TrustUI: Enabling Trusted User Input and Output Channels to Web-Applications in Untrusted Client Platforms

Ijlal Loutfi[1] and Audun Jøsang[2]

University of Oslo, Gaustadalléen 23B, 0373 Oslo,Norway
loutfi.ijlal@gmail.com
audun.josang@mn.uio.no

Abstract. As the proliferation of web services continues to increase, so does our reliance on web browsers. On the one hand, users routinely input sensitive data, such as passwords, into its web-pages. Yet, nothing stops a compromised browser from leaking it to malicious third-parties. On the other hand, users also trust the displayed browser web-pages to be non-tampered with. Yet, a Man-In-The-Browser malware (MITB) can stealthily modify the HTML rendering of these sensitive web-pages. MITB can inject authentic looking input fields into legitimate web-pages, prompting unsuspecting users to enter their sensitive data into them. It can also change the displayed value of a payment amount transaction to get the user to approve it, while rerouting the funds to a different account.

Indeed, the lack of trusted input and output channels compromises the trust between web users and their respective web service providers. Therefore, the aim of this paper is to enable trusted input and output channels from end-users to web servers, despite the presence of a malicious browser, and even a compromised system software.

TrustUI is the name of the paper's proposed solution. On the one hand, TrustUI mitigates the lack of trusted output channels by first programmatically capturing the browser's screen directly from the GPU's frame buffer. It then runs image-analysis on the captured screen to compare it to a generic trustworthy reference image. On the other hand, TrustUI solves the lack of trusted input channels by establishing an end-to-end encrypted channel between web users and servers. It achieves this with a combination of two trusted execution environments: An Intel SGX web enclave instantiated within the browser address space, and a personal security device showcasing a secure element and a separate keypad input at the side of end-users.

This paper also experimentally evaluates the core functionalities of TrustUI and reports its results.

Keywords: Browser, Man-In-The-Browser Malware, Input Channel, Output Channel, Trusted Execution Environment, Intel SGX

1 Introduction

The web has become a vital part of today's modern society. Millions of users rely on it daily to retrieve information, transfer money, access e-government services, and conduct business [22]. They also rely on it to configure various safety-critical devices, such as industrial control systems and medical devices, which offer web interfaces for remote configuration [10]. This has made the web an attractive target for cybercriminals, who look for ways to compromise all its three core components: web servers, web clients and the network communication layer [20]. As a defence mechanism, the communication layer between web clients and servers is protected by end-to-end encryption technologies such as the Transport Layer Security, while web servers run on tightly managed commercial server platforms. They are frequently updated and often hardened with enterprise grade security mechanisms, such as Hardware Security Modules [19]. However, the web client remains the weakest attack vector, as it runs on the untrusted platforms of end-users.

Problem Statement. One of the most critical components of the client platform is the browser software. For many web users, it is their primary gateway for reaching and consuming web services. Consequently, browsers process and store various security-sensitive data, such as authentication credentials and credit card details. In this paper, we focus on the two following attack classes:

1. **Trusted Input Channel Attack**: when users need to communicate with their remote web service providers, they enter input data into their platforms through standard input interfaces, such as the keyboard. This input is then processed by the system software and the browser before being sent over the Internet to the web server. The later relies on this input to make various security decisions, such as granting access to the end-user and approving their transactions. However, a malicious browser or a keylogger which has hooked into the kernel or the browser' API is able to leak it to third-parties [17].
2. **Trusted Output Attack**: Web users routinely rely on what the browser displays to them in order to make security decisions. For instance, they check the banking transaction amounts displayed in their screen before approving them for payment. However, nothing stops a compromised malicious browser from redirecting the payment transaction to the attacker's account, then modifying the HTML rendering of the banking web page to still display the legitimate transaction. The end-user would then unsuspectingly confirm the transaction, and the bank would process it as if it originated from a legitimate user. It can also inject new authentic-looking fields prompting the user to enter sensitive information such as their credit card details. As the rest of the web-page, such as the URL and the 'https' lock, remain unchanged, it is very difficult for users to detect that they are interacting with a maliciously tampered web page [28] [6]. This type of malware is referred to as Malware-in-The-Browser (MITB).

Indeed, the lack of trusted input and output channels compromise the trust between end-users and their service providers. Therefore, the main research ques-

tion of how to enable web users to securely communicate with web application servers. In other words, we need to enable trusted input and output channels from the end-user to the web application, despite the presence of a malicious browser and even system software.
Our Solution. In this paper, a two-part solution architecture called TrustUI is proposed:

- **TrustUI-Trusted Output Solution**: The intuition behind TrustUI is the observation that it is ineffective to leave it to end users to detect when a web page's rendering has been modified as the new modified, since the rogue web-page can look completely identical to the original one. Furthermore, solutions proposing out-of-band verification, such as SMS's sent by banks to confirm payment details, can still be defeated. This is due to two fundamental weaknesses: on the one hand, out-of-band modalities are subject to habituation. Users simply start confirming the transactions out of habit without double-checking their details, or become inattentive to small transaction modifications such as similar bank accounts, or slightly rounded transaction amounts [5]. On the other hand, even if out-of-band verification is successfully carried out by the user, its effectiveness is only limited to attack scenarios where only dynamic page values, such as transaction amount, are modified. Multi-Factor out-of-band solutions are ineffective against attack scenarios where rogue elements are added into the web page, such as input fields prompting users to enter the passwords into them. However, TrustUI intuition is that a web server, such as a bank, can increase trust in the transaction they are processing if they have a guarantee that it matches the screen displayed to the end-user. In a parallel world, web servers would have a way to access the users' screen before every transaction. In the absence of that, TrustUI proposes a two-step solution which can achieve the same guarantees, albeit in a different manner. The first component is a screen capture engine: it programmatically captures a bitmap representation of the rendered display directly from the GPU frame buffer, which is out of reach from malware-in-the-Browser. The second component is the image analysis engine, which cross-compares the captured image to a reference one. This allows it to detect any rogue injected elements. Furthermore, if the goal is to detect tampering with dynamic values such as the transaction balance, the optical character recognition can be performed.
- **TrustUI-Trusted Input Solution**: to protect the confidentiality of user input from a compromised browser and operating system, TrustUI aims to build an end-2-end encrypted input channel between the end-users' input interface and their web server. A straightforward solution to this problem would be to encrypt all sensitive input at the level of the keyboard's subsystem, before it becomes accessible to the rest of the platform's system software and browser. However, this solution is impractical with out-of-the box commodity platforms, since the standard keyboards are not able to perform secure cryptographic operations and negotiate keys. Instead, TrustUI proposes to equip users with personal security devices with dedicated input

keypads. These PSDs can establish an end-2-end encrypted channel to a web enclave which TrustUI establishes within the browser's address space. Intel SGX can protect user's input secure against all platform malware, including a malicious operating system. Furthermore, the security of PSD's is based on an embedded secure element, and a separate keypad and display [15]. These two Trusted execution environments negotiate and establish an end-2-end encrypted channel. The user then uses the keypad of the trusted device to enter their security sensitive input, which is sent encrypted to the web enclave and finally to the web application.

Outline. The rest of the paper is organized as follows: section 2 presents the needed background. Section 3 introduces the proposed solution. Subsequently, section 4 presents the experimental results. The paper then follows with an overview of the related work, and concludes with a discussion of the results, open research questions and suggested future work.

2 BACKGROUND

2.1 Man-in-the-Browser-Attacks

Man-In-The-Browser malware (MITB) is a 'Trojan that infects endpoints through malicious email attachments, links, or even when a user visits an infected website [21]. Once a platform is infected, MITB malware hooks itself into browser elements such as Browser Helper Objects (BHO), plugins, JavaScript, Add-on features, Ajax calls and DOM Object models [6].An MITB can then see and do everything a web user can see and do with a browser. The attacks it launches usually fall into one of two common patterns. First, MITB can silently wait for the user to visit a target web-page, such as the authentication page of a bank. It then intercepts the authentication credentials and leaks them to other malicious third-parties. In the second attack scenario, the MITB takes a more active role. It modifies the rendering of the webpage elements, effectively changing what is being displayed for the user. For instance, it can modify the value of a payment transaction, or even inject new authentic-looking fields prompting the user to enter sensitive information such as their credit card details. As the rest of the web-page such as the URL and the 'https' lock remain unchanged, it is very difficult for users to detect that they are interacting with a maliciously tampered web page.

In this way, MITB malware is different and more dangerous than traditional phishing attacks. while the later uses links or email attachments to get users to a fake website where they input their secure data, MITB still lets the victim interact with the legitimate website, while passively stealing their data or actively modifying their web-page display [21].

2.2 Frame Buffer

The frame buffer is part of the graphics subsystem. It is a large, contiguous piece of computer memory that sits between the processor and the display. Whenever

the CPU sends a frame update to the graphics subsystem, the graphics processor forms a picture of the screen image and stores it in the frame buffer. 'It typically stores a bitmap representation of the screen, which consists of colour values for every pixel to be shown on the display. Colour values are commonly stored in 1-bit binary (monochrome), 4-bit palettized, 8-bit palettized, 16-bit high colour and 24-bit true colour formats. The data is sent as a digital signal to the display [16] [13].

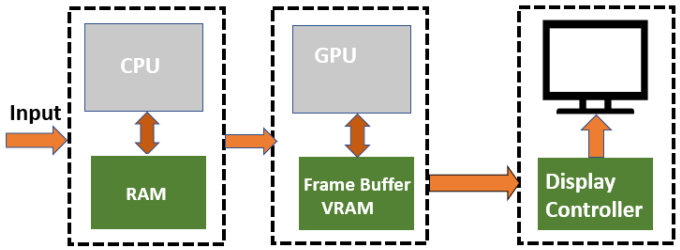

Fig. 1. Buffer Frame: A System Overview

2.3 Intel Software Guard Extensions

A Trusted Execution Environment (TEE) is a system security primitive that provides enforced isolation for executing security sensitive application logic. The TEE isolates its hardware and software resources from a general processing environment, also referred to as the rich execution environment (REE), where most of the platform software, including the primary operating system, runs.

Intel Software Guard Extension (SGX) are a recent security extension for the X86 Intel processor family. Its main aim is to allow ring 3 applications to control their own security by creating enclaves [7]. Enclaves are thus hardware TEEs that applications can create within their address space. Enclaves can then be used to run security-sensitive code and data, where no other software residing on the platform can access it, including the privileged system level software such as the operating system and the hypervisor. Inside the CPU's perimeter, enclave's data is processed in plaintext. Once outside the CPU, it always remains encrypted by the new encryption engine, using a hardware protected key which is generated upon every platform reboot. However, enclaves lack the ability to execute system calls, as this relies on the operating system which is considered untrusted within the SGX threat model [7].

2.4 Personal Security Devices

Personal Security Devices (PSD) are minimal portable embedded devices, which are used to carry security-sensitive operations outside the perimeter of the po-

tentially compromised general-purpose end-point devices. One way to conceptualize them is as a smart card with a display and a keypad. Indeed, they are often designed around a secure element, which is augmented by secure storage, and IO channels. While PSDs have been around for many decades, they have been popularized in recent years by the rise of cryptocurrencies which has prompted the need for wallets to store the cryptographic keys away from the malicious platforms [3].

OffPAD is a one such PSD which has been developed within the university of Oslo. It uses a secure dual-architecture, where certified secure elements are attached to a general-purpose microcontroller which drives the attached UI devices. These are a fingerprint reader, a keypad and a screen [26]

3 SOLUTION OVERVIEW

3.1 TrustUI-Trusted Output Channel Solution

As discussed in the introduction, there are currently no perfect solutions for users and web servers to detect a browser page that has been tampered with. Even multi-factor authentication solutions which rely on out-of-band modalities such as SMS's are not enough. They are completely useless against MITB attacks which inject rogue new elements into the browser's page. Furthermore, even if they can help guard against modifications to dynamic values, such as transaction amounts, they are vulnerable to habituation. Users simply start confirming the transactions out of habit without double-checking their details, or become inattentive to small transaction modifications such as similar bank accounts, or slightly rounded transaction amounts [5].

Therefore, TrustUI intuition is that a web server, such as a bank server, can increase trust in the transaction they are processing if they have a guarantee that it matches the screen displayed to the end-user. TrustUI proposes the following two-steps solution: first, a screen capture engine which programmatically captures a bitmap representation of the rendered display directly from the GPU frame buffer, which is out of reach from malware-in-the-Browser. Second, an image analysis engine, which cross-compares the captured image to a reference one, or which perform Optical Character Recognition, depending on the type of transaction being processed.

USE CASE While we have expanded on the description of such a malware-in-the browser attack in previous sections, we would like to provide the following use case as concrete motivating example to use for the rest of the paper:

1. A user wants to pay 100 USD to the merchant's account A.
2. A Man-In-The-Browser intercepts the transaction and changes the details to 'pay 200 USD to the intruder's bank account B', then forwards it to the bank web server.

3. the bank's web server sends a confirmation request to the user to approve the payment of 200 USD to the intruder's bank account B, in the form of a web page to be rendered by the browser and displayed on the screen.
4. The Man-In-The-Browser malware intercepts the HTML rendering request, and modifies it to display 'approve 100 USD to the merchant's bank account A'.
5. The user is prompted to approve the payment of 100 USD to the Merchant's bank account A.
6. The user approves the payment transaction, and the browser relays it to the bank's server.
7. The bank processes a payment of 200 USD to the intruder's account.

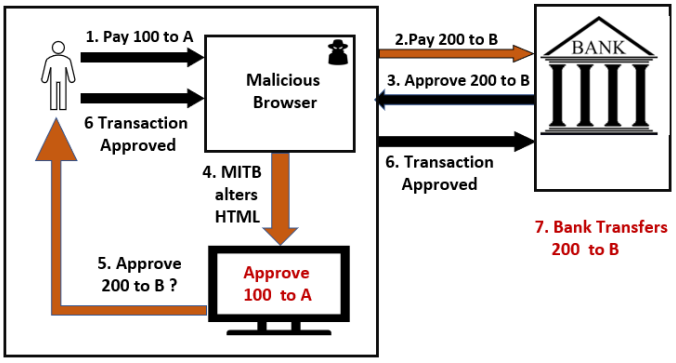

Fig. 2. Malware In the Browser Attack Scenario

SOLUTION ARCHITECTURE TrustUI proposes a two-step solution to allow web servers to gain trust that users are not viewing tampered with web pages.

1. **Screen Capture Engine**: it captures a bitmap representation of the rendered display directly from the GPU frame buffer, which is out of reach for MITB malware.
2. **Image Analysis Engine**: It cross-compares the captured image to a reference one and/or perform OCR to extract numerical and textual values from the capture.

SCREEN CAPTURE ENGINE To get a trustworthy copy of the displayed image, we need to capture it programmatically from a system component that is deemed trustworthy under our threat mode. The frame buffer is one such

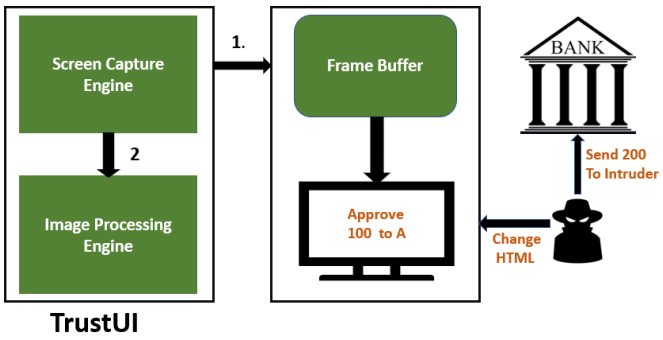

Fig. 3. TrsutUI Solution Overview For the Trusted Output Channel Attack

component. On the one hand, it always has a bitmap representation of the image to be displayed. On the other hand, MITB malware cannot tamper with its content. To securely access the frame buffer, there exists several options which are platform-dependent. In this section, we discuss solutions relevant to Windows platforms, since we used it to run the experiments part of this paper:

- **Kernel Mode Capturing**: it requires writing a kernel video miniport driver, that would have unprivileged access to the frame buffer [18].
- **User Mode Capturing**: it leverages one of the APIs that are offered by the Windows operating system for applications to programmatically access the frame buffer.
  1. **Microsoft DirectX**: This is a collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms [18]. The most interesting API for this paper is the desktop duplication API. It provides controlled access to applications which require remote access to a desktop , and allows them to capture frames as they are updated. 'The interface IDirect3DDevice8 provides GetFrontBuffer() method that takes a IDirect3DSurface8 object pointer and copies the contents of the front buffer onto that surface. Once the screen is captured onto the surface, we can use the function D3DXSaveSurfaceToFile() to save the surface directly to the disk in bitmap format' [18].
  2. **Graphics Device Interface** GDI is also a collection of APIs. It is used for representing graphical objects in the frame buffer, and then transmitting them to output devices such as monitors and printers. GDI is based on the Device Context (DC) object, which is handle to either an entire screen, a window, a subset of a window or a printer. Details about how to use GDI for screen capture are outline in the implementation section.

IMAGE PROCESSING ENGINE Once we have programmatically captured the content of the screen from the frame buffer and reconstructed the

image, we need to process it so as to decide whether it has been tampered with or not. The most straightforward solution here is to compare the captured image to a reference one, which we assume would have been sent by the web server securely. This is sufficient when TrustUI needs to detect whether MITB has injected new elements into the webpage or not, this would be sufficient to conclude whether the web page has been tampered with. However, if the goal is to detect whether a dynamic value such as the bank transaction details has been modified, then Optical Character Recognition is also needed. It can then be compared to the transaction amount being sent back to the server.

While TrustUI can handle both cases, the experiments reported in this paper focus only on comparing a screen capture to a legitimate reference one, without the OCR part. A straightforward solution for TrustUI is to compare the bitmap representation of the two images pixel by pixel. This is sufficient in our case, since TrustUI needs to capture the visual similarity between two images and not its semantic one.

3.2 Trusted Input Channel

When an end-user inputs their sensitive data into the platform's keyboard, its confidentiality can be compromised not only by a malicious browser, but also by all the software stack on that platform. A straightforward solution to this problem would be to encrypt all sensitive input at the level of the keyboard's subsystem, before it becomes accessible to the rest of the platform's system software and browser. However, this solution is impractical for several reasons: first, standard keyboards are not able to perform secure cryptographic operations and negotiate keys. Second, it is not clear where the decryption would take place in such a scenario.

TrustUI still relies on the same idea of having the sensitive input encrypted while it is present within the platform. However, given the above discussed limitations, TrustUI leverages two additional hardware-based trusted execution environments to enable the solution.

- End-user side: instead of relying on the default platform keyboard interface, TrustUI equips the user with trusted embedded device, otherwise referred to as Personal Security Devices, PSD. At the core of PSDs' architecture is a secure element able of securely performing cryptographic operations, and which communicates securely with the PSD's dedicated UI channels, namely its keypad and display.
- Browser side: TrustUI instantiates an Intel SGX enclave within the browser's address space. This web enclave can then securely negotiate a secure channel with the user's PSD, decrypt all incoming sensitive input, and then forward it to the appropriate service provider. The enclave's operations are protected against any malware residing within either the browser or the platform's system software.

In order to enable this use case, TrustUI relies on a two-step protocol:

1. Secure Channel Establishment: this is a one-time interactive protocol which aims to establish a shared secret key between the web enclave and the personal security device.
2. Message Exchange: it takes place every time a user wants to enter security sensitive web input into a browser's page. The PSD uses the pre-shared secret key, negotiated in step 1, to encrypt the input, while the web enclave uses it for decryption.

INTEL REMOTE ATTESTATION A core component of TrustUI secure channel establishment is the Intel remote attestation procedure. This section outlines its details before moving forward with presenting the solution.

Before a remote service provider provisions secrets into the enclave or accepts the output of its computation, it needs to establish two trust guarantees. First, the enclave needs to be running on a genuine SGX enabled Intel processor, and not an emulated environment. Second, the enclave's initial code and data which were loaded by the operating system should not have been tampered with. Remote attestation is the process which ensures these two guarantees. It is an interactive protocol between the attested platform, the attesting remote service provider and the Intel Attestation Service (IAS), online service operated by Intel [7] [9].

Each SGX hardware has a unique attestation key, which is only accessible by special Intel SGX architectural enclaves, such as the quoting enclave. During an attestation protocol, the quoting enclave first forms a structure referred to as a quote, composed of a hash of measurements reflecting the order and the content of code which has been initially loaded into the enclave by the operating system. Subsequently, the quoting enclave signs the quote using the attestation key and forwards it to the remote service provider and the Intel Attestation Service. The IAS verifies that the signature has been generated by a genuine non-revoked Intel SGX hardware, while the service provider compares the measurement value to a reference one, so as to ensure that the enclave code has not been tampered with during the loading process [7] .

SOLUTION COMPONENTS In this section, we present an overview of the components that are part of the TrustUI solution of the trusted input problem, before introducing the secure channel establishment protocol.

Intel Attestation Service: IAS is involved during the remote attestation process of the web enclave. It has an identity key pair and a certificate signed by Intel.

Remote Service Provider: in our use case, RSV is the web server who needs to establish trust that the input it receives is coming from legitimate users and not malware.

Personal Security Device: the PSD has an identity key pair, whose public key is certified by the manufacturing authority. The certificate authority public key is publicly available to third-parties.

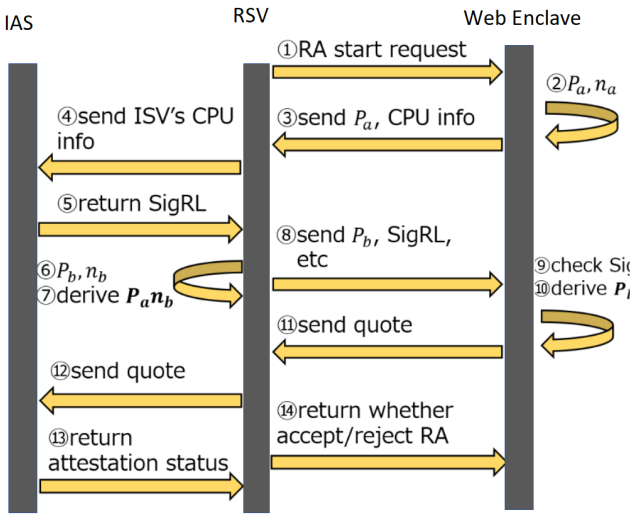

Fig. 4. Malware In the Browser Attack Scenario* [4]

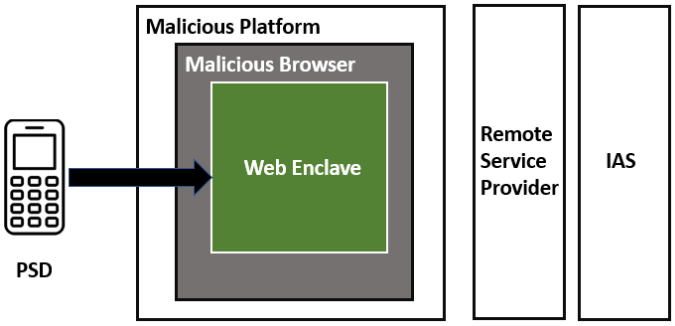

Fig. 5. TrustUI Trusted Input Solution Overview

WebEnclave: it is responsible for negotiating the secure channel with the PSD, and then using the pre-shared key to decrypt the PSD's encrypted input.

SECURE CHANNEL ESTABLISHMENT This a one-time interactive protocol performed at set up time, so as to provide the PSD and the web enclave with a symmetric share key they can use for subsequent communication.

1. PSD generates Pa and constructs Msg1=(nonce,Pa), signs it with its identity private key and sends it to the web enclave.
2. The web Enclave verifies the PSD signature to ensure it has been generated by a genuine hardware non-revoked PSD. We assume that all manufactured PSDs are signed by an authority whose certificate is public ally available to all parties involved in the protocol.

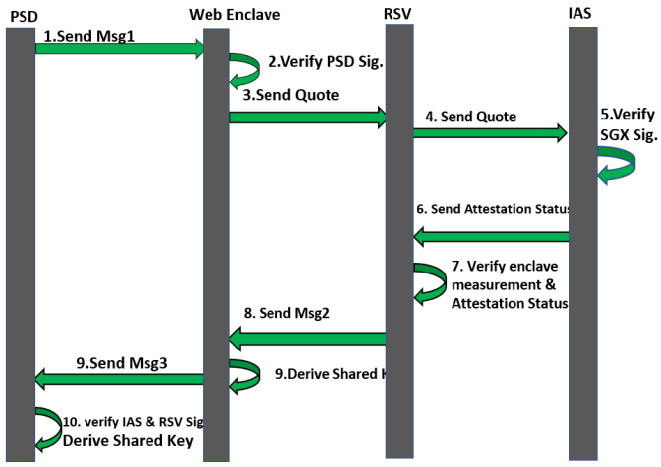

Fig. 6. Secure Channel Establishment Protocol

3. The web enclave invokes the quoting enclave to generate a quote structure and sends it to the RSV.
4. RSV forwards the quote to the IAS.
5. IAS verifies that the quote structure has been generated by a genuine non-revoked Intel SGX hardware.
6. IAS returns the attestation status to the RSV.
7. RSV verifies the attestation status and the enclave measurement, so as to ensure that the enclave has been loaded properly by the untrusted system software.
8. RSV constructs Msg2 = Signed by RSV identity private key (IAS quote response). RSV sends Msg2 to the web enclave.
9. Web enclave generates Pb, and derives Pab and constructs Msg3= (Msg2 + nonce + Pb) encrypted with Pab+ Pb
10. PSD derives Pab, decrypts Msg3, and verifies the IAS and RSV signatures.
11. Web enclave generates gd, and a TPM signature over the quote
12. OffPAD verifies TPM quote with enterprise certificate. Verifies signature of IAS, RS, and device's shared key

At the end of this protocol, both enclave and OffPAD derives a shared secret key K.

4 Experimental Set-Up and Evaluation

4.1 Trusted Output

The experiments were performed on an HP machine with an Intel Core(TM)i7 -75000 CPU, a 32GB RAM, and an Intel(R) HD Graphics 620. The platform

runs a 64-bit Windows 10 Operating system Home edition. The experiment uses Instagram's login page as a use case. In order to mimic the MITB attack which injects rogue input fields into a page, we draw a rogue Instagram mock-up login page. This malicious page hides the legitimate username and password fields, which are centrally indented in the legitimate page, and replaces them with identical looking username and password input fields which are indented to the left. Except from this modification, the webpage including the URL looks completely identical to the legitimate Instagram login page.

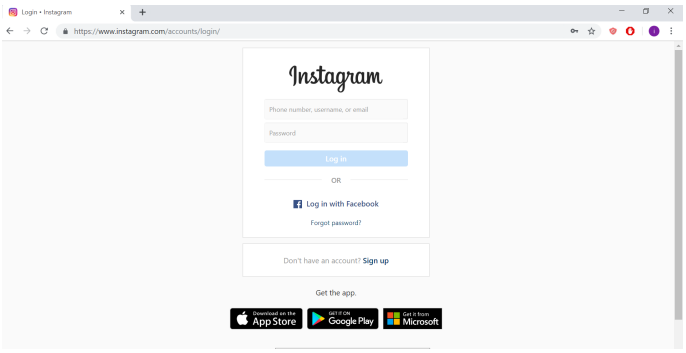

Fig. 7. Reference Legitimate Page- Username/Password Fields are in the middle

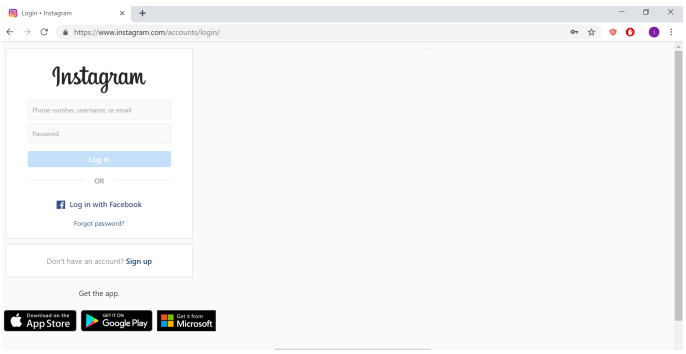

Fig. 8. Malicious Page- Rogue Injected Username/Password field-Left Indented

In order to capture the displayed screen, TrustedUI used an open source implementation of the GDI screen capture solution. [2]. GDI is based on the Device Context (DC) object, which is a handle to either an entire screen, a window, a subset of the window or a printer. Therefore, when TrustUI needs

to programmatically capture the screen displayed by the browser, the following steps are followed:

- Get the device context of the desktop to be captured using the function GetDeksopWindow().
- Get the DC of the desktop window using the function GetDC();
- Create a compatible DC for the Desktop DC along with a compatible bitmap to select into that compatible DC using CreateCompatibleBitmap()

After having captured the image from the frame buffer, we perform a pixel-by-pixel comparison of the bitmap representations of the reference image and the captured one, which is supposedly malicious. As the two images we used are different, we get a trigger that the webpage is malicious.

While the delay introduced by the screen capture image is insignificant, the image analysis takes 4 seconds to complete.

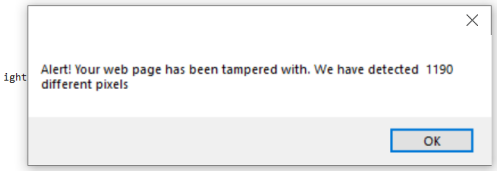

Fig. 9. MITB attack detected

4.2 Trusted Input

The implementation of TrustUI trusted input focuses on its core component, which is the client side. The experiments were run on an HP machine with an Intel Core(TM)i7 -75000 CPU that is SGX enabled, a 32GB RAM. The platform runs a 64-bit Windows 10 Operating system Home edition. The provisioning was left out of this paper's experimented and assumed to have been carried at a prior time as per the above described protocol, or that the shared key has been instantiated manually during the platform initialization. For the personal security device, the experiment uses an Arduino Uno, which simulates the keypad. The Arduino Uno sends encrypted messages over a serial communication port to the application which creates an enclave. This experiment used AES-256 for encryption. The main overhead introduced in this scenario is the fact that the user needs to use an additional piece of hardware to enter his input.

5 RELATED WORK

Trusted Input: The lack of a trusted channel from the web user to the web server is a recognized problem. Since the release of Intel SGX, it became ap-

```
#endif
{
    /*************************************/

    arduino = new SerialPort(portName);
    while (!arduino->isConnected()) {
        arduino = new SerialPort(portName);
    }

    //Checking if arduino is connected or not
    if (arduino->isConnected()) {
        std::cout << "Connection established at port " << portName << endl;
    }


    std::cout << "Receiving data from arduino " << portName << endl;
    arduino->readSerialPort(incomingData, MAX_DATA_LENGTH);
    printf("%s \n", incomingData);

    std::cout << "Sending data to arduino " << portName << endl;
    arduino->writeSerialPort("hello\n", MAX_DATA_LENGTH);
```

Fig. 10. Arduino Serial Communication

parent that enclaves can play an important role in protecting security-sensitive input data from malicious client platforms. However, Intel SGX lacks support for trusted input channels to the end-user. Therefore, there have been several proposals to resolve this. One set of solutions relies on the hypervisor to establish generic secure IO channels, as is the case of SGXIO [27]. The other set of solutions uses external embedded hardware devices to establish a secure communication channel to either the SGX enclave of the web server. Indeed, the idea of using an external trusted embedded for trusted UI is not new, as it dates back to the 80s, before the proliferation of smart phones and personal security devices [24]. More recently, there has been a renewed interest in using these trusted devices with Intel SGX. Just as we were working on this research paper, SGX-USB proposed a solution with practically the same architecture as TrustUI [14]. Fidelius and ProximetEE also propose very similar architectures [11] [9]

Trusted Input: Authentication credentials are a common attack target for active Man-In-The-Browser malware. Therefore, many service providers, such as banks, propose two-factor authentication which rely on out-of-band verification to mitigate this attack. Such solutions can be defeated because of two fundamental weaknesses. On the one hand, the out-of-band modality such, as SMSs sent by the bank to confirm transaction details, are subject to user habituation. Users simply start to confirm the transactions out of habit without verifying the SMS details, or become inattentive to small transaction modifications such as similar bank accounts, or slightly rounded transaction amounts [5]. On the other hand, even if out-of-band verification is successfully carried out by the user, its effectiveness is only limited to attack scenarios where only dynamic page values are modified. It is, however, ineffective against attack scenarios where rogue elements are added into the web page, such as input fields prompting users to enter the passwords into them. Therefore, solutions which aim at improving the overall security of the browser software have been proposed. Promos integrates support for browsers at the OS-level [23]. Cloud terminal runs a lightweight secure thin terminal locally and outsources the rest of the security-sensitive computation

to a remote server [1]. Shadowcrypt uses a Shadow DOM to allow encrypted input/output for web applications [12].

6 Conclusions and Future Work

Users and web browsers make various security decisions based on client input and output channels. Therefore, this paper proposes TrustUI, a new architecture solution for enabling secure input and output channels from the end-user to the web server, despite the presence of a malicious browser, able to leak sensitive users' input, and modify the browser pages displayed to them. In order to mitigate the trusted output attacks, we design a two-component solution, where we first programmatically capture the screen directly from the frame buffer, then compare it to a reference trustworthy screen. The paper then experimentally evaluates the screen capture engine using an open source GDI API implementation. As for the trusted input attack, we propose to equip users with personal security devices with dedicated input keypads. These PSDs are able to establish an end-2-end encrypted channel to a web enclave which we establish within the browser's process address space. We use an Arduino Uno to simulate user's input into the PSD and send AES encrypted input messages to the application.

As a future work, we would like to implement and run both the image processing and screen capture engines within a more trustworthy execution environment, such as an Intel SGX enclave. In this case, TrustUI would need to utilize libraries which allow applications to run unmodified within the enclaves, such as Graphene [25]. Finally, TrustUI acknowledges the fact that it burdens the user with the need to carry an extra hardware device. Therefore, we would to explore as a future work to explore the feasibility of using the new android transaction confirmation API. This would allow TrustUI to use smartphones instead the personal security devices, while providing similar security guarantees and more usability [8].

References

1. 20th Annual Computer Security Applications Conference (ACSAC 2004), 6-10 December 2004, Tucson, AZ, USA. IEEE Computer Society (2004), http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=9473
2. Various methods for capturing the screen (2006), https://www.codeproject.com/Articles/5051/Various-methods-for-capturing-the-screen
3. Personal security devices (2017), https://ledger.readthedocs.io/en/latest/background/personal_security_devices.html
4. Intel sgx remote attestation (2019), https://qiita.com/Cliffford/items/095b1df450583b4803f2
5. Alzomai, M.H., AlFayyadh, B.: Display integrity assurance for sms transaction authorization (2011)
6. Cain, C.: Analyzing Man-in-the-Browser (MITB) Attacks (2014), https://www.sans.org/reading-room/whitepapers/forensics/analyzing-man-in-the-browser-mitb-attacks-35687

7. Costan, V., Devadas, S.: Intel sgx explained. Cryptology ePrint Archive, Report 2016/086 (2016), https://eprint.iacr.org/2016/086
8. Danisevskis, J.: Android protected confirmation: Taking transaction security to the next level (2018), https://android-developers.googleblog.com/2018/10/android-protected-confirmation.html
9. Dhar, A., Puddu, I., Kostiainen, K., Capkun, S.: Proximitee: Hardened SGX attestation and trusted path through proximity verification. IACR Cryptology ePrint Archive p. 902 (2018)
10. Dhar, A., Yu, D., Kostiainen, K., Capkun, S.: Integrikey: End-to-end integrity protection of user input. IACR Cryptology ePrint Archive p. 1245 (2017)
11. Eskandarian, S., Cogan, J., Birnbaum, S., Brandon, P.C.W., Franke, D., Fraser, F., Jr., G.G., Gong, E., Nguyen, H.T., Sethi, T.K., Subbiah, V., Backes, M., Pellegrino, G., Boneh, D.: Fidelius: Protecting user secrets from compromised browsers. CoRR (2018)
12. He, W., Akhawe, D., Jain, S., Shi, E., Song, D.: Shadowcrypt: Encrypted web applications for everyone. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 1028–1039. CCS '14, ACM, New York, NY, USA (2014)
13. Heckbert, P.: Color image quantization for frame buffer display. SIGGRAPH Comput. Graph. (3), 297–307 (Jul 1982)
14. Jang, Y.: Building Trust In The User IO In Computer Systems. Ph.D. thesis (2017)
15. Jøsang, A., Rosenberger, C., Miralabé, L., Klevjer, H., Varmedal, K.A., Daveau, J., Husa, K.E., Taugbøl, P.: Local user-centric identity management (1), 1 (2015)
16. Margaret Rouse: Man in the browser attack (2017), https://searchstorage.techtarget.com/definition/video-RAM
17. Ortolani, S., Crispo, B.: NoisyKey: Tolerating Keyloggers via Keystrokes Hiding. In: Proceedings of the 7th USENIX Conference on Hot Topics in Security. pp. 2–2. HotSec'12, USENIX Association (2012)
18. Palem, G.: Capturing the screen (2016), https://gk.palem.in/screencap.html
19. Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N.: The ghost in the browser analysis of web-based malware. In: Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets. pp. 4–4. HotBots'07, USENIX Association, Berkeley, CA, USA (2007)
20. Raj Samani, François Paget: Cybercrime exposed: Cybercrime-as-a-service (2013), https://www.networksasia.net/article/cybercrime-exposed-cybercrime-service-1372901941
21. Secure Box: Man in the browser attack, https://securebox.comodo.com/ssl-sniffing/man-in-the-browser-attack/
22. Szydlowski, M., Kruegel, C., Kirda, E.: Secure input for web applications. In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). pp. 375–384 (Dec 2007). https://doi.org/10.1109/ACSAC.2007.28
23. Ta-Min, R., Litty, L., Lie, D.: Splitting interfaces: Making trust between applications and operating systems configurable. In: OSDI (2006)
24. Tamrakar, S., Ekberg, J.E., Laitinen, P., Asokan, N., Aura, T.: Can hand-held computers still be better smart cards? In: Chen, L., Yung, M. (eds.) Trusted Systems. pp. 200–218. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
25. che Tsai, C., Porter, D.E., Vij, M.: Graphene-sgx: A practical library OS for unmodified applications on SGX. In: 2017 USENIX Annual Technical Conference (USENIX ATC 17). pp. 645–658. USENIX Association, Santa Clara, CA (2017)

26. Varmedal, K.A., Klevjer, H., ag, J.H., Jøsang, A., Vincent, J., Miralabé, L.: The OffPAD: Requirements and Usage. In: NSS (2013). https://doi.org/10.1007/978-3-642-38631-2-7
27. Weiser, S., Werner, M.: Sgxio: Generic trusted i/o path for intel sgx. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. pp. 261–268. CODASPY '17, ACM, New York, NY, USA (2017)
28. Ye, Z.E., Smith, S., Anthony, D.: Trusted paths for browsers. ACM Trans. Inf. Syst. Secur. (2), 153–186 (May 2005)