SessionMagnifier: A Simple Approach to Secure and Convenient Kiosk Browsing

Chuan Yue
The College of William and Mary
Williamsburg, VA 23187, USA
cyue@cs.wm.edu

Haining Wang
The College of William and Mary
Williamsburg, VA 23187, USA
hnw@cs.wm.edu

ABSTRACT

Many people use public computers to browse the Web and perform important online activities. However, public computers are usually far less trustworthy than peoples' own computers because they are more vulnerable to various security attacks. In this paper, we propose SessionMagnifier, a simple approach to secure and convenient kiosk browsing. The key idea of SessionMagnifier is to enable an extended browser on a mobile device and a regular browser on a public computer to collaboratively support a Web session. This approach simply requires a SessionMagnifier browser extension to be installed on a trusted mobile device. A user can securely perform sensitive interactions on the mobile device and conveniently perform other browsing interactions on the public computer. We implemented SessionMagnifier for Mozilla's Fennec browser and evaluated it on a Nokia N810 Internet Tablet. Our evaluation and analysis demonstrate that SessionMagnifier is simple, secure, and usable.

Author Keywords

Web browsing, kiosk, mobile device, security, usability, Ajax.

ACM Classification Keywords

H.4.3 Information Systems Applications: Communications Applications—*Information browsers*; H.5.2 Information Interfaces and Presentation: User Interfaces—*User-centered design*; K.6.5 Management of Computing and Information Systems: Security and Protection—*Authentication, Invasive software, Unauthorized access*.

General Terms

Design, Experimentation, Human Factors, Security.

INTRODUCTION

Web browsing has become such an integral part of our everyday lives that we use browsers to perform many important tasks such as banking, shopping, and bill-paying. To facilitate ubiquitous Web access, many kiosk environments such as cafés, airport lounges, and hotel business centers provide people with Internet-connected public computers. These public computers often have high-speed network connections. They are also convenient to use since they normally have full-size keyboards and large displays. People who do not own a computer or carry a laptop with them frequently use these public computers to browse the Web.

Unfortunately, public computers are usually far less trustworthy than peoples' own computers. By "trustworthy", we mean that it is less likely that malware or spyware has been installed on a computer to log user input, steal account information, and even secretly hijack a secure (HTTPS) Web browsing session to make fraudulent transactions. Public computers are used by many people to run different applications and visit various websites; consequently, it is very likely for them to be infected with malware or spyware. Simply searching "public computer security" online, we can find numerous articles suggesting that people should not use public computers to perform sensitive activities. For example, Microsoft suggests that to be really safe, a user should not enter any sensitive information into a public computer [22].

To secure kiosk computing environments, researchers have proposed a number of solutions [3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 17]. Most of these solutions use a trusted mobile device such as a PDA (Personal Digital Assistant) or a mobile phone to enhance the security of kiosk computing environments, and we refer to them as *PDA-based* solutions. Mobile devices are favored by PDA-based solutions because (1) they are more portable than desktop and laptop computers, and (2) they are generally more trustworthy than public computers. Nevertheless, using small user interfaces on mobile handheld devices is inherently difficult [16].

Many of these PDA-based solutions focus on specific objectives such as securing application or data access [10, 15], securing user authentication or input [3, 7, 9, 11, 17], and verifying software integrity [5], so they cannot be easily adopted to secure an entire Web browsing session. Some solutions do have the objective of securing an entire kiosk browsing session [6, 8, 13, 14], but they suffer from a few drawbacks that limit their practical use.

In this paper, we propose SessionMagnifier, a simple approach to secure and convenient kiosk browsing. SessionMagnifier also relies on a trusted mobile device and is a PDA-based solution. However, our position is that with the support of a *trusted mobile device* (referred to as *PDA*), a

UbiComp 2009, Sep 30 – Oct 3, 2009, Orlando, Florida, USA.

solution to the problem of securing Web browsing on an *untrusted public computer* (referred to as *PC*) should, in essence, strive to synthesize the usability advantages of a PC and the security advantages of a PDA. Otherwise, a user can simply take the security risks of only using a PC, or a user can simply tolerate the inconvenience of only using a PDA with its small keyboard and display. Note that we use the term PDA to represent either a mobile phone or a PDA in this paper, and we expect people to eventually use our solution on mobile phones which are more popular than PDAs.

SessionMagnifier is designed as a browser extension, and the key idea is to enable an extended browser on a PDA and a regular browser on a PC to collaboratively support a Web session. After a user types in the address of a website and initiates a Web session from the PDA, the extended browser on the PDA accurately synchronizes a modified copy of its latest webpage document to a regular browser on the PC. The copied webpage document is modified to achieve accurate webpage rendering on the PC browser, to track a user's interaction with the same webpage on the PC browser, and to prevent sensitive information from leaking to the PC.

This solution is simple and practical because a user only needs to carry a trusted mobile device and install a SessionMagnifier extension to the device's Web browser – no third-party proxy is needed, no installation or configuration on an untrusted computer is needed, no Web server modification is needed, and no extra cryptographic key exchange is needed. SessionMagnifier provides a strong security guarantee because end-to-end security is directly established between a trusted mobile device and a remote Web server; meanwhile, simple and explicit communication interfaces are defined to enforce strong isolation between a PDA and a PC. SessionMagnifier enables a user to fully take advantage of the convenience of using a PC. This is because only very sensitive interactions such as entering username and password need to be directly performed from the PDA while all other browsing interactions can be conveniently performed from the PC.

We implemented SessionMagnifier for Mozilla's Fennec mobile browser [23]. We installed Fennec and SessionMagnifier on a Nokia N810 Internet Tablet and conducted evaluations on usability, performance, and feasibility. Our evaluation and analysis demonstrate that the proposed simple solution can be practically used to support secure and convenient Web browsing on untrusted public computers.

RELATED WORK

Balfanz and Felten [1] introduced a *splitting-trust paradigm* to divide an application between a small trusted mobile device and a bigger, more powerful, but possibly untrusted computer. SessionMagnifier is inspired by this paradigm; however, we do not split a browser but instead enable an extended browser on a trusted mobile device and a regular browser on an untrusted computer to collaboratively support a Web session. The splitting-trust paradigm has also inspired many other kiosk computing solutions that rely on a trusted mobile device. We classify these solutions into four categories based on their different objectives.

Securing Application or Data Access

Oprea et al. [10] proposed a *three-party* secure remote terminal architecture to enable users to access their sensitive home computing environment via a trusted mobile device and an untrusted terminal. This three-party architecture is based on a thin-client VNC (Virtual Network Computing) remote display system [12], in which a VNC server can update the framebuffer displayed on a VNC client. Sharp et al. [15] proposed a VNC-based *thin-client* architecture to support secure access to unmodified applications. This architecture is similar to the three-party architecture [10], but it provides additional mechanisms to obfuscate the content displayed on an untrusted display. These VNC-based secure application or data access solutions work at the framebuffer level with high overhead, so they cannot be naturally adopted to support smooth Web interactions. In addition, trusted VNC servers must be deployed in these solutions.

Securing User Authentication or Input

Parno et al. [11] built a *Phoolproof* phishing prevention system that uses a trusted mobile device to perform mutual authentication between a user and a website. Mannan and Oorschot [7] proposed the *MP-Auth* protocol, in which a trusted mobile device turns a long-term password into a one-time password via the public key of an intended server; therefore, a user's long-term password will not be revealed to phishing sites or untrusted computers. McCune et al. [9] proposed a *BitE* framework that leverages the features of TPM (Trusted Platform Module) to establish an encrypted input tunnel from a trusted mobile device to an application running on a TPM-equipped untrusted computer. Clarke et al. [3] and Wu et al. [17] designed protocols that rely on both a trusted third-party proxy and a trusted mobile device to secure authentication on untrusted computers. In addition, Florencio and Herley [4] proposed approaches to secure password input on untrusted computers without using mobile devices. All these solutions focus on securing user authentication or input, so they are not directly applicable for securing Web browsing sessions.

Verifying Software Integrity

Garriss et al. [5] built a system that uses a mobile device to establish trust in a kiosk computing environment. This system employs both a TPM module equipped on a kiosk computer and an integrity attestation server of the kiosk, and it focuses on verifying the identity and integrity of software loaded on a public computer before revealing sensitive information to the computer. However, our SessionMagnifier focuses on securing Web browsing sessions on potentially untrusted public computers.

Securing Web Browsing Sessions

A few kiosk computing solutions share the same objective with our SessionMagnifier: securing Web browsing sessions. Ross et al. [13] proposed a *composable secure proxy* architecture to provide secure multi-modal access to Web services from any device. A similar proxy-based architecture called *Delegate* [6] was proposed to enable users to access Web services from untrusted computers. In these solutions, essentially it is the browser on an untrusted computer that ac-

cesses remote Web servers; meanwhile, secure proxies perform content and control filtering functionalities. Two main obstacles impede the adoption of these proxy-based solutions. First, secure third-party proxies must be widely deployed, well managed, and fully trusted by users. Second, to secure Web browsing, a proxy must use very complicated and comprehensive rules to validate requests, remove sensitive content, maintain user information, and manage session information such as HTTP cookies.

Margolin et al. [8] introduced a *Guardian* framework that uses a PDA as a proxy for all interactions between an untrusted computer and remote Web servers. This framework eliminates the requirement of using secure third-party proxies by moving their content and control filtering functionalities to a PDA. However, since it is still the browser on the untrusted computer that accesses remote Web servers, this solution does not reduce the inherent complexity of proxy-based solutions. Our SessionMagnifier directly uses the Web browser on a trusted mobile device to access remote Web servers, so it provides strong security assurances and greatly reduces the complexity of content and control filtering.

Recently, Sharp et al. [14] proposed a *split-trust browsing* architecture to explore splitting trust at the HTML level for Web applications. However, this architecture has three drawbacks that limit its practical application. First, its critical component the RDC (Remote Device Communication) agent must be installed on an untrusted computer. Second, its end-to-end security between a trusted mobile device and a remote Web server depends on an extra authentication and key-exchange process coordinated by the RDC agent. Third, it assumes that either Web applications are explicitly written or secure HTML-rewriting proxies are used to support split-trust browsing. In contrast, our SessionMagnifier is much simpler and more practically applicable – nothing needs to be installed or configured on an untrusted computer, end-to-end security between a trusted mobile device and a remote Web server is ensured by existing HTTPS connections, no third-party proxy is needed, and no modification needs to be made to existing Web applications.

DESIGN

In this section, we first use a motivating example to illustrate the use of SessionMagnifier in a kiosk browsing environment. We then define the threat model and assumptions under which SessionMagnifier operates. Finally, we present the architecture design of SessionMagnifier.

A Motivating Example

Alice goes on a trip without carrying her laptop, but she wants to bid an item at eBay.com. During the last hours of the bidding, she needs to check the latest bidding status and make appropriate adjustments as necessary. Alice takes a PDA (or a mobile phone) with her, but she feels uncomfortable to continuously use the small keyboard and display of the PDA. She finds a public computer in an Internet café, but she has concerns about the security and privacy of using this public computer to log into her online accounts. Figure 1 illustrates such a kiosk browsing environment, and the problem here is how Alice can securely and conveniently sign into her online accounts and make transactions.

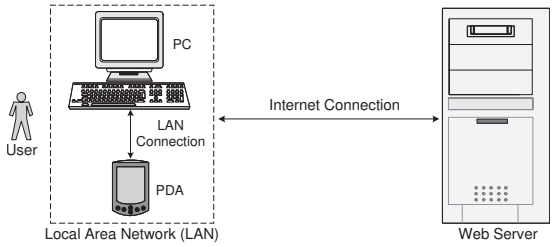

Figure 1. Kiosk browsing environment.

Fortunately, Alice can use a simple SessionMagnifier extension installed on the PDA browser to solve the problem. Alice first uses the PDA to connect to the Internet and signs into her eBay account from the PDA browser. The Internet connection is established either directly via Wi-Fi or indirectly via a USB-based or Bluetooth-based virtual network adapter of the public computer. Next, Alice turns on the SessionMagnifier extension installed on the PDA browser. She then types a URL address displayed by SessionMagnifier into the address bar of a regular PC browser and enables the connection between the PDA browser and the PC browser.

Starting from this point, SessionMagnifier synchronizes new webpage content from the PDA browser to the PC browser, and Alice can conveniently view and interact with the same webpage using the PC browser. Her interactions initiated from the PC browser will be sent back to SessionMagnifier and then securely sent out to eBay.com. Alice can verify and confirm any important interactions initiated from the PC browser, and she can also use an "Auto On" toolbar button to bypass this verification and confirmation step for less important interactions. Meanwhile, using a "Sync On" toolbar button, Alice can switch on or off the synchronization on each specific webpage so that she can input and view sensitive information only on the PDA browser. Alice may continue the bidding process until she wins or loses the auction.

Threat Model and Assumptions

In a kiosk browsing environment, attackers are interested in stealing a user's sensitive information such as username and password to commit identity theft. They are also interested in hijacking a user's browsing session to generate fraudulent transactions. More specifically, attackers may use the following five types of attacks to achieve their goals.

- **input stealing** – acquire sensitive input information by using software or hardware keyloggers.
- **output stealing** – acquire sensitive output information by using screen or window capture software.
- **session information stealing** – acquire sensitive session information such as HTTP cookies and session IDs through malware or spyware.
- **session hijacking** – (secretly) control a session and make fraudulent transactions through malware.
- **network attacks** – perform the above stealing and hijacking attacks at the network-level.

We define the capabilities of an attacker at two different levels: host-level and network-level. With host-level capabilities, an attacker can install malicious hardware and software on a public computer and perform the first four types of attacks listed above. With network-level capabilities, an attacker can eavesdrop or tamper with network messages to perform passive or active network attacks.

We consider three typical types of Web sessions: pure HTTPS sessions, pure HTTP sessions, and hybrid sessions. In a pure HTTPS session, a Web server uses SSL/TLS cryptographic protocols to protect all important interactions with an authenticated user, and a user can also authenticate the Web server by inspecting its certificate. In a pure HTTP session, a Web server does not provide any transport layer security protection. In a hybrid session, a Web server uses SSL/TLS cryptographic protocols to protect the user authentication process, but it uses both HTTP and HTTPS to serve an authenticated user. Pure HTTPS sessions are supported by high-security institutions such as banks and credit card companies. Hybrid sessions are used by service providers such as Yahoo Mail. Pure HTTP sessions are used by websites that provide less sensitive services. For pure HTTPS sessions, we grant an attacker both the host-level and network-level capabilities. For hybrid and pure HTTP sessions, we only grant an attacker the host-level capabilities.

Like previous studies, we assume that a user's mobile device is a priori secure. In our design, the simple and explicit communication interfaces between a PDA and a PC further protect the security of the PDA. Given the prevalence of phishing attacks, we also assume that a user is security conscious and is able to discern phishing through, for example, inspecting a Web server's certificate validated by the PDA browser.

Architecture Design

Figure 2 illustrates the high-level architecture of SessionMagnifier. A user simply installs the SessionMagnifier extension on a PDA browser; nothing needs to be installed or configured on a regular PC browser, and no third-party proxy is required. At the network layer, the PC can access the PDA via TCP connections. At the application layer, the regular PC browser communicates with the extended PDA browser using the HTTP protocol. A user directly uses the PDA browser to establish a Web session with a remote Web server. The SessionMagnifier extension is responsible for synchronizing the latest HTML webpage document from the PDA browser to the PC browser, and it is also responsible for accepting interactions initiated from the PC browser and securely performing these interactions on the PDA browser.

The simple architecture of SessionMagnifier leverages two important features of modern Web browsers: end-user extensibility [18, 19] and Ajax (Asynchronous JavaScript and XML) technology [20]. End-user extensibility allows the SessionMagnifier browser extension to maximize its capabilities and seamlessly integrate its functionalities with modern browsers. Ajax technology enables a regular PC browser to periodically send HTTP requests to SessionMagnifier and maintain the communication with the PDA browser. End-

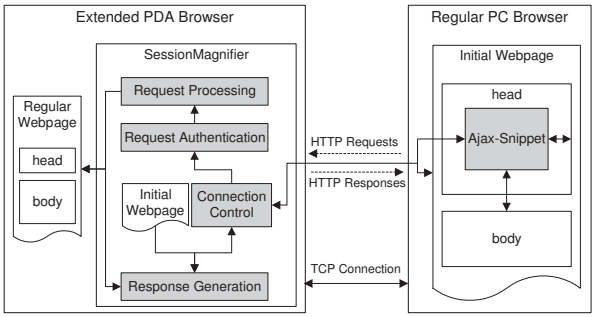

Figure 2. SessionMagnifier high-level architecture.

user extensibility is well supported by popular browsers such as Firefox [18] and Internet Explorer [19], and Ajax technology has received wide acceptance among all popular Web browsers [20]. Therefore, SessionMagnifier can be practically implemented and deployed on popular Web browsers.

In a kiosk browsing environment, establishing TCP connections between a PDA and a PC is feasible, and having Internet access for a PDA is also feasible. Using Wi-Fi, a user can easily establish both types of network connections. If Wi-Fi is not available, a user can use USB or Bluetooth to enable TCP connections between a PDA and a PC; meanwhile, using various *Internet access over USB* or *Internet access over Bluetooth* techniques (e.g., Microsoft ActiveSync), a user can also easily obtain Internet access for a PDA. Therefore, SessionMagnifier can be practically used in kiosks.

The SessionMagnifier extension consists of four main components: *connection control*, *request authentication*, *request processing*, and *response generation*. In addition, it also contains an *initial webpage*, which is an HTML file to be sent to a regular PC browser. We still use the previous motivating example to describe the roles played by the four components and the initial webpage in a kiosk browsing session.

Connection Control

When Alice turns on the SessionMagnifier extension installed on the PDA browser, the connection control component starts to work. This component uses a server socket to listen for new incoming connections from a PC. The server socket is TCP-based so that connections can be directly made from a regular PC browser. After the server socket binds to the IP address (e.g., 192.168.1.3, assigned by the kiosk LAN network) and a TCP port (e.g., 3000) of the PDA, the connection control component will display the URL address of SessionMagnifier (e.g., http://192.168.1.3:3000) to Alice.

Establishing the connection from a PC browser to the PDA browser is just like visiting a regular website. Alice simply types the URL address of SessionMagnifier into the address bar of the regular PC browser and sends out an initial HTTP request. When the connection control component of SessionMagnifier receives this initial HTTP request, it displays the source IP address of the request in a dialog box and asks Alice to confirm this connection. This confirmation dialog box is employed to help Alice make sure that the initial connection request does come from her PC.

If Alice accepts this initial connection request, the connection control component will read the initial webpage and send it to the PC browser. In an analogy, the initial webpage of SessionMagnifier is like the homepage of a regular website. The *body* of the initial webpage is very simple. It provides a simple form to ask Alice to submit a one-time password. The *head* of the initial page mainly contains Ajax-Snippet, which is a set of XHR (XMLHttpRequest) [20] related objects and functions. After the initial webpage is loaded on the PC browser, Ajax-Snippet will periodically send out "POST" type XHR polling requests to the connection control component of SessionMagnifier. Ajax-Snippet sends an XHR polling request and the connection control component returns a response; therefore, all further communication between the PC browser and the PDA browser can be automatically carried out.

If Ajax-Snippet receives a response message that contains a new webpage document, it will smoothly update the head and body of the initial webpage to keep the webpage content on the PC browser synchronized with that on the PDA browser. Meanwhile, Ajax-Snippet always resides in the head of the current webpage on the PC browser to maintain the communication with the PDA browser. Ajax-Snippet uses "POST" type XHR polling requests so that any interaction information such as link clicking or form filling on the PC browser can be directly piggybacked onto an XHR polling request and sent to the PDA browser.

Request Authentication

The one-time password mentioned above is generated and stored by SessionMagnifier on the PDA browser. On the PC browser, the same password submitted by Alice will not be transmitted to the PDA; it is just stored and used by Ajax-Snippet to compute the HMAC (keyed-Hash Message Authentication Code) for each XHR polling request. Before sending an XHR polling request, Ajax-Snippet computes an HMAC for the header and content of the request and appends the HMAC as an additional parameter of the request URI.

When the connection control component of SessionMagnifier receives an XHR polling request, it will forward the request to the request authentication component. The request authentication component will then compute a new HMAC for the received request (discarding the additional HMAC parameter) and compare the computed HMAC with the HMAC embedded in the request URI. If the two HMACs are identical, the XHR polling request is regarded as valid and is further forwarded to the request processing component. We use such a request authentication mechanism to protect the browsing session on the PDA and to ensure that SessionMagnifier only processes the requests sent from Alice's PC browser.

Request Processing

When the request processing component receives a valid XHR polling request, it will perform two tasks: new content checking and interaction information merging. The former is to check whether new webpage document on the PDA browser needs to be synchronized to the PC browser. The later is to check whether a user's interaction information on the PC browser needs to be merged to the PDA browser.

SessionMagnifier keeps a timestamp for the latest webpage document on the PDA browser. A timestamp used by SessionMagnifier is the number of milliseconds since midnight of January 1st, 1970. Whenever a new webpage document is synchronized to the PC browser, the timestamp of the document is also sent to Ajax-Snippet using the same response message. Whenever Ajax-Snippet sends an XHR polling request to SessionMagnifier, it carries back the timestamp of the current webpage document on the PC browser using the same request message.

The request processing component compares the two timestamp values to determine whether the webpage document on the PC browser is outdated. If the timestamp of the webpage document on the PC browser is older than that on the PDA browser, the request processing component informs the response generation component to synchronize the new webpage document on the PDA browser to the PC browser. Otherwise, it simply informs the response generation component to send back an empty response message to Ajax-Snippet.

The request processing component also examines the content of this "POST " type XHR polling request to see whether any interaction information is carried back from the PC browser. If new interaction information is carried in the XHR polling request, the request processing component will further execute the following four steps to merge the interaction information to the PDA browser. First, it will accurately reflect the interaction information (e.g., form filling information) to the corresponding webpage elements on the PDA browser. Second, it will highlight these webpage elements and scroll them into the view window of the PDA browser. Third, it will display a modal dialog box to ask Alice to verify the highlighted webpage elements. Finally, if Alice confirms that the interaction information reflected on the PDA browser is what she did on the PC browser, the request processing component will actually perform the interaction (e.g., submitting a form) on the PDA browser; otherwise, the request processing component will undo the changes made in the first two steps and ignore the interaction information carried in this XHR polling request. By only performing confirmed interaction information on the PDA browser, the request processing component assures a user that important interaction information is not tampered with or injected by attackers.

Response Generation

The response generation component is the most critical component of SessionMagnifier, and it poses three main design challenges: (1) how to enable high-quality webpage document synchronization from the PDA browser to the PC browser, (2) how to enable accurate user interaction on the PC browser, and (3) how to prevent sensitive information from leaking out of the PDA. When a new webpage document is loaded on the PDA browser, the response generation component uses the procedure shown in Figure 3 to generate a response message for the PC browser. This procedure consists of four

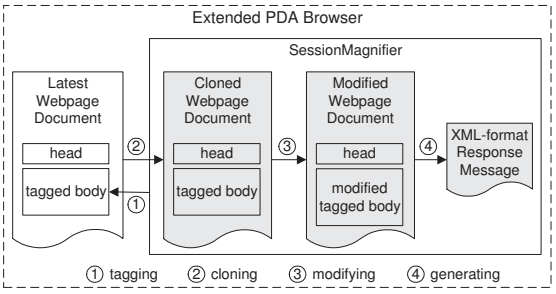

Figure 3. Response generation procedure.

main steps: *tagging*, *cloning*, *modifying*, and *generating*. We now detail these four steps to explain how we addressed the design challenges of the response generation component.

(1) Tagging: In the tagging step, the response generation component adds `id` attributes to the *interested actionable elements* of the latest webpage document on the PDA browser. We define interested actionable elements as the elements on which keyboard or mouse interactions will trigger the loading of a new webpage document. For example, links, forms, and clickable input elements outside of the forms are all regarded as interested actionable elements.

Tagging allows SessionMagnifier to simply use unique element identifiers to accurately track interested actionable elements on both the PC browser and the PDA browser. By directly tagging the webpage document on the PDA browser, the response generation component saves memory space and avoids the complexity of using any additional mapping mechanisms. Note that the response generation component only tags those interested actionable elements that do not have an `id` attribute, so it will not affect the behavior of the webpage document on the PDA browser.

(2) Cloning: In the cloning step, the response generation component uses the standard *cloneNode* DOM (Document Object Model) method to clone a complete copy of the above tagged webpage document. Using a cloned webpage document has two advantages. One is that standard DOM methods can still be handily used to modify the webpage. The other is that any further modification is only made to this cloned copy without polluting the webpage document on the PDA browser.

(3) Modifying: In the modifying step, the response generation component makes three main modifications to the cloned webpage document: URL address modification, event handler modification, and sensitive information filtering.

In general, each HTML webpage document has a set of associated supplementary objects such as stylesheets, images, and scripts. After loading a webpage document synchronized from the PDA browser, the PC browser must also download the associated supplementary objects in order to accurately render the webpage. To support the downloading of supplementary objects, the response generation component changes all the relative URL addresses contained in the cloned webpage document to the absolute URL addresses of the original Web servers. Without such a modification, the PC browser will send all relative URL requests to the PDA browser because it actually always connects to the SessionMagnifier browser extension on the PDA browser.

To track Alice's interaction with the same webpage on the PC browser, the response generation component must change the event handlers of those interested actionable elements. For form elements, the response generation component changes their *onsubmit* event handlers by adding a call to a specific JavaScript function residing in Ajax-Snippet. Therefore, later on when Alice submits a form on the PC browser, the `id` attribute value and element values of the form will be passed to Ajax-Snippet and then sent back to SessionMagnifier via an XHR polling request. In a similar way, the response generation component changes *onclick* event handlers of links and other clickable input elements outside of the forms to track click interactions performed on the PC browser.

Filtering sensitive information is much simpler in SessionMagnifier than in existing solutions [6, 8, 13, 14]. The main reason is that SessionMagnifier only synchronizes webpage documents to the PC browser, and no session control information such as HTTP cookies will be leaked to the PC. Therefore, the response generation component only needs to filter out sensitive information contained in a webpage document itself. To achieve this goal, the response generation component mainly uses the following two strategies.

One is to remove any possibly sensitive information that is useless to the interaction and display of the webpage on the PC browser. For example, webpages often contain sensitive information such as session IDs in their URL links and form *action* attributes. The response generation component simply sets all form *action* attribute values to empty, and it also sets all link *href* attribute values to empty. Note that tracking form submitting and link clicking is enabled by the above event handler modifications; therefore, the original *action* and *href* attribute values are useless to the interaction of a webpage on the PC browser.

The second strategy is to obfuscate personalized sensitive information. The basic idea is to replace user-specified sensitive information with information that is meaningless to attackers. For example, many websites display username information on their webpages for a logged-in user. Preventing the leakage of username information is important for protecting against attacks such as phishing and password guessing. SessionMagnifier maintains a rule table, in which simple filtering rules are defined by a user to specify which information should be obfuscated for each specific website. These rules could be defined for stable values (e.g., username), and they could also be defined for dynamic values (e.g., online banking balance) if the corresponding HTML elements of those values have stable IDs. The response generation component simply applies the rules to remove sensitive information contained in the cloned webpage document.

(4) Generating: In the last step, the response generation component extracts information from the modified webpage

document and generates an XML-format response message. The response message is in XML-format so that later Ajax-Snippet can accurately extract structured response information from the *responseXML* attribute of an XHR object. The modified webpage document is an HTML document, but an XHR object expects to receive a valid XML document. Since HTML webpages are often malformed, directly sending a modified webpage document to Ajax-Snippet will often result in parsing errors. Therefore, the response generation component will extract essential head and body information from a modified webpage document, encapsulate the extracted information in an XML-format response message, and finally send out the response message to Ajax-Snippet.

Initial Webpage

We mentioned that after the initial webpage is loaded on the PC browser, Ajax-Snippet will always keep itself within the head of the current webpage on the PC browser and periodically send out "POST" type XHR polling requests to communicate with SessionMagnifier.

Whenever Ajax-Snippet receives an XML-format response message that contains a new webpage document, it will first use the head information contained in the response message to replace the head of the current webpage on the PC browser. To support proper rendering on different browsers such as Internet Explorer and Firefox, Ajax-Snippet detects the type of the PC browser and performs this replacement for each top-level child of the head element. Ajax-Snippet will then use the body information contained in the response message to replace the body of the current webpage on the PC browser. By combining the structural advantages of using DOM methods and the simplicity advantages of using the *innerHTML* property of HTML elements, Ajax-Snippet can smoothly and accurately keep the webpage content on the PC browser synchronized with that on the PDA browser.

Meanwhile, whenever Alice interacts with an interested actionable element of the synchronized webpage on the PC browser, a call to a specific JavaScript function is made to extract the interaction information. The extracted information will be carried in the content of the next XHR polling request and synchronized to SessionMagnifier.

IMPLEMENTATION

The SessionMagnifier browser extension is designed to be implementable on different Web browsers. Indeed, only the connection control component is browser-specific; the request authentication, request processing, and response generation components and the initial webpage can all be implemented using standard JavaScript and HTML that are supported by modern Web browsers.

We implemented a full-fledged SessionMagnifier browser extension for Mozilla's Fennec browser [23], which is the mobile version of Firefox and is currently in alpha release. Similar to Firefox, Fennec provides full support for add-ons and rich Internet applications. Our SessionMagnifier extension for Fennec is purely written in JavaScript and HTML. Due to the space limit, we only briefly describe the implementation of the connection control component that is specific to Fennec, and delineate the webpage content and interaction synchronization capabilities of SessionMagnifier in our current implementation.

We implemented the connection control component of SessionMagnifier as a server socket object of Mozilla's nsIServerSocket interface. For this server socket object, we created a socket listener object of Mozilla's nsIServerSocketListener interface to asynchronously accept incoming TCP connections, and we also associated a data listener object of Mozilla's nsIStreamListener interface to a connected socket transport to asynchronously accept HTTP requests. In a Fennec browser extension, these objects can be easily created and manipulated using JavaScript code to realize the functionality of the connection control component.

For webpage content synchronization, SessionMagnifier supports various webpages including dynamic webpages (e.g., Google Maps) that use Ajax, CSS, or other DHTML techniques. SessionMagnifier detects dynamic webpage changes on a PDA browser and synchronizes the new content to a PC browser. In principle, any type of webpage content could be synchronized by SessionMagnifier. However, the current version of SessionMagnifier cannot properly synchronize some webpages such as Gmail webpages due to the unfinished implementation on iframe elements. For interaction synchronization, our current implementation supports interactions on those *interested actionable elements* as defined in our design. However, SessionMagnifier can also easily synchronize any other interactions (e.g., those altering document elements without calling for new data) as long as their corresponding HTML elements support event handlers (which could be *onsubmit*, *onclick*, or any other handlers). We will provide support for other necessary interactions in our future implementation.

SECURITY ANALYSIS

The security assurances provided by SessionMagnifier can be attributed to three factors: using a trusted PDA, accessing a remote Web server directly from a PDA browser, and enforcing strong isolation between a PDA and a PC. We now analyze the security of SessionMagnifier based on the threat model and assumptions defined in our design. We must emphasize that SessionMagnifier aims to enhance the security of using an untrusted public computer for Web browsing, but it does not attempt to secure a kiosk environment itself. In other words, the security upper-bound of using SessionMagnifier is equivalent to that of using a user's own laptop computer in a kiosk environment.

Using a trusted PDA, SessionMagnifier is robust against input stealing attacks and output stealing attacks. A user simply enters sensitive information such as username and password on the PDA browser, so the keyloggers installed on the PC cannot acquire sensitive input information. Meanwhile, user-specified information is obfuscated by the response generation component of SessionMagnifier, so it is very hard for screen or window capture software installed on the PC to acquire sensitive output information. The ability

of SessionMagnifier to provide these two types of security assurances is the same as that of other solutions to securing kiosk browsing sessions [6, 8, 13, 14].

Accessing a remote Web server directly from the PDA browser is a unique feature of SessionMagnifier because other solutions [6, 8, 13, 14] all use the browser on an untrusted computer to establish a Web session with a remote Web server. Besides, SessionMagnifier enforces a strong isolation between a PDA and a PC by only allowing HTTP communications. Combining these two factors, SessionMagnifier provides high security assurances to protect against other three types of attacks: session information stealing attacks, session hijacking attacks, and network attacks.

SessionMagnifier is robust against session information stealing attacks. Since SessionMagnifier only synchronizes the content of a modified HTML document from the PDA browser to the PC browser, session information such as HTTP cookies will never be leaked to the PC. Meanwhile, since all useless values such as form *action* attribute values and link *href* attribute values are simply set to empty by the response generation component of SessionMagnifier, no session IDs contained in these values will be revealed to the PC. Preventing the leakage of HTTP cookies and session IDs is important because an attacker can use them to further steal other sensitive user data or hijack browsing sessions. Unfortunately, this type of security assurance is not considered in [14], and it is considered in [6, 8, 13] by employing very complex filtering rules and mapping mechanisms.

SessionMagnifier is robust against session hijacking attacks. Since a Web session is established between the PDA browser and a remote Web server, malware installed on the PC cannot directly seize the control of a session to make fraudulent transactions. The only possible way for an attacker to hijack a session is to tamper with or inject interaction information in an XHR polling request. SessionMagnifier defends against such attacks by using its request processing component to accurately reflect interaction information on the PDA browser and only perform user-confirmed interactions. Protection against session hijacking is also considered in [6, 8, 13, 14]. However, because these solutions use the PC browser to establish a Web session with a remote Web server, they must perform very complex request validations but still cannot achieve the same security level as that of SessionMagnifier.

In terms of network attacks, SessionMagnifier ensures end-to-end security between a trusted mobile device and a remote Web server by directly using existing HTTPS connections. For pure HTTPS sessions, SessionMagnifier is robust against network attacks. For hybrid sessions, SessionMagnifier is robust against network attacks for HTTPS webpages. For pure HTTP sessions, there is no strong incentive to defend against network attacks because in general Web servers that do not use SSL/TLS only provide less sensitive services. Existing solutions [6, 8, 13, 14] provide similar security guarantees against network attacks as provided by SessionMagnifier; however, they often necessitate an additional SSL/TLS connection or encryption channel still mainly because they use the PC browser to establish a Web session with a remote Web server.

EVALUATION

In this section, we focus on evaluating the usability of SessionMagnifier. We also briefly present the performance and feasibility evaluation results.

Usability Evaluation

Our primary goal is to measure whether using SessionMagnifier is more convenient than merely using a PDA browser. To achieve this goal, we conducted a usability study based on a real eBay bidding scenario.

Participants

Twenty-two adults, 11 females and 11 males, participated in our user study. They were voluntary students and faculty members recruited from eight degree programs of two universities. Nineteen participants were between ages of 18 and 30, and three participants were over 30 years old. We did not screen participants based on experiences using different Web browsers, using mobile devices, or using eBay services.

Scenario and Procedure

We presented such a scenario to each participant: "Suppose you want to bid a book titled *'Xbox 360 games in a nutshell'* at eBay.com. You visit www.ebay.com and sign into an eBay testing account. You search the book using its title and find the item. You place a higher bid by adding one dollar and get confirmation that you are currently the highest bidder. Finally, you sign out of eBay.". Note that the book item was added to eBay using a seller's account created by us, and the eBay testing account was also created by us.

We asked each participant to perform this eBay bidding scenario using two procedures A and B. In procedure A, a participant only uses a PDA; in procedure B, a participant uses both a PDA and a PC. We used a Nokia N810 Internet Tablet as the PDA, and we pre-installed a Fennec browser [23] and our SessionMagnifier browser extension on it. In procedure A, each participant used the Fennec browser (with SessionMagnifier turned off) on the PDA to perform the bidding scenario. In procedure B, each participant used the Fennec browser (with SessionMagnifier turned on) on the PDA and a regular browser on a PC to perform the bidding scenario. We randomly assigned 11 participants to first perform procedure A and the other 11 participants to first perform procedure B. Before a test, we trained each participant on the use of the PDA and the Fennec browser. We also explained the purpose of SessionMagnifier, and it seems that all the participants understood the threats addressed by SessionMagnifier.

We presented the tasks of the two procedures to each participant. Procedure A has 10 tasks and procedure B has 18 tasks. Each task is a specific browsing action such as clicking on the "Sign in" link or typing "Xbox 360 games in a nutshell" into the "Find" input field. Procedure B has more tasks because we asked each participant to verify and confirm all the interaction information sent back to the PDA browser. For

The six questions common to both procedures A and B (replacing the 'X' in the questions with 'A' or 'B')
Q1: Typing into input fields of a webpage in procedure X is easy
Q2: Clicking links of a webpage in procedure X is easy
Q3: Clicking buttons of a webpage in procedure X is easy
Q4: Scrolling a webpage in procedure X is easy
Q5: Viewing webpage content in procedure X is easy
Q6: Overall, performing procedure X is easy
The four questions specific to procedure B
QB1: Typing the URL address http://192.168.1.3:3000 of SessionMagnifier into the address bar of a PC browser in procedure B is easy
QB2: Clicking the "Sync On" toolbar button in procedure B is easy
QB3: Verifying a highlighted element (form, link, button) in procedure B is easy
QB4: Confirming an action (form, link, button) using the dialog box in procedure B is easy

Table 1. The 16 close-ended questions.

example, after the search (on "Xbox 360 games in a nutshell") performed on the PC browser is reflected on Fennec, SessionMagnifier highlights the border of the search form with red color and displays a modal dialog box on the PDA. Using this dialog box, a participant can either confirm this form submission by clicking on the "OK" button or ignore this form submission by clicking on the "Cancel" button.

Data Collection

We collected data through observation and questionnaire. When a participant was performing the two procedures, we observed the progress of the tasks. After a participant finished the two procedures, we asked the participant to answer a five-point Likert-scale (Strongly disagree, Disagree, Neither agree nor disagree, Agree, Strongly Agree) [21] questionnaire. The questionnaire consists of 16 close-ended questions as listed in Table 1 (Q1 to Q6 were asked for both procedures). We also asked participants to write down open-ended comments on using SessionMagnifier.

Results and Analysis

We observed that all the 22 participants successfully finished the two procedures. We converted the responses to the Likert-scale questionnaire to numeric values (1=Strongly disagree, 2=Disagree, 3=Neither agree nor disagree, 4=Agree, 5=Strongly Agree) and compared the responses to procedures A and B using t-tests. Strictly speaking, since the responses are ordinal data, they do not necessarily have interval scales. However, in practice this type of analysis is acceptable [2].

Figure 4 illustrates the mean ratings to questions Q1 to Q6 for the two procedures. We can see that for all the six questions the mean ratings to procedure B are much higher than those to procedure A. The t-tests (with 95% confidence interval) further reveal that the mean rating differences between the two procedures are significant for each of the six questions. These results clearly indicate that SessionMagnifier enables users to exploit the usability advantages of using the large keyboard and display of a PC.

Using a similar method, we analyzed the responses to the four questions specific to procedure B. The mean ratings to

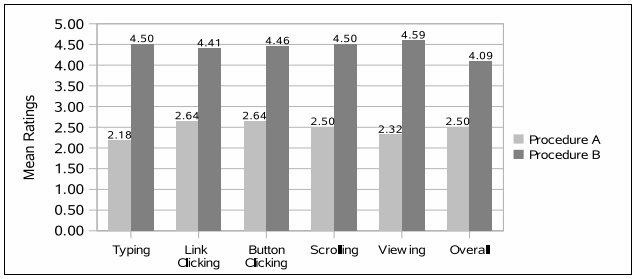

Figure 4. Mean ratings to questions Q1 to Q6.

questions QB1 to QB4 are: 3.96, 4.09, 3.64, and 3.86, respectively. One-sample t-test (with 95% confidence interval) against the test value of three shows that the mean ratings to these questions are higher than three with statistical significance. These results indicate that performing the specific interactions introduced in procedure B is not difficult to users.

We further analyzed participants' open-ended comments on using SessionMagnifier. We found that 14 participants clearly mentioned that typing into input fields and viewing webpages are very convenient in procedure B. We also found that nine participants mentioned that it would be better if the number of verifying and confirming steps could be reduced. We should note that verification and confirmation are necessary steps for important interactions, and they were or should be considered in other splitting-trust based solutions. Moreover, SessionMagnifier allows a user to bypass this verification and confirmation step using the "Auto On" toolbar button. In procedure B, we disabled the "Auto On" feature to measure the worst case usability, but we believe that a user can actually be trained to use this feature to confidently bypass less important interactions.

Performance and Feasibility Evaluation

In our performance evaluation, we mainly measured the speed of SessionMagnifier in response generation (i.e., the procedure illustrated in Figure 3) and response transmission. We used Fennec to visit five homepages. The page size and response generation time (average of five runs) of these homepages are listed in Table 2. We can see that the larger and more complex the HTML page is, the more generation time is needed. Response generation is not very efficient for large webpages, but we believe that the main reason is the poor memory management in the current alpha release of Fennec. Using the Linux `top` command, we observed that even without SessionMagnifier, Fennec requires over 107% of memory (128MB RAM) on Nokia N810 Internet Tablet by just loading the google.com homepage; however, the built-in browser on Nokia N810 only requires less than 78% of memory even when loading the amazon.com homepage. We believe that increasing memory or an improved Fennec can help reduce the response generation time of SessionMagnifier. In terms of the response transmission speed, since the PDA and the PC are located in the same LAN, the generated response message can normally be transmitted from the PDA to the PC within a second.

Site Name	Page Size (KB)	Generation Time (second)
google.com	8.9	0.36
ebay.com	49.5	1.37
bestbuy.com	80.2	2.64
weather.com	148.7	2.71
amazon.com	201.1	3.03

Table 2. Page size and response generation time of five homepages.

In our feasibility evaluation, we mainly tested whether TCP connections between a PDA and a PC can be established via Wi-Fi in kiosk environments. We conducted experiments at 20 public places (seven hotels, seven restaurants, three libraries, two gyms, and one coffee shop) that offer free Wi-Fi Internet access. Since some places do not provide public computers, we used a laptop to act as a public PC. At each place, we did not do any special configuration on either the PDA (the Nokia N810 Internet Tablet) or the PC, but just connected them to the same Wi-Fi access point to acquire IP addresses. We observed that TCP connections between the PDA and the PC are blocked (perhaps due to strict security restrictions) at three hotels and two restaurants. At the other fifteen places, the PC can connect to the PDA using a TCP port (e.g. 3000), and we successfully performed Web browsing using SessionMagnifier. These results indicate that it is practical to use SessionMagnifier at many free Wi-Fi hotspots. Meanwhile, as mentioned in the design section, a kiosk environment that plans to enable SessionMagnifier can also use USB or Bluetooth, in addition to Wi-Fi.

CONCLUSION

We presented SessionMagnifier, a simple approach to secure and convenient kiosk browsing. SessionMagnifier strives to synthesize the usability advantages of a public computer and the security advantages of a mobile device. Since a Web session is directly established between the PDA browser and a remote Web server, SessionMagnifier provides a strong end-to-end security guarantee and greatly reduces the complexity of content and control filtering. Since a user can perform the majority of browsing interactions from the PC and only perform very sensitive interactions from the PDA, SessionMagnifier enables a user to fully take advantage of the convenience of using a PC. We presented the design of SessionMagnifier in detail and analyzed the security of SessionMagnifier using a rigorous threat model. We implemented SessionMagnifier for Mozilla's Fennec browser and evaluated its usability, performance, and feasibility. Our evaluation and analysis demonstrate that SessionMagnifier is simple, secure, and usable.

In future work, we will enhance the implementation and evaluation of SessionMagnifier. In particular, we will improve our usability evaluation, for example, by gathering information about participants' experience with mobile devices, by allowing participants to use our tool with only basic instead of step-by-step instructions, by collecting data illustrating participants' thoughts on security aspects of using our tool, and by incorporating some security attack scenarios.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their valuable suggestions and Tim Kindberg (our shepherd) for his great help in improving this paper. This work was partially supported by NSF grants CNS-0627339 and CNS-0627340, and a *2008 Arts & Sciences Graduate Research Grant* awarded to the first author by the College of William and Mary.

REFERENCES

1. D. Balfanz and E. W. Felten. Hand-held computers can be better smart cards. In *Proc. of the USENIX Security Symposium*, 1999.
2. S. Chiasson, P. van Oorschot, and R. Biddle. A usability study and critique of two password managers. In *Proc. of the USENIX Security Symposium*, 2006.
3. D. E. Clarke, B. Gassend, T. Kotwal, M. Burnside, M. van Dijk, S. Devadas, and R. L. Rivest. The untrusted computer problem and camera-based authentication. In *Proc. of the Pervasive Computing*, 2002.
4. D. Florencio and C. Herley. Klassp: Entering passwords on a spyware infected machine using a shared-secret proxy. In *Proc. of the ACSAC*, 2006.
5. S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Trustworthy and personalized computing on public kiosks. In *Proc. of the MobiSys*, 2008.
6. R. C. Jammalamadaka, T. W. van der Horst, S. Mehrotra, K. E. Seamons, and N. Venkasubramanian. Delegate: A proxy based architecture for secure website access from an untrusted machine. In *Proc. of the ACSAC*, 2006.
7. M. Mannan and P. C. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *Proc. of the Financial Cryptography*, 2007.
8. N. B. Margolin, M. Wright, and B. N. Levine. Guardian: A framework for privacy control in untrusted environments. Technical Report, University of Massachusetts, Amherst, 2004.
9. J. M. McCune, A. Perrig, and M. K. Reiter. Bump in the ether: a framework for securing sensitive user input. In *Proc. of the USENIX Annual Technical Conference*, 2006.
10. A. Oprea, D. Balfanz, G. Durfee, and D. K. Smetters. Securing a remote terminal application with a mobile trusted device. In *Proc. of the ACSAC*, 2004.
11. B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. In *Proc. of the Financial Cryptography*, 2006.
12. T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.
13. S. J. Ross, J. L. Hill, M. Y. Chen, A. D. Joseph, D. E. Culler, and E. A. Brewer. A composable framework for secure multi-modal access to internet services from post-pc devices. *Mob. Netw. Appl.*, 7(5):389–406, 2002.
14. R. Sharp, A. Madhavapeddy, R. Want, and T. Pering. Enhancing web browsing security on public terminals using mobile composition. In *Proceeding of the MobiSys*, 2008.
15. R. Sharp, J. Scott, and A. R. Beresford. Secure mobile computing via public terminals. In *Proc. of the Pervasive Computing*, 2006.
16. R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light. The personal server: Changing the way we think about ubiquitous computing. In *Proc. of the Ubicomp*, 2002.
17. M. Wu, S. Garfinkel, and R. Miller. Secure web authentication with mobile phones. In *Proc. of the DIMACS Workshop on Usable Privacy and Security Software*, 2004.
18. https://developer.mozilla.org/en/Extensions.
19. http://msdn.microsoft.com/en-us/library/aa753587(VS.85).aspx.
20. http://en.wikipedia.org/wiki/Ajax_(programming).
21. http://en.wikipedia.org/wiki/Likert_scale.
22. 5 safety tips for using a public computer. http://www.microsoft.com/protect/yourself/mobile/publicpc.mspx.
23. Fennec. https://wiki.mozilla.org/Fennec.