Securing Web Inputs Using Parallel Session Attachments

Ziqi Yang[1,2,3(✉)], Ruite Xu[1], Qixiao Lin[4], Shikun Wu[1], Jian Mao[4], and Zhenkai Liang[5]

[1] Zhejiang University, Hangzhou, China
{yangziqi,xuruite,wushikun}@zju.edu.cn
[2] ZJU-Hangzhou Global Scientific and Technological Innovation Center, Hangzhou, China
[3] Key Laboratory of Blockchain and Cyberspace of Zhejiang Province Governance, Hangzhou, China
[4] Beihang University, Beijing, China
{linqx529,maojian}@buaa.edu.cn
[5] National University of Singapore, Singapore, Singapore
liangzk@comp.nus.edu.sg

Abstract. Web applications have become a cornerstone of the critical cyber infrastructure powering our daily life. Untrusted browser environments, such as public computers and browsers with untrusted extensions, may expose sensitive data in web applications to attackers. One way to protect sensitive data in web sessions is to isolate it using a trusted environment, such as a trusted mobile phone. However, existing solutions either require modifications of web applications to incorporate the trusted environment, or require developers to manually pre-label sensitive data. To address these issues, we propose, WebTeleporter, a lightweight framework to protect users' sensitive input through a trusted mobile environment. It attaches to the original web session an independent secure parallel session that isolates sensitive input without any change to web applications. WebTeleporter is highly flexible, such that users can choose to opt in to the secure environment at any time, and choose sensitive input to protect on demand. Our evaluation demonstrates that WebTeleporter is compatible with 11 popular web applications and frameworks. It can protect 99% of pages that contain sensitive input. It takes low overhead to deploy WebTeleporter, which is a one-time effort for various applications. WebTeleporter introduces negligible performance overhead, i.e., 13.9% increase in loading time, and 0.37% decrease in throughput.

1 Introduction

Web applications are becoming a cornerstone of the critical cyber infrastructure powering our daily activities. While a wide variety of sensitive data is processed on the web platform, threats to data security and privacy arise from many

H. Duan et al. (Eds.): SecureComm 2023, LNICST 568, pp. 189–208, 2025.
https://doi.org/10.1007/978-3-031-64954-7_10

aspects. One particular threat is from untrusted browser environments, which may expose sensitive user data in web applications to attackers, such as passwords, credit card numbers, etc. For example, when users access web applications from desktop computers, the complex software stack underlying all web sessions, on which the security depends on, may be malicious including operating system, installed applications, malware exploit kit [34], browsers and extensions [25], and JavaScript in web pages [43]. In addition, monitoring and auditing software, e.g., that installed on enterprise desktop computers, may inadvertently capture and even store such sensitive input data in its logs.

Researchers have proposed solutions that utilize another secure device (including mobile phone) [6,10,28,32,39,40] to protect the security of web sessions. Felten et al. [6] propose to use the PalmPilot instead of smart cards to protect the private key. They implement a one-time password generator on the PalmPilot to provide a trusted authentication path, and users can decide on which device to display the encrypted sensitive data. Oprea et al. [32] use the PDA as a trusted device to protect sensitive data from being leaked to untrusted terminals when the user accesses her home computing environment through a "remote desktop" application. MP-Auth protocol [28] is proposed for online authentication by using a trusted mobile terminal to secure the password from malware on the untrusted computer. Richard et al. [40] leverage a trusted mobile device to protect the sensitive input and display of applications from the untrusted terminal. It needs an extra "thin-client server" as a relay to connect the mobile and the terminal. Richard et al. then [39] propose to split the input and the display of web applications between the untrusted computer and the trusted mobile. A browser extension acting as a Remote Desktop Client (RDC) agent is required on the client side. Session Juggler [10] transfers session data between an untrusted computer and a secure mobile device to let users input login credentials on the mobile side, thus protecting the credential data.

However, these solutions have several limitations. First, modifications of the web applications are required to incorporate the secure device making them not transparent to web applications. Second, web application developers need to prelabel the data (mostly manually) to protect and decide how the security device is integrated into the logic of a web application, which makes these approaches lack the flexibility for users to protect sensitive data on demand. Third, some of them [10] work only in the login phase, and leave the remaining web session unprotected. Such limitations urges the need of a more transparent, lightweight and flexible solution supporting continuous protection.

In this paper, we propose WebTeleporter, a framework that protects users's sensitive inputs via attaching an independent trusted parallel session to the original web session. WebTeleporter works as a middleware between a web application and a client (i.e., a browser), and creates a parallel secure session in a trusted device (i.e., a smart phone). Sensitive input fields are "teleported" to the trusted parallel session for protection without interaction with the web application, which makes WebTeleporter *transparent* to web applications. The incorporation of the parallel session requires no change of the web

application logic and no modifications of the browsers as well. The independence between WebTeleporter and web applications makes it *lightweight* and easy to deploy among various web applications in practice. Moreover, such independence also enables WebTeleporter to opt in to or out of the web session in a flexible manner. Users can decide which input fields to teleport in a simple "click-and-go" manner, which leads to a *flexible* protection on users' demand. With WebTeleporter, users retain the convenience of accessing their web applications from any devices of their choice, and are assured of no compromise of their sensitive input data.

Specifically, from each HTTP response, it detects each input field in the original session and specially marks it on the browser page. Once the user clicks on them, WebTeleporter transfers the selected input fields to the attached parallel session, and thus sensitive input fields are securely protected in the trusted session. When WebTeleporter receives HTTP requests from both the original session and the parallel session, it automatically combines the original session with sensitive data submitted in the parallel session to form one complete HTTP request for the web application. In this manner, users' sensitive data are submitted through a secure parallel session separated from the original session, with keeping it transparent to the web application.

There are multiple implementation choices for WebTeleporter. Considering the need of an safe input channel, we choose mobile devices as the client environment to create the secure parallel session. It greatly increases the adversary's effort to launch attacks as the adversary needs to control both the mobile device and the untrusted computer at the same time. For the server side, in order to support HTTPS connections, we made WebTeleporter a module of Apache server [5] positioned between the Apache core and the TLS module, such that the web data are not encrypted for WebTeleporter.

We successfully deployed WebTeleporter on 11 popular web applications and frameworks in various categories, e.g., Elgg [16] and Wordpress [46]. WebTeleporter is able to protect 99% of web pages with sensitive input. In our deployment, it takes seconds to setup WebTeleporter, which is a one-time task for all web applications. After deploying WebTeleporter, the average loading time for each page incurs only 13.9% overhead, and the average throughput for each application incurs only 0.37% reduction. Our experiments demonstrate that WebTeleporter is compatible with existing web applications, and is easy to deploy. It introduces negligible performance overhead.

Contributions. In summary, we make the following contributions in the paper.

- We propose the *parallel session attachment* mechanism to protect important input data in web sessions. It provides flexible protection of web data and is transparent to server-side applications.
- We design WebTeleporter as a middleware-based framework to protect users' sensitive input using parallel session attachments. It creates the secure parallel session using mobile devices. It is lightweight to deploy among web applications without any change of them.

– We prototyped WebTeleporter and evaluated it using 11 open-source web applications. Our evaluation with several real-world web applications and application frameworks shows that WebTeleporter is applicable to existing applications. It is easy to deploy and incurs negligible performance overhead.

2 Problem Overview

2.1 Threat Model and Scope

In this work, we aim to protect input data of web sessions on untrusted computers, such as username, password, credit card information, answers of security questions, etc. The sources of threats include malware in the device's operating system and software, compromised browsers, plugins and extensions, as well as malicious JavaScript code injected into a web application. Adversaries can collect any data on web pages through these threats. We assume that users have access to a trusted mobile device with secure network connection.

The network connection of the computer is another source of threat, but here we only consider attacks that aim at exposing users' data, e.g., eavesdropping [20,48], without significantly altering the functionality of the web application. Regular phishing attack [4,45] is not considered in this work. We also assume that if a web application adopts HTTPS, it does not further perform any application-level encryption. The evaluation we performed on 11 applications implies that this hypothesis is widely supported in practice.

As an initial proof of concept, we consider only input fields that is submitted through one HTTP request or XMLHttpRequest in this work, i.e., by clicking a "submit" button via POST message either in an HTTP form or by AJAX function. "Transmit as user types" by monitoring input field changes, e.g., via listening on the `OnInput` event, is out of scope.

2.2 Motivating Example

We motivate our work with a real-world login example as shown in Fig. 1b and Fig. 1c, where the user is required to input username and password to login to the web application. The login UI is rendered from the HTML code as presented in Fig. 1b. The username and password input boxes are composed by the two `<input>` elements. When the user submits the login form, the browser sends a POST request to the web application with the username and password encapsulated in the "Form Data" field of the request (as labeled in the red rectangle in Fig. 1c). In this way, the username and password are transmitted to the web application from the untrusted computer browser.

Suppose the password is sensitive data and the username is insensitive. The idea of our approach is to protect the password by transfering it to a secure client environment that is isolated from the untrusted computer. To understand the problem, we need to take a close look at how web sessions are established and processed during browsing.

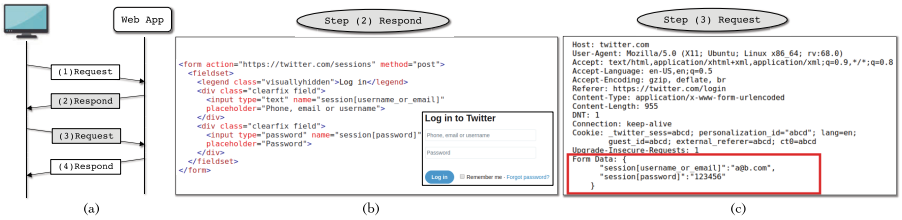


Fig. 1. An illustrative protocol of a typical web sessions (Twitter). (Color figure online)

A web session refers to the communication between a browser and a web server. It represents the time between a user's first arrival at a page in the site until the time she stops using the site. A web session is defined as a server-side storage of information that is desired to persist throughout the user's interaction with the web application. Instead of storing large and constantly changing information via cookies in the user's browser, only a unique identifier, namely *session ID*, is stored on the client side, which is assigned by the server and used to keep track of this session. This session ID is passed to the web server each time the browser makes an HTTP request. The web application pairs this session ID with its internal database and retrieves the stored variables for use by the requested page [26]. An illustrative protocol of a typical web session is presented in Fig. 1a. It is initiated by the web browser when a user visits the website hosted on the web server. During a web session, a user's data, some of which is sensitive, is transmitted between a client and a server via HTTP requests and responses. An example is shown in Fig. 1b and Fig. 1c. In this paper, we focus on protecting sensitive data in web sessions.

2.3 Design Goals

There are several objectives in devising a practical solution for such web session protection needs.

- **Flexibility & Usability.** It is important to allow users to flexibly decide which input data to protect in a web page, especially in a limited display space on a mobile device [3], for the sake of usability and security reason. In addition, it should be convenient for a user to opt in to or out of the system without the trade-off of user experience.
- **Transparency.** Transparency to web servers is quite challenging but highly desirable as doing so significantly smoothing out the deployment of the solution without requiring retrofitting web application code.
- **Compatibility with HTTPS.** To enhance web security, HTTPS connections are widely deployed by more and more websites. This setting makes existing solutions that leverage session hijacking technique invalid. It is challenging for a system to be transparent to web applications and to also support HTTPS.

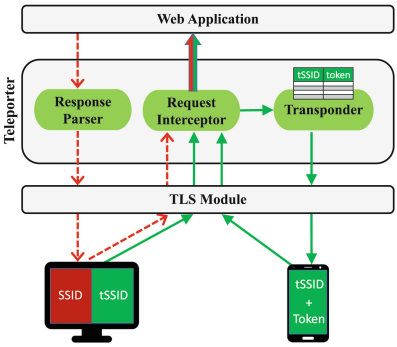


Fig. 2. WEBTELEPORTER architecture.

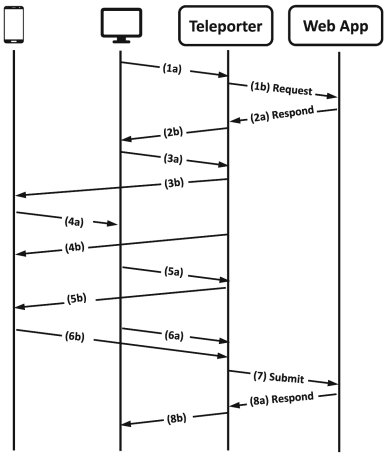


Fig. 3. WEBTELEPORTER protocol.

3 Design

3.1 Approach Overview

We present the architecture of WEBTELEPORTER in Fig. 2. WEBTELEPORTER is designed as a middleware between the TLS module of the web server and the web application. Such design enables it to support HTTPS connections of the web application because all the web data is decrypted to plain data in this middleware. WEBTELEPORTER works by creating an additional secure *parallel session attachment* to the original web session, such that sensitive input data in the original session can be securely transferred to and protected in the parallel session. Let *SSID* represent the session ID that indexes the original web session. WEBTELEPORTER creates another session ID, denoted as *tSSID*, to index the parallel session. The tSSID is stored in both the computer's browser and the mobile browser, and is managed in an internal table of WEBTELEPORTER. Note that the parallel session is independent from the original session, which makes WEBTELEPORTER work in the entire session and also easy to deploy for different web applications. The mobile browser is assigned a secret token by WEBTELEPORTER for authentication. WEBTELEPORTER consists of three components: Response Parser (RP), Request Interceptor (RI), and Transponder (TP). RP hooks the outgoing web data and injects WEBTELEPORTER's code. RI hooks the ingoing web data from both the computer and the mobile phone, and integrates them into one submission. TP transfers sensitive input in the computer's browser to the mobile browser.

The protocol of WEBTELEPORTER is presented in Fig. 3. Whenever an HTTP response is produced by the application (step 2a and 8a), RP parses the web

page and injects auxiliary JavaScript code into it (step 2b and 8b). On the one hand, the JS code collects any input fields (elements) on the current page and draws a UI for users to choose sensitive input. It sends the user's selection to TP (step 3a) and TP pushes the selected input fields to the mobile browser (step 3b). On the other hand, it retrieves a new tSSID generated by TP and stores it in the computer's browser if no parallel session is created. The mobile browser joins the parallel session by accessing TP with the same tSSID which is obtained from the computer's browser (step 4a). TP verifies it and returns a secret token to the mobile browser (4b). At this point, the parallel session is fully created and indexed by the tSSID. The user submits insensitive data from the computer's browser as usual (step 6a). RI intercepts such HTTP request, and holds it, awaiting sensitive input data from the mobile side (step 6b). Afterwards it integrates the earlier HTTP request with the sensitive input into one request for the web application (step 7). Note that the order of step 6a and 6b can be swapped. Throughout the process, there is no web logic change in terms of the web application. Hence, WebTeleporter is transparent to the web application.

The concurrency of WebTeleporter is handled by the web server. For example, we implement WebTeleporter as a module of Apache server. For each new request, Apache server will launch WebTeleporter individually. All WebTeleporter instances share the same database which contains the tSSID, secret token and other storage.

3.2 Parallel Session Attachments

WebTeleporter leverages an independent session from the original web session to manage the multiple input devices (i.e., computer and mobile phone), which we refer to as parallel session.

The parallel session is indexed by tSSID, which is stored in three places: the computer's browser, the mobile browser and WebTeleporter. It helps WebTeleporter to manage the computer and the mobile device. When the computer's browser visits the web application for the first time, the auxiliary JS code injected by RP in the response page asks TP for a new tSSID, and stores it in the browser. The parallel session is fully established when the mobile device joins, and ends when it is detached by users. The mobile device scans the QR code produced by the injected JS code on the computer page that directs the mobile browser to access TP to establish the parallel session. Then it navigates to TP along with the obtained tSSID. TP verifies the tSSID and returns a secret token to the mobile browser so the mobile device can be authenticated. At this point, the parallel session is fully established. Afterwards, WebTeleporter effectively manages both devices by the tSSID and secret token. The parallel session can be terminated at anytime when the user clicks the "detach" button on the mobile page. The mobile device can be added again by repeating the above process.

3.3 Input Element Migration

Input element migration (parallel) leads to the isolation of sensitive input from the untrusted computer, and further protection of it on the secure mobile device. To teleport elements between sessions, WebTeleporter interacts with the two devices to obtain the selection of users' intended input fields, and push them to the mobile side.

Input element migration can be performed after the parallel session is fully established. The injected JS code scans the whole web page to collect any input fields (i.e. either statically or dynamically), generating an interface for the user to choose the sensitive input fields, e.g., by appending a checkbox widget to the input field. It tells WebTeleporter the user's selection by accessing TP. TP then pushes the selected input to the mobile device. The communication protocol between TP and the mobile browser has many choices, such as HTTPS and WebSocket [19]. Note that the selected input elements on the computer page are set as disabled and filled with dummy data. This prevents the users from accidentally typing any sensitive data, and also to bypass any local input validation check.

3.4 Input Integration

WebTeleporter integrates the two HTTP requests from both devices into one complete request for the web application, which is transparent to the web application. In this work, we consider only the `<form>` element for submitting the POST message, where key-value pairs compose the submitted data.

The process of input integration is as follows: RI hooks each HTTP request from both devices, and accomplishes input integration. It takes hold of the form data in the computer request if any input fields in it are previously teleported. When the mobile browser submits sensitive data, RI extracts the sensitive data from it to fill the form data in the computer request by looking up the keys of the sensitive input fields. At this point, the input data from the two devices are integrated into one complete form. Such complete form is then forwarded to the web application. Throughout the input integration process, the sensitive data submitted from the mobile device is securely isolated from the untrusted computer and its network. It is transparent to the web application. As shown in Fig. 3, the web application sees only the integrated submission.

3.5 Security Analysis

WebTeleporter not only improves the login security by limiting adversary's effectiveness to allowing it only to capture short-lived session credentials rather than long term ones like Session Juggler, but also supports flexible protection over all sensitive input data during the entire session even in HTTPS connections. Through parallel, users' long term sensitive input is never inputted or transferred to the untrusted computer, thus deviating from any security threats

on it. Furthermore, it effectively mitigates the risks of using an untrusted network connection by transmitting sensitive data over a mobile network (4G/5G) which is encrypted. By establishing an independent parallel session, it authenticates the user's mobile device, and only accepts eligible mobile requests, so it can protect the integrity of users' sensitive input as well.

We analyze several possible attacks to demonstrate the security of WebTeleporter. The major source of threats comes from the untrusted computer, where the injected JS code runs. First, the adversary could replace the URL in the QR code with a malicious one to phish users. Although such phishing attacks are out of scope, it is mitigated by our specially developed mobile application (i.e., browser), which has a user consent popup that emphases the domain request, and a domain blacklist check. Second, the adversary could also tamper the tSSID in the QR code, or even register and occupy the tSSID before the victim. This will lead to an unsuccessful registration of the user's mobile device such that no valid parallel session is established. The user is not able to input anything on the mobile device. Such denial-of-service attack implies that the computer is strongly compromised, and the user is supposed to stop operating it. Third, the adversary could prevent teleport by stopping the injected JS code, or forge the user-selected input fields on the computer's browser to phish users. Such attacks also imply denial-of-service threats, and users can easily refuse to input sensitive data on the computer.

In summary, WebTeleporter guarantees the security invariant that any input fields pushed to the mobile device are secure to input, and the sensitive data is submitted to WebTeleporter confidentially and completely. This provides users with a trusted environment to handle sensitive data.

4 Implementation

We implement WebTeleporter as a module of Apache HTTP server [5] of version 2.4.34 on port 80. The Apache HTTP server comprises a small core for basic functionality, and many additional modules to extend the core functionality for special purposes. Information transmission between modules relies on the core. Modules are registered to the core in a defined order, i.e., forming a filter chain, to serve a request appropriately.

The Response Parser (RP) and Request Interceptor (RI) components are implemented as Apache output filter module and input filter module respectively. They are registered between the Apache Core and the SSL module in the filter chain to support HTTPS connections. The Transponder (TP) component is implemented as an Golang [35] application within an independent Golang web server listening on port 8080. In order to bypass the Content Security Policy (CSP) [41] check on the different port number of TP, we set on the Apache server a reverse proxy rule where connections to TP on port 80 will be internally re-directed to port 8080 on which the Golang server is listening. Hence, the Golang server is transparent to web pages.

RP parses each HTML response from the application, and injects external JavaScript in its `<head>` element. Such script will run after the page is fully

loaded and the `<body>` element is rendered. It then connects to TP to fetch the teleportation session ID (tSSID) for the creation of the QR code. It also detects any input fields, e.g., `<input>`, `<select>`, `<textarea>` elements on the page. We implement the user interface to choose sensitive input by appending a checkbox element to each input element. The script is able to collect users' choice and send it to TP via a POST request. In the meantime, the script disables selected input elements with also filling them with default dummy data. This is to bypass any local empty check on these inputs.

TP manages the teleportation session, and also works as the bridge between the two devices. TP creates the attached teleportation session with tSSID, and sends it to the two devices. We implement the communication between TP and the mobile device using WebSocket protocol [19] which enables sending messages to a server and receiving event-driven responses without having to poll the server for a reply. TP pushes the selected inputs to the mobile browser, and stores the mobile device's submission in an internal storage which RI can also access.

RI integrates submissions of two devices into one complete submission, and then forwards it to the application. In particular, it fills the computer's POST submission with the mobile device's submission by looking up from the internal storage. Then a complete POST message is forwarded to the application.

5 Evaluation

5.1 Experimental Setup

We perform our experiments on an Apache HTTP server of version 2.4.34 on Ubuntu 14.04, where PHP (v5.6.32), MySQL server (v5.7), and Golang server (v1.6) are installed. The server consists of Intel®Xeon(R) CPU E5-2660 v3 @ 2.60 GHz × 16, and 64 GB memory.

We evaluate WebTeleporter on 11 open-source web applications from 5 different categories including 3 frameworks (WordPress [46], Joomla [24] and Drupal [15]), described in Table 5 in Appendix A.

5.2 Feasibility and Effectiveness

We evaluate the feasibility of WebTeleporter by testing its installation time and if it runs correctly for each application. We demonstrate its effectiveness in protecting sensitive input by measuring the success rate R of WebTeleporter in protecting pages that contain sensitive input for each application. The success rate R is measured by $R = 1 - \frac{number\ of\ failed\ pages}{number\ of\ tested\ pages}$.

Feasibility. It takes less than 10 s to install WebTeleporter on the Apache Server including configurations. Once installed, WebTeleporter is able to seamlessly work for any of the 11 applications with no further manual effort. This demonstrates that WebTeleporter is very lightweight and easy to deploy for various applications. We further calculate the portion of input boxes that submit data via POST requests and show the statistics in Table 1. Among the 11

evaluated applications, over 99% of the sensitive input boxes send data through POST requests. This result demonstrates the practicability of our threat model.

Effectiveness. For each application, we test WebTeleporter on the pages that contain sensitive input as shown in Table 1. In total, WebTeleporter is able to protect the sensitive input data in the trusted mobile device with 99% success rate. The only 2 failed cases on Elgg and ZenCart are caused by the same reason. In particular, there are more than two input elements with the same `name` attribute in the failed pages. The injected script fails to handle such cases, so it cannot correctly detect users' selection. However, these issues can be simply addressed by refining the script.

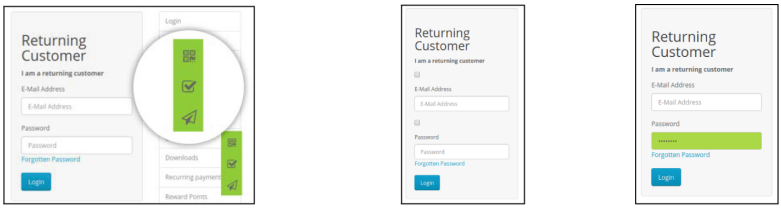
Table 1. Statistics of input boxes and input boxes that use POST requests and effectiveness of WebTeleporter in protecting web pages that contain sensitive input. Note that IB stands for Input box, S and I stand for sensitive and in-sensitive respectively.

Application	Statistics						Effectiveness		
	IB	POST	S-IB	S-POST	I-IB	I-POST	#Tested Pages	#Failed Pages	Success Rate
Elgg	78	78	29	29	49	49	20	1	95
HotCRP	145	139	68	68	77	71	14	0	100
OpenConf	62	62	31	31	31	31	10	0	100
PrestaShop	161	159	90	88	71	71	24	0	100
OpenCart	185	185	96	96	89	89	30	0	100
osCommerce	109	105	60	60	49	45	23	0	100
Zencart	39	39	34	34	5	5	17	1	94.1
WordPress	77	77	41	41	36	36	17	0	100
Joomla	76	76	29	29	47	47	23	0	100
Drupal	112	112	20	20	92	92	20	0	100
Piwigo	28	27	14	14	14	13	9	0	100
Total	1072	1059	512	510	560	549	207	2	99
Percentage		98.79%		99.61%		98.04%			

5.3 Case Study: OpenCart

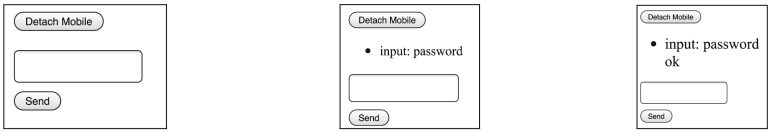
We use OpenCart as a case study to illustrate in details how WebTeleporter works. Figure 4a shows the main control panel of WebTeleporter, an attached floating widget at the bottom right corner of the browser window.

Teleportation Session Creation. The options from top to bottom in the control panel are "Show QR code", "Choose input boxes", and "Teleport". When the first option is clicked, the script queries TP for tSSID to generate the QR code. For example, the QR code content could be http://www.example.com/teleporter?ACTION=attachMobile&tSSID=1534681405BsB42eMb8F, where http://www.example.com/teleporter is the URL of TP, and `ACTION` tells TP the operation, and `tSSID` is the pre-fetched tSSID. When the mobile browser visits such URL, it is directed to a default page of WebTeleporter as shown in Fig. 5a.



(a) (b) (c)

Fig. 4. UI of WebTeleporter on the computer browser. (Color figure online)



(a) (b) (c)

Fig. 5. UI of WebTeleporter on the mobile browser.

This page establishes a WebSocket connection with TP, and then enters a "chat" mode with TP, awaiting further input messages. At this point, an teleportation session is successfully attached to the web session with tSSID as the reference. Note that the "Detach Mobile" button is used to detach the parallel session from the original session at any time.

Page Teleportation. To select input boxes, WebTeleporter appends an checkbox widget to each input box as shown in Fig. 4b, which is triggered by clicking the second button in the control panel. Suppose the user chooses to teleport "password" box by clicking the third button in the control panel. The injected script will disable the password box and set its color to green to make it look different from other input boxes. Besides, the script also fills it with default dummy data to bypass any local empty check, to avoid submission failure from the computer.

Input Integration. Once the password box is teleported, TP pushes the its key to the mobile browser as shown in Fig. 5b. Then the user can securely input her password in the trusted mobile browser and submit. TP receives the password and saves it in an internal storage which RI can also access. In the meanwhile, a feedback message is sent to the mobile browser to show if the recent submission is successful or not. Figure 5c shows a successful case. The user then continues inputting her email address on the untrusted computer browser and submits. RI intercepts this request and fills the POST form with the password extracted from the internal storage. Then a complete submission is forwarded to the web application.

5.4 Performance Overhead

The performance overhead of WebTeleporter is mainly caused by the additional processing of the Response Parser component and the additional running time of the scripts it injects. We evaluate the performance overhead by measuring the additional page loading time and the throughput reduction of each application.

Page Loading Time. We measure the loading time of the login page of each application without and with deploying WebTeleporter. In particular, we use Chrome of version 67.0.3396.99 to visit the remote Apache web server. We disable the cache function of Chrome. Each page is loaded 30 times to compute its average loading time. Table 2 presents results. We can see that on average WebTeleporter introduces 13.9% overhead in the page loading time.

Table 2. Loading time (seconds) of each application without and with deploying WebTeleporter

Application	Loading Time		Diff	Overhead
	Without	With		
Elgg	0.897	1.008	0.111	12.4%
HotCRP	0.190	0.235	0.045	23.7%
OpenConf	0.041	0.053	0.012	29.3%
PrestaShop	0.747	0.789	0.042	5.6%
OpenCart	0.563	0.658	0.095	16.9
ZenCart	0.327	0.397	0.070	21.4%
osCommerce	0.323	0.379	0.056	17.3%
WordPress	0.294	0.321	0.027	9.2%
Joomla	0.360	0.413	0.053	14.7
Drupal	0.177	0.197	0.020	11.3%
Piwigo	0.191	0.235	0.044	23.3%
Average	0.374	0.426	0.052	13.9%

Throughput. Throughput, in a way reflecting the capacity of the web server, is measured as total number of transactions or requests sent to a web server in a given time (e.g., per second). We use Apache JMeter [21] to measure the throughput. We set the ramp-up period[1] to 15 s. To measure the maximum throughput for each web application, we gradually increase the number of threads (from zero) until response error occurs. The increasing interval is 100 threads when the number of threads is smaller than 1,000. The interval is changed to 1,000 threads when the number of threads is larger than 1,000. Table 3 presents the results. WebTeleporter on average introduces 0.37% reduction in the throughput, which is considered negligible.

[1] Ramp-up period indicates the time duration for each thread to start.

Table 3. Throughput (transactions per second) of each application without and with deploying WebTeleporter

Application	# Threads	Throughput		Diff	Reduction
		Without	With		
Elgg	800	46.1	44.4	1.7	3.69%
HotCRP	6,000	377.1	376.8	0.3	0.08%
OpenConf	3,000	194.1	192.8	1.3	0.67%
PrestaShop	300	14.5	14.3	0.2	1.38%
OpenCart	10,000	543.9	540.5	3.4	0.63%
ZenCart	1,000	57.7	57.0	0.7	1.21%
osCommerce	13,000	540.2	538.9	1.3	0.24%
WordPress	1,000	65.2	65.2	0	0%
Joomla	500	28.8	28.7	0.1	0.35
Drupal	3,000	193.9	193.9	0	0%
Piwigo	6,000	372.5	372.4	0.1	0.03%
Average	4055	221.27	220.45	0.82	0.37%

5.5 Comparison with Prior Work

We compare WebTeleporter with prior work that utilizes a trusted mobile device to protect sensitive data, and present comparison results in Table 4. WebTeleporter is transparent to web applications because it does not modify any source code of applications, but prior work has to change the application logic to incorporate the trusted device. Session Juggler [10] does not change application source code but it is not flexible. WebTeleporter is highly flexible in the sense that it enables users to select sensitive input to protect on demand, while prior work does not have such flexibility. Before using systems in prior work, a user has to install either an ad hoc browser or a mobile application. While in WebTeleporter, only a regular mobile browser with the function of scanning QR code is needed.

Table 4. Comparison of WebTeleporter and prior work

Approach	Security	Trusted Env.	Transp.[a]	HTTPS	Flexib.[b]	Extra Server	User Effort
PDA [32]	Sensitive input	Mobile device		✓			Purchase ad hoc augmented PDA device
MP-Auth [28]	Login password	Browser & mobile device		✓			Install ad hoc browser and mobile app
Thin-Client [40]	Sensitive input & output	Mobile device		✓		✓	Install ad hoc mobile app
Split-Trust [39]	Sensitive input & output	Browser & mobile device		✓			Install ad hoc browser extension and mobile app
Session-Juggler [10]	Login password	Browser & mobile device	✓	✓		✓	Install ad hoc bookmarklet and mobile app
UbiKiMa [18]	Login password	Browser & mobile device		✓		✓	Install ad hoc bookmarklet and mobile app
Fidelius [17]	Sensitive input	SGX	✓	✓			additional trusted devices and a browser component
WebTeleporter	Sensitive input	Mobile device	✓	✓	✓		A built-in mobile browser is sufficient, such as Chrome (with built-in scanning function).

[a]Transp. is short for Transparency
[b]Flexib. is short for Flexibility.

6 Discussion

We integrate WebTeleporter into the Apache server framework. Nonetheless, WebTeleporter is not limited to this implementation choice. For example, it can be implemented as a standalone reverse proxy server as well. However, it could be more tricky to support HTTPS connections in this setting. One possible solution is to make the proxy a Man-In-The-Middle (MITM) proxy with a digital certificate. The untrusted browsers have to install such certificate locally if it is not trusted by the default CA. Besides, manual proxy setting of the browsers are required. Nevertheless, certificate installation and proxy setting might require administrative privileges which are usually not available in public computers.

WebTeleporter is not limited to only input data. It can be easily tuned to protect output data (e.g., display data) as well, because both input and output data are DOM objects in terms of WebTeleporter with no difference. Nonetheless, the challenging part is that sensitive output data has to be transferred to the trusted mobile device before it reaches the untrusted computer. Not like input data, it is too late for users on the untrusted computer to choose what output data to be protected, because the sensitive output data could have already been exposed to attackers. Therefore, this requires WebTeleporter to automatically identify sensitive output data and push it to the trusted mobile device at the server side. We leave automatically identifying sensitive data as future work.

WebTeleporter, as an attachment of the original web session, seamlessly involves another device in the web session, which makes WebTeleporter easy to be extended for further potential scenarios. For instance, it can be tuned to support more than two devices not only for security reasons, for example, in the scenario of multi-party cooperations. We leave this as future work.

WebTeleporter provides a general framework to incorporate a trusted environment into web sessions. Such trusted environment is not limited to the mobile phone. Other forms of trusted environments such as SGX [12] and TrustZone [38] can be also supported by implementing the communication with TP. In future work, we will explore more trusted environments.

7 Related Work

7.1 Privilege Separation in Web Applications

The majority of previous approaches require the modification of applications, and are deployed based on the trust of browsers [1,2,9,11,14,44]. Few approaches only relies on the trusted operating system [13] or the hardware [23,27], e.g., TrustZone.

Akhawe et al. propose a method for privilege separation in HTML5 applications [2], in which HTML5 applications can create an arbitrary number of child components with no privileges. Further, They also design a data-confined sandbox to confine data for HTML5 applications [1]. UserPath [9] introduces user sub-origins to isolate user's sensitive data and and privileged code. PathCutter [11] divides a web application into different views and then isolates the different views at the client side to defend XSS JavaScript worms. AdSentry [14] is an isolation framework to confine JavaScript-based advertisements. However, all these methods are based on the trusted browser and also need modification of web applications.

Cryptons [13] introduces a small, trusted crypton-kernel that runs in a separate OS process and sandboxes of the untrusted browsers to protect the sensitive data of web applications. DroidVault [27] is based on the trusted hardware TrustZone to isolate sensitive data. Sensitive data remain encrypted throughout its lifetime. Defensive JavaScript (DJS) [7] is proposed to realize language-based isolation from untrusted JavaScript in the same origin of web applications. However, all these approaches need the modification of web applications to deploy.

In [51], they propose a method to protect private web content from embedded scripts. In UbiKiMa [18], it proposes the use of strong cryptography to protect users' login credentials on web applications. GuarDroid [42] is proposed to protect user's password from untrusted apps. [8] presented a web authorization delegation scheme for mobile devices that utilizes a native companion app, the MobileAuthenticator, to realize a trusted UI. [17] uses trusted hardware enclaves integrated into the browser to enable protection of user secrets during web browsing sessions, even if the entire underlying browser and OS are fully controlled by a malicious attacker.

7.2 Mobile Device-Based Solutions

There are extensive researches on using a separate device to secure the web application authentication [6,10,28,32,39,40]. Existing solutions all require the server-side and/or client-side changes to incorporate security devices and also

lack flexibility for users to customize sensitive data on demand and even apply the system or not.

In addition, another research branch is to establish "trusted-path" through the untrusted components [29,47,49,52]. Wu et al. propose "Web Wallet", a solution that provides a newly designed UI to prevent user credentials leakage from the phishing attacks [47]; Zhou et al. present a mechanism for verifiable trusted path establishment on commodity x86 computers [52].

However, none of the above approaches protect sensitive data and operations in web applications during the entire session.

8 Conclusion

Accessing web applications on an untrusted computer poses high risk to security and privacy of sensitive data in the web session. In this paper, we present WebTeleporter, a lightweight middleware-based framework to protect sensitive input in an independent teleportation session beyond the original web session. The teleportation session is securely isolated using a trusted environment, e.g., a trusted mobile device. It is flexible for users to opt in to the system at any time, and to choose sensitive input to protect. Our experiments demonstrate that WebTeleporter is very easy to deploy with negligible manual effort. It is compatible with various real-world web applications. Moreover, WebTeleporter introduces negligible performance overhead.

Acknowledgements. This work was supported by National Natural Science Foundation of China (62102353).

A Appendix

Table 5. Web applications evaluated in our experiments

Category	Application	Version	Popularity
Social Network	Elgg [16]	2.3.7	>2,800,000 downloads
Conference Management	HotCRP [22]	2.101	Used by USENIX, SIGCOMM, etc.
	OpenConf [31]	6.81	Used by ACSAC, IEEE, W3C, ACM, etc.
E-Commerce	PrestaShop [37]	1.7.4.1	Powering >250,000 online stores
	OpenCart [30]	3.0.2.0	Powering >270,000 live websites
	ZenCart [50]	1.5.5	Used by >0.2% of all web sites worldwide
	osCommerce [33]	2.3.4	>12,000 registered sites
Content Management	WordPress [46]	4.9.7	Used by 50,000 new sites every day
	Joomla [24]	3.8.10	>50,000,000 downloads
	Drupal [15]	8.5.5	Used by >2.1% of all web sites worldwide
File Sharing	Piwigo [36]	2.9.3	Translated into 58 languages

References

1. Akhawe, D., Li, F., He, W., Saxena, P., Song, D.: Data-confined HTML5 applications. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 736–754. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40203-6_41
2. Akhawe, D., Saxena, P., Song, D.: Privilege separation in HTML5 applications. In: USENIX Security (2012)
3. Amrutkar, C., Kim, Y.S., Traynor, P.: Detecting mobile malicious webpages in real time. IEEE Trans. Mob. Comput. **16**(8), 2184–2197 (2017). https://doi.org/10.1109/TMC.2016.2575828
4. Amrutkar, C., Traynor, P., van Oorschot, P.C.: An empirical evaluation of security indicators in mobile web browsers. IEEE Trans. Mob. Comput. **14**(5), 889–903 (2015). https://doi.org/10.1109/TMC.2013.90
5. Apache: HTTP server project. https://httpd.apache.org/
6. Balfanz, D., Felten, E.W.: Hand-held computers can be better smart cards. In: Proceedings of the 8th Conference on USENIX Security Symposium, SSYM 1999, vol. 8, p. 2. USENIX Association, Berkeley (1999). http://dl.acm.org/citation.cfm?id=1251421.1251423
7. Bhargavan, K., Delignat-Lavaud, A., Maffeis, S.: Language-based defenses against untrusted browser origins. In: USENIX Security, pp. 653–670. Citeseer (2013)
8. Braun, B., Koestler, J., Posegga, J., Johns, M.: A trusted UI for the mobile web. In: Cuppens-Boulahia, N., Cuppens, F., Jajodia, S., Abou El Kalam, A., Sans, T. (eds.) SEC 2014. IAICT, vol. 428, pp. 127–141. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55415-5_11
9. Budianto, E., Jia, Y., Dong, X., Saxena, P., Liang, Z.: You can't be me: enabling trusted paths and user sub-origins in web browsers. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 150–171. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11379-1_8
10. Bursztein, E., Soman, C., Boneh, D., Mitchell, J.C.: SessionJuggler: secure web login from an untrusted terminal using session hijacking. In: Proceedings of the 21st International Conference on World Wide Web, pp. 321–330. ACM (2012)
11. Cao, Y., Yegneswaran, V., Porras, P., Chen, Y.: PathCutter: severing the self-propagation path of XSS JavaScript worms in social web networks. In: Network and Distributed System Security Symposium (2012)
12. Costan, V., Devadas, S.: Intel SGX explained. IACR Cryptology ePrint Archive 2016/086, pp. 1–118 (2016)
13. Dong, X., Chen, Z., Siadati, H., Tople, S., Saxena, P., Liang, Z.: Protecting sensitive web content from client-side vulnerabilities with CRYPTONS. In: Proceedings of the 20th ACM Conference on Computer and Communications Security (2013)
14. Dong, X., Tran, M., Liang, Z., Jiang, X.: AdSentry: comprehensive and flexible confinement of JavaScript-based advertisements. In: ACSAC (2011)
15. Drupal: Open source CMS. https://www.drupal.org/
16. Elgg: introducing a powerful open source social networking engine. https://elgg.org/
17. Eskandarian, S., et al.: Fidelius: protecting user secrets from compromised browsers. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 264–280 (2019). https://doi.org/10.1109/SP.2019.00036
18. Everts, M., Hoepman, J.H., Siljee, J.: UbiKiMa: ubiquitous authentication using a smartphone, migrating from passwords to strong cryptography. In: Proceedings of the 2013 ACM Workshop on Digital Identity Management, pp. 19–24. ACM (2013)

19. Fette, I., Melnikov, A.: The WebSocket protocol. Technical report (2011)
20. Ghaderi, M., Goeckel, D., Orda, A., Dehghan, M.: Minimum energy routing and jamming to thwart wireless network eavesdroppers. IEEE Trans. Mob. Comput. **14**(7), 1433–1448 (2015). https://doi.org/10.1109/TMC.2014.2354031
21. Halili, E.H.: Apache JMeter: A Practical Beginner's Guide to Automated Testing and Performance Measurement for Your Websites. Packt Publishing Ltd. (2008)
22. HotCRP: HotCRP conference management software. http://read.seas.harvard.edu/~kohler/hotcrp/
23. Hwang, D., Yeleuov, S., Seo, J., Chung, M., Moon, H., Paek, Y.: Ambassy: a runtime framework to delegate trusted applications in an ARM/FPGA hybrid system. IEEE Trans. Mob. Comput. **22**(2), 708–719 (2023). https://doi.org/10.1109/TMC.2021.3086143
24. Joomla: The CMS trusted by millions for their websites. https://www.joomla.org/
25. Kapravelos, A., Grier, C., Chachra, N., Kruegel, C., Vigna, G., Paxson, V.: Hulk: eliciting malicious behavior in browser extensions. In: Proceedings of the 23rd USENIX Security Symposium (2014)
26. LassoSoft: Understanding cookies and sessions. http://www.lassosoft.com/Tutorial-Understanding-Cookies-and-Sessions
27. Li, X., Hu, H., Bai, G., Jia, Y., Liang, Z., Saxena, P.: DroidVault: a trusted data vault for Android devices. In: 2014 19th International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 29–38. IEEE (2014)
28. Mannan, M., van Oorschot, P.C.: Using a personal device to strengthen password authentication from an untrusted computer. In: Dietrich, S., Dhamija, R. (eds.) FC 2007. LNCS, vol. 4886, pp. 88–103. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77366-5_11
29. McCune, J.M., Perrig, A., Reiter, M.K.: Safe passage for passwords and other sensitive data. In: Network and Distributed System Security Symposium (2009)
30. OpenCart: Open source shopping cart solution. http://www.opencart.com/
31. OpenConf: Peer-review, abstract and conference management. http://www.openconf.com/
32. Oprea, A., Balfanz, D., Durfee, G., Smetters, D.K.: Securing a remote terminal application with a mobile trusted device. In: 2004 20th Annual Computer Security Applications Conference, pp. 438–447. IEEE (2004)
33. osCommerce: Creating online stores worldwide. http://www.oscommerce.com/
34. Technical White Paper: Reversal and analysis of Zeus and SpyEye banking trojans. http://www.ioactive.com/pdfs/ZeusSpyEyeBankingTrojanAnalysis.pdf
35. Pike, R.: The go programming language. Talk given at Google's Tech Talks (2009)
36. Piwigo: Piwigo is open source photo gallery software for the web. http://piwigo.org/
37. PrestaShop: Free ecommerce software. https://www.prestashop.com/
38. Santos, N., Raj, H., Saroiu, S., Wolman, A.: Using ARM TrustZone to build a trusted language runtime for mobile applications. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014, pp. 67–80. ACM, New York (2014). https://doi.org/10.1145/2541940.2541949
39. Sharp, R., Madhavapeddy, A., Want, R., Pering, T.: Enhancing web browsing security on public terminals using mobile composition. In: Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services, pp. 94–105. ACM (2008)

40. Sharp, R., Scott, J., Beresford, A.R.: Secure mobile computing via public terminals. In: Fishkin, K.P., Schiele, B., Nixon, P., Quigley, A. (eds.) Pervasive 2006. LNCS, vol. 3968, pp. 238–253. Springer, Heidelberg (2006). https://doi.org/10.1007/11748625_15
41. Stamm, S., Sterne, B., Markham, G.: Reining in the web with content security policy. In: Proceedings of the 19th International Conference on World Wide Web, pp. 921–930. ACM (2010)
42. Tong, T., Evans, D.: GuarDroid: a trusted path for password entry. In: Proceedings of Mobile Security Technologies (MoST) (2013)
43. Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., Vigna, G.: Cross site scripting prevention with dynamic data tainting and static analysis. In: NDSS, vol. 2007, p. 12 (2007)
44. Wang, X., Du, Y., Wang, C., Wang, Q., Fang, L.: WebEnclave: protect web secrets from browser extensions with software enclave. IEEE Trans. Dependable Secure Comput. **19**(5), 3055–3070 (2022). https://doi.org/10.1109/TDSC.2021.3081867
45. Whittaker, C., Ryner, B., Nazif, M.: Large-scale automatic classification of phishing pages. In: NDSS, vol. 10, p. 2010 (2010)
46. WordPress: Blog tool, publishing platform, and CMS. https://wordpress.org/
47. Wu, M., Miller, R.C., Little, G.: Web wallet: preventing phishing attacks by revealing user intentions. In: Symposium on Usable Privacy and Security (2006)
48. Xia, H., Brustoloni, J.C.: Hardening web browsers against man-in-the-middle and eavesdropping attacks. In: Proceedings of the 14th International Conference on World Wide Web, WWW 2005, pp. 489–498. ACM, New York (2005). https://doi.org/10.1145/1060745.1060817
49. Ye, Z.E., Smith, S.: Trusted paths for browsers. In: USENIX Security (2002)
50. Zencart: Putting the dream of your own business within reach of anyone! https://www.zen-cart.com/
51. Zhou, Y., Evans, D.: Protecting private web content from embedded scripts. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 60–79. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23822-2_4
52. Zhou, Z., Gligor, V.D., Newsome, J., McCune, J.M.: Building verifiable trusted path on commodity x86 computers. In: IEEE Symposium on Security and Privacy (2012)