field is present, the quote is sent to the IAS for verification. If the quote verification passes, the addon changes its icon to show that the web page supports SafeKeeper (see Figure 1). The addon checks that the enclave's public key matches the value included in the quote, and if so, proceeds to agree on a Diffie-Hellman (DH) session key.

```
$hash = md5($salt . $password, TRUE);
do {
  $hash = md5($hash . $password, TRUE);
} while (--$count);
```

Listing 1: Original WordPress hashing

Before entering the password, the user should click on the SafeKeeper icon to turn the page into "highlighting" mode. In this mode, the browser addon decreases the opacity of the page elements that are not protected, and highlights the protected element so that it stands out on the page. Additionally, a tooltip points the user to the element that is protected by the service. A popup window (shown in Figure 2) tells the user that SafeKeeper is now highlighting the protected input fields. Clicking on the icon again turns off highlighting mode and restores the page to its original appearance. Turning highlighting mode on and off can prevent certain types of delayed spoofing attacks, as described in Section 5.3. Figure 3 shows the same page with highlighting mode on and off. The user can be sure that the input entered in the highlighted field will be encrypted before sending to the server.

Figure 1: SafeKeeper browser addon icons: the first one indicates that SafeKeeper is unavailable on the website, or that the attestation protocol has failed; the second icon shows that SafeKeeper is supported and that a secure channel has been established; and the third icon is used when SafeKeeper is highlighting the protected input fields (see Section 3).

3.1 Integrating SafeKeeper with WordPress

We integrated SafeKeeper into WordPress by replacing its original password hashing algorithm with a call to the SafeKeeper enclave. We implemented a PHP extension using the PHP-CPP library [10]. WordPress comes with a PHP class `PasswordHash` [11] that provides the functionality for generating salts, hashing passwords and checking that the provided password matches the hash value. Listing 1 shows the MD5-based hash algorithm in `PasswordHash`.

In the SafeKeeper version of WordPress, we substituted the MD5 hash with a call to `sgx_cmac` function. The function parameter is the same as the input to the MD5 hash function: the concatenation of the password and the salt. The function performs a call to the SafeKeeper service, and obtains the password CMAC. The service

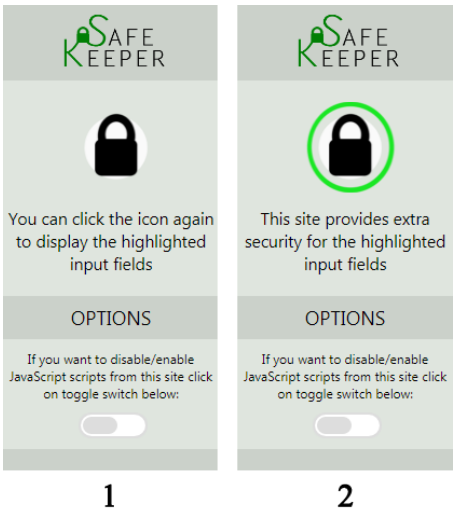

Figure 2: SafeKeeper popup windows: (1) the page supports SafeKeeper, but the protected input fields are not currently highlighted; (2) SafeKeeper is highlighting the protected fields in the page.

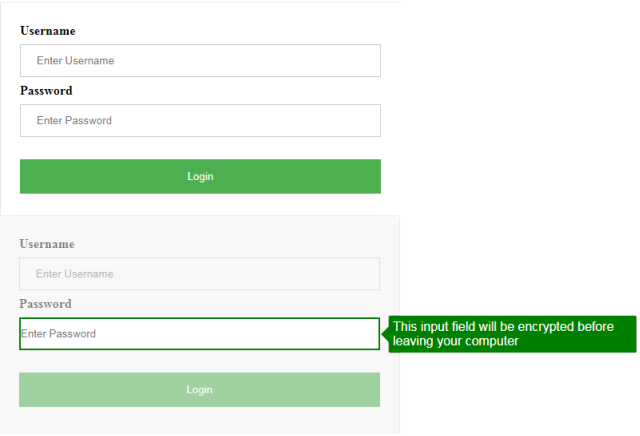

Figure 3: *Top:* The original webpage; *Bottom:* SafeKeeper highlighting the password input field.

only outputs a CMAC of the password, which is stored in the password database. Thus even in case of a compromised or a malicious web server the user password is protected by the key that is not available to an adversary. Additionally, the service exposes an API call to get the quote from the enclave.

When an HTTP server gets a request that needs to be processed by PHP interpreter, typically the communication between the server and the PHP process occurs over Common Gateway Interface (CGI). In some configurations there is one PHP process running constantly and serving all the requests. For scalability reasons though often there are multiple children that are pre-forked and the requests are scheduled (distributed) to them by the parent process. In SGX, if a process is forked after starting an enclave, the child process cannot invoke calls to the enclave. To overcome this, our implementation of SafeKeeper on the server side consists of two parts: a PHP extension exposing the necessary API to PHP, and a standalone service containing the enclave to serve the requests from the PHP extension. The communication between the PHP extension and SafeKeeper service is via TCP/IP sockets. This allows for better scalability where multiple HTTP frontend servers communicate with a single Trusted Application instance. These frontend servers do not need to be SGX-enabled.

3.2 Intel SGX Attestation

Verification of the remote server is achieved using remote attestation, as explained in Section 2. The Intel SGX remote attestation process involves contacting the Intel Attestation Service (IAS) to verify the quote obtained from the enclave. Before this communication is possible, the party that wishes to verify quotes must enroll with Intel IAS and obtain a certificate from Intel to be used in the TLS session with the IAS. In a user-centric setting like SafeKeeper, where the verifier is the browser addon running on end user devices, the logistics of arranging every end user device to enroll with the IAS is simply unrealistic. Presumably, Intel's IAS design is not intended to be used in such user-centric settings. To solve this problem we have developed a proxy to communicate with the IAS. This proxy receives verification requests from SafeKeeper addon to forward them to the IAS, performing the necessary encryption for the IAS using our registered TLS certificate.

The proxy does not need to be trusted because its sole purpose is to send the quote to the IAS, retrieve the response, and send it back to the verifier. Since the response is signed by the IAS, the verifier can check that the response is valid and that it originates from the IAS by verifying the signature. We implemented the proxy using an Nginx server, installing our TLS certificate, and configuring the IAS to be an upstream for the HTTP proxy. SafeKeeper enclave uses the same public DH ephemeral key for every communication with SafeKeeper addons, hence the quote is also the same. Utilizing this, the proxy can cache the IAS response and serve following requests from the cache, reducing the load on the IAS and decreasing the quote verification latency.

An alternative solution would be for the SafeKeeper server itself to act as a "proxy", contact the IAS and obtain the attestation response from the IAS. The SafeKeeper enclave can be provisioned with the certificate to contact the IAS to perform the attestation.

4 RELATED WORK

Security threats against passwords have led to much research on server-side password protection mechanisms. For example, purpose-built hash algorithms have been designed to slow down brute-force attacks [1, 2]. In contrast, since SafeKeeper protects the password using a CMAC, there is no need to slow down the hash operation.

On the client side the research focuses on storing user passwords so that only one master password needs to be remembered. Preventing adversaries from mounting offline attacks on password vaults is an important research area [3, 5]. However phishing protection and notifying users about potentially malicious websites is usually performed by checking the TLS certificate. PwdHash [13] proposes to hash every password transparently on the client side along with

the URL. This results in passwords that are different for each website even if the original password is the same and prevents phishing. SafeKeeper visually shows the user if the website is trustworthy and what fields are protected.

5 DEMONSTRATIONS

We will demonstrate how SafeKeeper protects web passwords by showing three aspects of the system. Firstly, we will show the normal user experience when visiting a WordPress website that uses SafeKeeper. Secondly, we will show via live packet-capture that the user's password is encrypted before it leaves the browser. Thirdly, we will show a selection of websites that attempt to spoof the SafeKeeper UI, and the audience will be able to see first-hand if they can detect this forgery using SafeKeeper. There are several spoofing possibilities that were used in the SafeKeeper user study [9] to empirically evaluate whether users understand the signaling employed by SafeKeeper and use it to correctly identify spoofed websites. We describe each of these aspects in detail below.

5.1 WordPress using SafeKeeper

We will demonstrate two WordPress websites, one on which uses SafeKeeper. We will show the processes of creating an account and logging into the website in both cases. When using the SafeKeeper-enabled website, the browser addon verifies the SGX quote and displays a lock icon if the verification succeeds. The user can then click on the icon to highlight the protected input fields, as explained in Section 3. This demonstration will also show that there is no noticeable performance overhead when using SafeKeeper.

5.2 Password Encryption

While the audience interacts with the websites, we will show a live packet capture between the browser and the web server. This will show the initial quote being sent from the server to the browser, and the SafeKeeper browser addon verifying this quote using our IAS proxy as explained in Section 3.2. After the user enters password and submits the form, the POST request will have the password value encrypted with the generated DH key. We will perform this packet capture using Wireshark and the connection between the browser and server will not be by TLS. This will illustrate how SafeKeeper protects passwords even over insecure HTTP connections.

5.3 Attempted Spoofing

There are several ways an adversary may attempt to spoof the SafeKeeper UI to trick the user into entering the password into an unprotected input field. First, the adversary may serve a page where the password field is not protected, even though the quote is valid. In this case, the SafeKeeper addon will change its icon to the lock but if the user does not click on the icon and simply enters the password into the form field, it will be sent to the server un-encrypted and can be obtained by an adversary who has compromised the sever. To avoid this spoofing, users should always click on the addon icon and only enter their passwords into protected fields.

Secondly, the adversary may attempt to spoof the SafeKeeper highlighting for a field that is not protected. The quote verification may succeed or fail, depending on whether the adversary runs the SafeKeeper enclave. Again, before entering the password into a "highlighted" field, the icon should be checked and clicked. The adversary's spoofed highlighting will fade when the real SafeKeeper addon performs its own highlighting.

Additionally, the adversary can highlight the field after a delay, trying to trick the user into believing that the highlighting is a result of the icon click. Since all highlighting is performed by JavaScript, this could in theory allow the adversary to change the highlighting even when the SafeKeeper addon is in highlighting mode. However, since the adversary does not have the information about when the user clicks the icon, it is possible to detect this delayed highlighting spoofing by clicking the icon several times, or in the worst case, using the SafeKeeper browser addon to disable all other JavaScript.

To demonstrate how SafeKeeper would look on real websites, and to demonstrate possible spoofing attempts, we used the HTTrack tool to clone the login pages of 25 popular websites [6]. We then modified some of these pages to use SafeKeeper and others to attempt to spoof the SafeKeeper UI. Our interactive demonstration will give the audience the chance to use SafeKeeper and attempt to detect the different types of spoofing described above.

ACKNOWLEDGMENTS

This work was supported in part by the Intel Collaborative Research Institute for Secure Computing at Aalto University, and the Cloud Security Serivces (CloSer) project (3881/31/2016), funded by Tekes. M. Mannan is supported in part by an NSERC Discovery Grant and a NordSecMob Scholarship.

REFERENCES

[1] A. Biryukov, D. Dinu, and D. Khovratovich. 2016. Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications. In *IEEE European Symposium on Security and Privacy*. https://doi.org/10.1109/EuroSP.2016.31
[2] J. Blocki and A. Datta. 2016. CASH: A Cost Asymmetric Secure Hash Algorithm for Optimal Password Protection. In *IEEE Computer Security Foundations Symposium*. https://doi.org/10.1109/CSF.2016.33
[3] Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. 2010. Kamouflage: Loss-Resistant Password Management. In *European Symposium on Research in Computer Security*. https://doi.org/10.1007/978-3-642-15497-3_18
[4] Joseph Bonneau. 2012. The Science of Guessing: Analyzing an Anonymized Corpus of 70 million Passwords. In *IEEE Symposium on Security and Privacy*. https://doi.org/10.1109/SP.2012.49
[5] Rahul Chatterjee, Joseph Bonneau, Ari Juels, and Thomas Ristenpart. 2015. Cracking-Resistant Password Vaults using Natural Language Encoders. In *IEEE Symposium on Security and Privacy*. https://doi.org/10.1109/SP.2015.36
[6] HTTrack Website Copier. 2017. (2017). https://www.httrack.com/.
[7] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. 2014. The Tangled Web of Password Reuse. In *Network and Distributed Systems Symposium*. https://doi.org/10.14722/ndss.2014.23357
[8] Intel Corporation. 2017. Software Guard Extensions (Intel SGX). (2017). https://software.intel.com/en-us/sgx.
[9] Klaudia Krawiecka, Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N. Asokan. 2018. SafeKeeper: Protecting Web Passwords using Trusted Execution Environments. In *The Web Conference (WWW)*. https://doi.org/10.1145/3178876.3186101
[10] PHP-CPP: A C++ library for developing PHP extensions. 2017. (2017). http://www.php-cpp.com/.
[11] PHPass: Portable PHP password hashing framework. 2017. (2017). http://www.openwall.com/phpass/.
[12] PhishTank.com. 2017. Statistics about phishing activity and PhishTank usage. (2017). https://www.phishtank.com/stats.php.
[13] Stanford PwdHash. 2017. (2017). https://pwdhash.github.io/website.
[14] Have I Been Pwned. 2017. (2017). https://haveibeenpwned.com/pwnedwebsites.
[15] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, D. Margolis, V. Paxson, and E. Bursztein. 2017. Data Breaches, Phishing, or Malware?: Understanding the Risks of Stolen Credentials. In *ACM SIGSAC Conference on Computer and Communications Security*. https://doi.org/10.1145/3133956.3134067