2010-12-13

Trusted Mobile Overlays

Robert Scott Robertson
Brigham Young University - Provo

Trusted Mobile Overlays

R. Scott Robertson

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Kent E. Seamons, Chair
Dan R. Olsen Jr.
Daniel M. A. Zappala

Department of Computer Science

Brigham Young University

April 2011

ABSTRACT

Trusted Mobile Overlays

R. Scott Robertson

Department of Computer Science

Master of Science

Sensitive information is increasingly moving online and as data moves further from the control of its owner, there are increased opportunities for it to fall into malicious hands. The Web is comprised of three untrusted components where there is a risk of information compromise: networks, service providers, and clients. This thesis presents Trusted Mobile Overlays: a system that leverages trusted mobile devices to protect users from these untrusted components of the Web, while minimizing deployment difficulties. It presents a high-level design of the system as well as a prototype that implements the design.

Keywords: security, public terminal, mobile phone

ACKNOWLEDGMENTS

Janelle's love, patience, and help inspired me throughout the entire thesis process. Alan provides a wonderful example of cheerfulness and hope despite difficulties. Dr. Seamons's thoughtful advice and reviews greatly contributed to the concepts on which the thesis is built, as well as to this document itself. Tim and the Internet Security Research lab provided the foundation and initial ideas on which to build. Brigham Young University - made possible by its owners and donors - made the ideal learning environment.

Thank you, all.

BRIGHAM YOUNG UNIVERSITY

SIGNATURE PAGE

of a thesis submitted by

R. Scott Robertson

The thesis of R. Scott Robertson is acceptable in its final form including (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory and ready for submission.

Date	Kent E. Seamons, Chair
Date	Dan R. Olsen Jr.
Date	Daniel M. A. Zappala
Date	Kent E. Seamons, Graduate Coordinator
Date	Thomas W. Sederberg, Associate Dean, College of Physical and Mathematical Sciences

Contents

Chapter 1

Introduction

We live in a world where sensitive information is increasingly available on the Web. From bank account details to social networking messages and medical information, there is a significant amount of sensitive data accessible online. This trend will likely continue as fast Internet access becomes more ubiquitous and web applications improve; people will increasingly rely on the Internet for personal information.

1.1 The Problem

The World Wide Web is inherently insecure. The web consists of three basic components - networks, service providers, and clients - that each require a different approach for protecting personal information

1.1.1 Untrusted Network

Information flowing between computers can be intercepted and even modified by those who have access to the network between the sending and receiving computers. Examples of attackers with sufficient access to read or change a network message include: a rogue employee at an Internet service provider, a patron on a library wired local area network, or a traveler in the vicinity of an airport wireless hotspot. In the context of the Internet, a user is rarely in control of all parts of the network.

A reliable solution to this problem has been available for some time. TLS (Transport Layer Security) allows network traffic to be encrypted to prevent eavesdropping and so that

attempts to modify the data will be detected by the receiving computer.

1.1.2 Untrusted Service Provider

In addition to network traffic being encrypted, people need to be able to trust the entity serving the information. Email providers, social networking sites, and web feed aggregators pass along information that people may consider sensitive. Service providers are entrusted with massive amounts of personal information, but privacy practices, especially of the largest social networking provider, Facebook, are often criticized [1]. In addition, even service providers with the best of intentions may fall victim to attacks from outside parties, thereby revealing sensitive user information. The Winter 2010 attacks on Google and other companies is a sobering recent example[2].

Various end-to-end encryption solutions help alleviate threats posed by untrusted service providers. Encrypted email systems like S/MIME and PGP, for example, allow email messages to be kept private from both malicious email providers and attackers.

1.1.3 Untrusted Client

Even with sensitive information safe from untrusted networks and service providers, there is still a vulnerable component in the web infrastructure: the client. Web clients are a combination of the web browsing software and computer on which users experience the web. Clients may be untrusted for several reasons. First, they may be public terminals, such as those found at Internet cafés, airports, hotels, schools, and libraries. As more information moves online these terminals provide a convenient way for people to access and update pertinent sensitive information wherever they are. Because these machines are out of the user's control, however, they cannot be assured of the practices used to secure them. Second, due to malware, even a user's own desktop or laptop computer may be in another's control.

Keystroke loggers and screen grabbers may allow whoever is in control of the client

[1] See `http://www.eff.org/deeplinks/2009/12/facebooks-new-privacy-changes-good-bad-and-ugly`

[2] See `http://www.wired.com/threatlevel/2010/01/operation-aurora/`, for one article.

to monitor anything that is input to or output from the client, including web interactions. This means that TLS is not enough to protect sensitive information from an untrusted client because the client has access to the unencrypted data. It also means that end-to-end encryption is not enough because the untrusted client has access to the contents of the overlays, which are displayed on the client's screen.

This thesis presents a solution to the problem of sensitive information on untrusted clients.

1.2 Motivating Scenario

Bob, a college student, has a few hours between classes. He goes to the university library, where he intends to spend the intermediate time using a library computer to do some homework and browse the Internet. He could use his smartphone, but enjoys the larger screen afforded by the library computer, finds it awkward to use his phone for long Internet sessions, and is concerned about his data usage for the month. A few minutes into his session, Bob decides to visit Facebook, and finds he has a private message from his dad:

> Bob, we'd love you to join in on our family gym membership. Please feel free to call and sign up whenever you get a chance. I think you may need the last four of my SSN to show you have permission to join. Your mom thinks we should cover your first year; you'll need my credit card number. Here's the info ...

When Bob sees the contents of the message, he wishes he hadn't viewed it on the possibly malware-infected library computer. He's also a little nervous about the social networking provider having access to this information.

A few minutes later, Bob is reading some news articles. He sees something interesting about a stock in his portfolio, so he decides to log into his broker's website. Most of what he finds there, like news articles, advertisements, and index prices are not sensitive. Some information, however, like his account numbers and the number of shares he owns *is* sensitive.

Bob doesn't need to see all of the sensitive information right now, but is curious about the sum total of his investments. He wishes he could somehow safely view just parts of the web page, but, again, all the information is now displayed on the public computer's screen.

Bob could come across several sensitive pieces of information throughout his browsing session. This data may be found on shopping, medical, financial, social media, or other types of websites. If Bob were using his trusted home computer, the data could be decrypted and displayed on his screen, while still protecting it from untrusted service providers. Because he is at the library, though, this is not an option.

It is true that Bob could use his mobile phone directly to browse the Web and avoid the library computer altogether. This is undesirable, though, because he intends to use the Internet for a long period of time, which may be cumbersome on his mobile phone. In addition, while it may be possible for him to turn to his mobile phone's web browser to visit only the sites he thinks might contain sensitive information, he can't predict everywhere he may come across secure content, and would be required to use his phone more frequently than necessary. He needs a solution where he can still enjoy the benefits of a large computer and free Internet for most of his browsing session, while taking advantage of his phone to read and compose just the secure content that he chooses.

1.3 Trusted Overlays

Trusted Overlays[19], developed by the Internet Security Research Lab at BYU, is a system that allows users to protect information from untrusted service providers. It is possible, for example, to view the contents of an email in a traditional web mail page, without the web mail provider being able to learn the plaintext.

While the original Trusted Overlays system protects users from untrusted service providers, it is not intended to keep them safe from untrusted clients. Because unencrypted text is shown on the client's screen, and input is received through the untrusted hardware and software, the system is vulnerable to malicious clients. Trusted *Mobile* Overlays extends

the idea of Trusted Overlays to provide protection from untrusted clients by moving the "trusted overlay" from the browser to a trusted mobile device.

1.3.1 Trusted Mobile Devices

Several researchers have suggested the use of trusted mobile devices, like smart cards, phones, or PDAs, to secure transactions on public terminals. Several properties of personal devices suggest that they are more secure than traditional computers, and especially than public terminals. First, people have physical control of their personal devices. Next, mobile phones and PDAs haven't been around as long as PCs, indicating that the methods of attack may not be as refined. Third, there is greater diversity among currently used mobile platforms than among untrusted clients (which very likely run a version of Windows on an Intel-based processor). This means specific attacks would likely need to be modified to be effective on the various mobile platforms. Lastly, mobile devices benefit from a smaller attack surface: they don't have to support as many services as traditional PC operating systems.

Since mobile devices are more likely to be secure than a public terminal, and they increasingly have their own Internet access, why not just use them in public places, rather than using public terminals at all? While it is possible to perform most common computing tasks on modern smartphones, PCs provide much larger screens, more comfortable keyboards, greater computing power, and possibly a faster and cheaper Internet connection (mobile providers may charge extra when their customers exceed a data usage threshold). For these reasons, smartphone owners may wish to use public computers for lengthy browsing sessions: the challenge is how to combine the convenience of public terminals with the security of mobile phones.

Chapter 2

Related Work

2.1 Mobile Devices as Trusted Platform

Many researchers have realized the advantages afforded by small, easily-carried computing devices. Nearly two decades ago Abadi et al. [1] proposed enhanced smart cards which could reduce the amount of information users reveal to untrusted clients. The proposed smart card would contain enough computing power to perform cryptographic operations. It would also have its own keyboard, so that secrets could be entered directly into the card, without revealing them to the untrusted computer.

Balfanz and Felten proposed that small computing devices can achieve the goals of smart cards [2]. They presented an implementation of PKCS #11 on a Palm Pilot, which allowed signing messages and reading encrypted email while using an untrusted machine, without revealing the private key. Users could choose whether to display decrypted messages on the untrusted terminal, or only on the Palm Pilot screen. They proposed a paradigm called "Splitting Trust", wherein applications are designed in such a way that sensitive operations can occur on a PDA, but computing-intensive operations happen on the PC. They showed that PDAs can fulfill the role of an I/O-enhanced smart card.

The preceding research demonstrates that trusted mobile devices provide a solution for improving user's security while using untrusted clients.

2.2 Authentication

The method presented by Clark et al. [6] seeks to ensure that what a user sees on an untrusted terminal is what was actually sent by a remote party. In this system a proxy server calculates a Message Authentication Code (MAC), using a key shared between the mobile device and the proxy, and encodes it, along with a one-time password and other information, in the form of an image. This image is sent to the untrusted computer. The mobile device's camera captures the screen including the encoded MAC and other metadata, which appear to the user as a bar at the bottom of the screen. The device then recreates the MAC for the captured screen, and compares it to the MAC extracted from the pixels in the image. If they match, the untrusted computer is considered faithful, and the user confirms this by sending the proxy the one-time password, which was also encoded in the image. Another scheme is presented that utilizes optical character recognition; it has similar goals to the encoded-image-based scheme.

Starnberger et al. [18] propose a system that solves a similar problem. It, too, leverages a mobile device's camera to confirm that a user sees what is expected on a public terminal. It uses barcodes rather than a custom pixel-mapping or OCR scheme. This approach requires less camera resolution and processing power from the mobile device, making an implementation more practical.

Other systems focus on authenticating mobile devices to each other. McCune et al. [13], for example, suggest using barcodes and mobile phones to provide mutual authentication.

2.3 Keeping Content Private

Research by Mannan and van Oorschot [11] seeks to protect long-term secrets by proposing a new protocol, MP-Auth. This protocol sends long-term secrets (e.g. a bank account number or password) over an untrusted channel, protecting the secrets with a service provider's public

key. Rather than typing the secrets directly into the untrusted terminal, they are stored or typed into a trusted device. The system also supports transaction verification. The system suggests a direct connection between the personal device and the untrusted terminal, which may not always be available. Another challenge is that the mobile device needs to have the service provider's public keys. Some suggestions for getting the keys are presented in the paper, but reliable public key distribution can be a major hurdle in a system's deployment.

Another system, described by Wu et al. [24], also uses mobile phones to help users safely authenticate without the two primary weaknesses of the previous paper. Namely, it allows a user to authenticate to remote websites, without the requirement of maintaining public keys or a connection between the untrusted host and the trusted mobile device. Another strength of the system is that it doesn't require changes to service providers. It works by having the user communicate through a security proxy that they control. The user types their username into the untrusted host, which sends it to their trusted security proxy server. The proxy server then sends a text message to the user's trusted mobile device, containing a link to an automatically generated URL on the proxy server. The user browses to the link on their mobile device and can confirm or reject the authentication request. If they confirm the connection, the secure proxy will act as a web proxy for the untrusted terminal, with the added benefit that cookies will be stored on the proxy server, so that they are not accessible to the untrusted machine. While this system avoids some problems, the added requirement of a trusted proxy presents a new challenge. Delegate [10] also uses a proxy to protect users' information, and addresses the problem of session hijacking. If the system determines that a web request is potentially dangerous it allows the user to confirm or deny the transaction.

Researchers have also proposed methods for protecting an entire session, rather than only credentials, on an untrusted computer. Shart et al. [16] describe a system where applications run on a thin server, and their output is echoed to both an untrusted computer and a trusted mobile device. This allows sensitive information to be displayed clearly on the

mobile device, but obscured on the untrusted computer. The system allows the user to take advantage of the comfortable input devices afforded by the untrusted machine by forwarding keyboard and mouse events to the thin-client server. Sensitive information, though, can be entered directly on the mobile device. While this would allow users to interact securely with web content while using a public terminal, the system requires an application server in the user's control, which they suggest could run directly from the mobile phone. Sharp et al. [17] present a similar idea, but applied specifically to web content. This system allows ciphertext to be embedded in HTML, and marked up in such a way that the system knows to transfer it to the phone where it can be safely read. The system also allows the secure creation of secure content. The system recommends a direct connection to the PC (like WiFi, USB or Bluetooth) and uses a browser extension on the untrusted computer, both of which may make deployment difficult.

"SessionMagnifier," presented by Yue and Wang, [25] seeks to protect an entire web session. The system includes a web server running on a trusted mobile device. The user connects the device to the same LAN as the untrusted PC. They then type the device's IP address into the untrusted PC's browser. Once the connection is established, the mobile device acts as a proxy for the untrusted device: preventing sensitive information from reaching the untrusted terminal and allowing actions originating at the untrusted terminal to be confirmed. The web content is displayed on both the trusted and untrusted screens, allowing credentials to be entered and sensitive information to be read via the mobile device.

Another approach, described by Parno et al. [14], is to use a trusted mobile device as a private key store. This system allows a user to authenticate to a remote service by leveraging the infrequently used client-side authentication provided by SSL/TLS. The mobile device is connected to the untrusted terminal. The browser on the terminal can then use the private key for TLS on the device to prove the users identity to the remote service. This allows a user to authenticate without requiring them to enter a username and password, thereby protecting them from keystroke loggers. The system does not address the problem of output

stealing. Also, it requires servers to support user certificates.

The system recently described by Fang et al. [7] is similar to what is proposed in this thesis. It allows users to capture ciphertext from an untrusted computer using a mobile phone camera. The paper discusses ways in which the user can verify that encrypted elements haven't been maliciously repositioned by the client. The system allows users to compose secure content using their mobile phone and then to type the resulting ciphertext into the untrusted client's keyboard. Thus, the system allows reading and composing secure content without requiring a trusted proxy server or changes to the untrusted client, which are difficulties with some of the proposals discussed previously. This system shows promise in solving the untrusted client problem. It does not appear, however, to be designed to protect against untrusted service providers. First, it assumes that the key used to encrypt the content will be negotiated between the phone and the server. Secondly, it presumes service providers' assistance in properly rendering the images.

Open Sesame, developed by Bernd Borchert [4], leverages a camera and barcodes. It allows web service providers to enhance the authentication process by providing a barcode on the login screen. Using the information in the barcode, the phone is able to produce credentials which it sends via its Internet connection to the service provider. The service provider confirms the credentials and allows the user access to the account. Another related system by Borchert [5], called Sichere Fenster (German for "Secure Windows") suggests presenting an encrypted secret message on the screen that can be decoded with the help of an external device. The external device (e.g. a smart card reader) is attached directly to an untrusted machine's monitor. This allows the decrypted message to be displayed directly on the untrusted monitor, without allowing the untrusted machine to learn the message. It also allows secure input by letting the user click on buttons in the secure area. The untrusted machine can intercept the clicks, but doesn't know what controls the user is clicking. The main disadvantage of this approach is that it requires the modification of public terminal hardware.

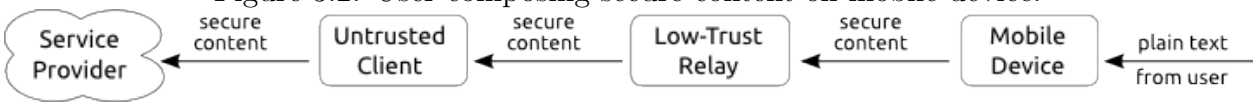
While the approaches described above improve the security of untrusted clients, there is still an opportunity for further research. Some of the systems seem to deal more with authentication of users or the authenticity of messages than securely reading and composing secret content. While other methods do allow secure reading and composing, they often involve modifying the untrusted PC, connecting a trusted device to the untrusted computer or its network, or using a trusted proxy server - all of which may make deployment more difficult. Thus, this thesis discusses a system with the ability to securely read and compose content while protecting it from untrusted networks, service providers, and clients. It takes advantage of the convenience afforded by untrusted clients without requiring them to undergo significant changes nor to allow a direct connections to their network. In addition, no personal proxy server is required.

Chapter 3

Architecture

Figure 3.1: User reading secure content from the web.

Figure 3.2: User composing secure content on mobile device.



This chapter discusses the architecture of Trusted Mobile Overlays. The basic problem addressed by the architecture is moving secure content between an untrusted computer and a trusted mobile device. If the secure content is already on the Web for a user to view, the task of the system is to get the ciphertext from the untrusted computer, where it is unsafe to decrypt, to the mobile device where it can be safely viewed (see Figure 3.1). If this secure content is instead composed by the user on their mobile device, the system needs to encrypt the message on the device, and deliver it to the untrusted client for use on the Internet (see Figure 3.2).

In addition to protecting user information from untrusted clients, one of the major design goals of Trusted Mobile Overlays is minimizing practical deployment difficulties. There are two issues in much of the previous work that would complicate deployment. First is the assumption that clients can be substantially modified, which should not be true of public computers. Second, some systems assume that users have access to their own secure proxy server, a burdensome requirement for a large class of users. The architecture of Trusted

Mobile Overlays includes a low-trust relay which addresses both of these concerns.

3.1 Untrusted Clients

For the purpose of this paper, an untrusted client is defined as a computer system on which a user would like to use the Internet. It may be untrusted because it is not in the control of the user (in a public library, for example) or it may be a computer that the user owns, which they fear is infected with malware. The client consists of hardware, an operating system, and application software, any of which may be compromised. We will assume that a user does not have administrative access to the client, including the ability to install software or connect hardware. This assumption makes the solution more generally applicable.

The approach taken by Trusted Mobile Overlays is to allow people to use untrusted clients for most of their interaction, and use their mobile devices to read and compose sensitive information. Because the mobile device component will not be aware of where the user is on the Internet, it will be the responsibility of the untrusted client to get content from the Web to the mobile device, and from the mobile device to the Web. The challenge is to do so while minimizing administrative access and trust in the client.

3.2 Service Providers

We define a service provider, trusted or untrusted, to be an entity that handles information on behalf of a user. This includes services like email providers, social networking sites, financial institutions, and e-commerce sites.

Some of these service providers, like webmail services, are considered "untrusted" not because the operating institution would purposefully misuse sensitive information, but because they are not, themselves, the intended users of that information. Also, even if an untrusted service provider is a model of integrity, it would take only one rogue employee or hacker to compromise a user's sensitive data. It would not be reasonable for the model to expect untrusted service providers to implement parts of our system. Specifically, we don't

expect that they will provide HTML mark-up or metadata designed to facilitate the use of Trusted Mobile Overlays. What *is* required from service providers is the ability for users to post and retrieve regular ASCII text.

However, our model also allows for service providers that can be trusted. This is true if the service provider is not only a relayer of data, but also an intended creator or viewer. Consider a bank, for example. A user trusts their bank with their financial information, and the bank would often be the entity creating sensitive data. From the perspective of a user, these trusted service providers can be thought of as just another user in the system. These trusted service providers would need to actively participate in Trusted Mobile Overlays to provide their patrons with the benefits of the system.

3.3 Low-Trust Relay

In an ideally-connected world, sensitive information would be able to flow freely between the untrusted computer and the mobile device (see Figures 3.1 and 3.2). In reality, however, this type of direct communication is not always possible. Since it is presumed that the user does not have administrative access to the untrusted client, it is unlikely that they can establish a direct connection (e.g., Bluetooth or USB) between the PC and the device. Also, while it is assumed that both the untrusted client and the mobile device are connected to the Internet, it is very likely that one or both are on networks that use Network Address Translation or some other solution which makes direct network connections difficult or impossible.

Since direct communication is difficult, the Trusted Mobile Overlay architecture includes a relay. The relay allows secure content to be transferred between the untrusted client and the mobile device. Because this relay can be on the public Internet, it should be accessible to both the untrusted client and the mobile device.

The design of the relay suggests that we can develop a common relay server for all users of the system, rather than require each user to possess and maintain their own secure proxy server. For example, there is no reason for the relay server to know the plaintext of,

or keys associated with, secure content. The goal is to minimize the trust required in this server. Thus, the server can support multiple simultaneous users.

3.4 Trusted Mobile Device

A trusted mobile device is a small computer under a user's control. We assume that these devices are less likely to be compromised than untrusted clients, which is reasonable because the devices are generally in the possession of the user, and the APIs available for these devices should present a smaller attack surface. Trusted mobile devices are assumed to be Internet-connected, although it is conceivable that the necessary transactions could be accomplished via the Short Message Service (SMS) or the Multimedia Messaging Service (MMS). These mobile devices need to allow input and output and be powerful enough to perform cryptographic operations in a reasonable amount of time. Modern smart phones meet the requirements for a "trusted mobile device" as defined in this architecture.

The mobile device component of the architecture is where the actual encryption and decryption takes place. Thus, it needs to provide a solution for problems like key management and distribution.

Chapter 4

Prototype Implementation

This chapter discusses a Trusted Mobile Overlays proof-of-concept prototype. Keep in mind that other implementations are possible. Presented first is information relevant to all of the components. Then, the client, relay server, and phone application are discussed in turn. Each of these component sections present possible design alternatives and then discuss the chosen implementation technique.

4.1 Definitions

Before the implementation is described in detail, we will define some terms as they are used in the context of the prototype:

AJAX (Asynchronous JavaScript and XML) Allows web pages to communicate with a server without reloading an entire frame; common in modern web applications.

Compose Content Secure content that is created by the user on their mobile device. It is encrypted on the device, and then sent to the untrusted browser where it can be posted to an HTML form.

Kiwi Domain The host name, scheme, and port of the relay server and client's JavaScript. `https://kiwi.byu.edu/`, which runs on port 443, is an example. Components of different domains are allowed limited communication by a browser's same-origin policy.

Kiwi System Includes authentication, key distribution, and data packaging components. It is used in the traditional Kiwi Trusted Overlay prototype. It was developed by Tim

van der Horst and Kent Seamons at the Internet Security Research Lab at Brigham Young University.

Read Content Secure content that is displayed to the user as base64 in the untrusted web browser. This content is sent from the browser to the phone where it is decrypted and read.

REST A simple means of communicating between distributed systems using HTTP.

Secure Content or Kiwi Package The actual ciphertext, as well as additional information needed to decrypt the message. It is often represented as base-64 encoded text. Because the prototype uses the Kiwi system, in the context of the prototype the terms "secure content" and "kiwi package" are interchangeable.

(Secure) Content ID Each secure content item is given an identifier that, along with a session id, allows the system to identify relevant content. This id also helps users match content on their mobile device and on the untrusted client.

4.2 Communication

Before describing the specifics of each component of the prototype, this section discusses an issue which cuts across the entire system: communication.

One of the primary tasks of a Trusted Mobile Overlay implementation is transferring secure content between devices. While the architecture suggests the use of a low-trust relay, it does not specify how, exactly, that relay would operate. An implementation would likely need to consider these choices:

- simulating a direct connect between the untrusted client and trusted mobile device
- storing and forwarding the messages

The prototype combines these approaches. The relay server provides a REST API which allows the client and the mobile device to communicate with the relay via HTTP. This

simplifies the design of the relay server and allows the system to take advantage of existing HTTP libraries.

One problem with using only traditional HTTP, though, is that requests must come from the client. If, for example, the relay server has a secure message for the browser to post to a web page, HTTP doesn't provide a mechanism for the server to notify the client. One solution is for the client application to poll the server: repeatedly requesting an update. This is troublesome because the more often the client polls the more quickly the user will see the new information, but the more computation and network resources will be wasted. Polling less frequently reduces the resource strain, but makes the system appear less responsive. In other words, polling creates a trade-off between resource-efficiency and perceived performance.

The untrusted client component of the prototype takes advantage of a recent development in browser technology called WebSocket [9], which allows clients and servers to maintain a persistent connection, meaning the server can notify the client of changes. Thus, in this prototype, cutting-edge browsers which support this new technology connect to the relay server with WebSocket. When they are notified of a change they use the REST API to learn the specifics of the update. To maintain compatibility with older browsers, the prototype falls back to polling the REST API directly when WebSocket is not available.

Although the mobile device application has more network communication flexibility than the browser-based component, it is convenient to use REST and WebSocket for its relay server communication as well.

4.3 Sessions Identifiers

Another area which touches all three components of the system is sessions. Sessions allow the prototype to fulfill two of the requirements of the Trusted Mobile Overlays architecture. First, sharing a unique session identifier allows an individual user's untrusted client and mobile device to be paired, which is necessary because they cannot communicate directly.

Second, sessions provide a way for the relay server to support multiple simultaneous users.

The prototype uses a session identifier approach similar to those common on the Web. Thus, even though much of the communication occurs over stateless HTTP, components can ensure that secure content is directed to the correct destination by marking the communication with a session identifier. The session identifier temporarily binds a user's untrusted client software and mobile phone application.

4.4 The Client

A Trusted Mobile Overlay client implementation needs to transfer secure content to and from a relay. This section discusses design alternatives and implementation details for the untrusted client.

4.4.1 Alternatives

Desktop Application

One option is a desktop application on the untrusted client. This program could allow a user to copy and paste read content from the web browser, and then send it to the relay. When compose content is available from the relay, the desktop application could retrieve it, and allow the user to copy and paste it into the browser.

This approach is not ideal for two reasons. First, untrusted clients are often computers in public places, meaning that users shouldn't have software installation rights. Second, copying and pasting between a web browser and another program would be inconvenient.

Another possible desktop application is a modified browser on a Bluetooth-enabled USB storage device. The modified browser could run the necessary client software as well as allow the user to connect their mobile device via Bluetooth. This would eliminate the need for communication via a relay. This option would be difficult if the client didn't allow USB connections or provide the ability to browse the filesystem.

Proxy

Another alternative is to have a web proxy which monitors the untrusted client's network traffic. This proxy could either be running on the client itself, or somewhere on the client's network. The proxy could silently listen to network traffic for read content being sent from a service provider to the client, and forward the read content to the relay.

This approach would be more convenient for the user than the desktop application, but still has some issues. First, it doesn't provide the user control over which read content they see on their phone. A user doesn't necessarily want to view every read content that comes across the network. Second, the web proxy approach would make it difficult for the user to specify where on the Web to post their compose content. Last, the lack of administrative access to the untrusted client would make this approach impractical.

Browser Extension

A third possibility is a browser extension, which is installed and runs on an untrusted client. It, like the proxy, could watch for read content and forward it to the relay. The extension approach mitigates two of the problems with the web proxy. First, it could allow the user to specify which read content to forward to the relay. Second, it would allow them to select which HTML element to use for compose content. A properly secured public computer, however, should not allow users to install browser extensions, meaning deployment of this option, too, would be problematic.

4.4.2 Untrusted Client Solution

The approach taken for the prototype is called a bookmarklet, which shares many of the advantages of the browser extension approach. A bookmarklet is simply a browser bookmark that contains JavaScript, rather than a traditional URL. It is installed in nearly the same way that a user creates a normal bookmark: by either dragging a link to the bookmark toolbar or choosing an option from the bookmarklet link's context menu. Once it is installed, the

user activates the bookmarklet by clicking on it. They need to activate it for each new page they visit. It is reasonable to assume that users of untrusted computers can be allowed to create temporary bookmarks, or at the very least copy and paste the bookmarklet's value into the address bar and press *enter*, which would have the same effect as clicking on the bookmarklet.

Cross-Domain Communication

In order to understand the reasons behind the specifics of the JavaScript bookmarklet application, it is important to appreciate the challenge of cross-domain communication. When a user invokes a bookmarklet, the JavaScript is run in the context of the current top level page, receiving the same permissions as the page itself. One important security feature of browsers is the same-origin policy, which restricts some cross-domain communication. Suppose, for example, that cross-domain communication was allowed via AJAX. This would mean that a malicious page could make an AJAX request to a user's email provider. The email provider may assume that the request was initialized by the user (especially if the cookies are sent along with the request) and dutifully return the user's inbox. The malicious page would then have access to the user's email. Users would simply have to visit the wrong web page while logged in to their email provider to fall victim to this attack. A similar attack could be carried out by allowing a page to access the Document Object Model of a frame from a different domain.

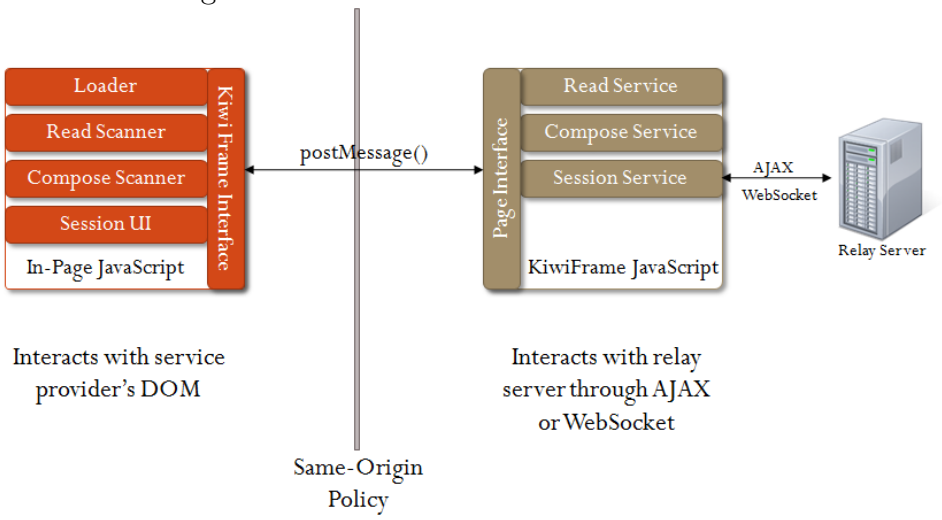
The client component prototype of Trusted Mobile Overlays, however, benefits from communicating across domains: to send read content it finds on any domain to a relay server in its own domain. Thus, we have to work around the cross-domain communication security measures. Browser vendors recognize that there are legitimate uses of cross-domain communication and have provided safe means to do so.

One such mechanism is Cross-Origin Resource Sharing [21] which uses HTTP headers to allow HTTP servers to give browsers permission to share their content with pages from

another domain. It is available in some recent versions of browsers.

Another approach, and the one used by this prototype, is the postMessage mechanism, which is widely supported in recent browsers. The postMessage function allows strings to be communicated between frames from different domains. The frames can then choose what to do as a result of those strings, taking into account the other frame's domain. The prototype injects an invisible iframe (we call it the Kiwi Frame) from the Kiwi domain into the top level page. The JavaScript in this iframe can communicate with the JavaScript injected in the top level page itself through postMessage calls. It can also communicate with the relay server via AJAX because the relay server resides in the same domain. In this way the iframe acts as a sort of communication broker between the relay server and the JavaScript running in the page, and allows the prototype to work with pages from any domain. The specific communication is described below.

Figure 4.1: Interaction of bookmarklet code.



In-Page JavaScript

The following components run in the context of the top level page. For example, if a user is interacting with his bank, this code runs as if it is part of the bank's web page.

Loader When a user invokes the bookmarklet, it downloads additional JavaScript from a server in the Kiwi domain. The first JavaScript downloaded is called the Loader. The Loader's responsibility is to download the rest of the JavaScript. The order in which the JavaScript is loaded is important because later code depends on previous code. This is somewhat difficult because some browsers download JavaScript specified in script tags asynchronously, which means scripts may run in an unexpected order. The loader handles this problem by listening for a script to finish loading before causing the next script to be loaded. In this way the Loader serializes the JavaScript download process.

Read Scanner The Read Scanner scans the page at a specified time interval, looking for text like this:

```
-----BEGIN Kiwi Message -----
UEsDBBQAAAgIAM9oL+x0cM7ChgUAAMMIAAATAAAAQnVuZGxlWzBdX21zZy5raXdpcKWW147jNhSG
X2XgWyOjLlnBjBM1q1vN8ki+U7O6qG5JTx/vJtjsTgoC5IoEyfPznA/gf/j2y1JXL3PSDzlo3nfI
...
woYFAADDCAAAEwAAAAAAAAAAAAAApIEAAAAAQnVuZGxlWzBdX21zZy5raXdpcFBLBQYAAAAAAQAB
AEEAAAC3BQAAAAA=
-----END Kiwi Message -----
```

The code interprets this as read content and places a button over the element containing the text. The button contains the text, *Send to Phone*. When the user clicks the button, the read content is sent to the Kiwi Frame, where it can then be routed to the relay server. The relay server then assigns the message a secure content id. The id is then routed back to this code, so that it can be displayed to the user; allowing them to associate the content they send from the browser with the content they receive on their mobile device.

Compose Scanner The Compose Scanner inspects the page for input elements (text areas, inputs, and editable iframes), and overlays them with a *Retrieve from Phone* button.

If there is no new compose content available, the button is grayed out. When there is new compose content the button is active and allows users to select which compose content to post. When the user clicks the button, the compose content is posted into the input element as base-64 encoded text. The user will then interact with the page and submit the form as usual in order to get the compose content to the service provider.

Session UI This JavaScript displays a session barcode to the user, so they can capture a session id with their mobile device's camera and establish a session. The UI disappears after the mobile device reports that the session is initialized.

Kiwi Frame Interface This code, which also runs in the context of the top level page, has two responsibilities. First, when it is loaded, it injects an iframe into the top level page from the Kiwi domain. It also routes messages to and from the iframe it creates.

Kiwi Frame JavaScript

The components discussed here run in the context of the Kiwi Frame. Because the iframe is from the Kiwi domain, these components are unable to interact directly with the JavaScript of the top level page, but they *can* communicate using the postMessage mechanism. The main reason this iframe exists is to allow AJAX calls to the relay server, which is on the Kiwi domain (see Figure 4.1).

Read Service The Read Service's responsibility is to push read content it receives from the web page to the relay server. It does this by making an AJAX call to a REST endpoint. Also, a read content identifier is returned from the relay, which it sends back to the button on the top-level page which caused the message to be sent.

In order to prevent requesting the same information multiple times, the Read Service (as well as the Compose Service) keeps track of the most recent id of the content it received. When making its REST request, it passes the last content id as a parameter, so that only

content with a higher id will be retrieved.

Compose Service The Compose Service is slightly more complicated than the Read Service. It retrieves compose content from the relay allowing it to be injected into an HTML element. The service keeps track of multiple compose contents, meaning a user can send multiple secure contents from their phone, and later choose which compose content goes in which form element.

If the browser in which the service is running supports WebSocket, then the Compose Service connects to the relay server with WebSocket, and waits to be notified that new compose content is available, at which time it makes an AJAX request using the REST API to get the actual content. It then notifies code running in the top-level page that new compose content is available. If the browser does not support WebSocket, it just makes the same REST API call to get new content at a given time interval.

Session Service The Session Service maintains a session id. When the bookmarklet is invoked, a session may or may not have already been established. If it has, the Session Service can just use the stored session id. If it has not been established, the service generates a random session id, and a message is sent causing the Session UI to be displayed. The session service then either connects via WebSocket or polls a REST interface, waiting for the server to notify it that the phone has confirmed the session. The session persists by using session cookies in the Kiwi domain. Thus, a new session is required each time the browser is opened, rather than for each invocation of the bookmarklet.

Page Interface Just as the code injected in the top-level page has a Kiwi Frame Interface, so too the Kiwi Frame has a Page Interface. The main responsibility of this component is to route messages between components in the Kiwi Frame and the injected top-level page JavaScript.

4.5 Relay Server

We assume there is not a reasonable means of direct communication between an untrusted client and a trusted mobile device, so a Trusted Mobile Overlay implementation needs a mechanism to relay secure content. This section presents alternatives and details relating to the relay server prototype.

4.5.1 Alternatives

Existing Servers

One solution would be to use an existing server implementation. There are essentially two general classes that could be considered.

First is existing message passing systems. A Trusted Mobile Overlays implementation could use the same type of technology that instant messaging services use to bypass the problem of endpoints being unable to directly connect over the Internet.

Another general approach would be to use an existing storage-based system. For example, the mobile device could store a file on an FTP or WebDAV server for the untrusted client to retrieve.

4.5.2 Relay Solution

Because the design of the relay server portion of the architecture is simple and specific, it made sense for the prototype to implement its own relay server, rather than try to simply use an existing server. The relay server prototype's interface is a combination of REST and WebSocket. It is implemented in Ruby using existing libraries for networking, HTTP, WebSocket, and more.

Douglas Crockford's JavaScript Object Notation (JSON) was used as the format for much of the messaging. It is well supported across platforms, light-weight, and easily converted into native data structures in ruby and JavaScript.

REST API

The REST API, through which the untrusted client interacts, has the following endpoint names, HTTP methods, and required parameters:

- bookmarklet_retrieve - GET - session_id, last_compose_content_id
- bookmarklet_send - POST - session_id, kiwi_package
- bookmarklet_is_session_confirmed - GET - session_id

The session_id is the unique session identifier shared between the untrusted client and the mobile device. The server uses it so that it can handle multiple simultaneous users; sending the right content to the right endpoint. The kiwi_package is the base64 representation of the secure content, which is described in more detail later in the Mobile Application section. The content_id field allow the endpoints to keep track of which content they've already shown to the user and allows the user to match content in the browser and on the phone. Content ids allow the untrusted client and mobile device to decide what content to request, rather than requiring the relay server to keep track of what content needs to be sent.

The REST API through which the mobile device communicates is similar:

- mobile_retrieve - GET - session_id, last_compose_content_id
- mobile_send - POST - session_id, kiwi_package
- mobile_confirm_session - POST - session_id

Note that the mobile device confirms a session, and the untrusted client checks that the session is confirmed so it can remove the Session UI barcode screen and allow the user to proceed.

WebSocket API

The prototype opens three WebSocket ports. Two of them allow the untrusted client and mobile device to be notified of new compose and read content. The other lets the client know that the mobile device has confirmed the session. This prevents the client, where WebSocket is available, from having to poll the server and improves network and battery usage on the mobile phone.

4.6 Mobile Application

The application running on the trusted mobile device is where encryption and decryption are performed. This is the only system component that accesses the plaintext and cryptographic keys. This section discusses design alternatives and demonstrates how the mobile application component of the prototype fulfills these three responsibilities:

1. Retrieve read content from the relay server, decrypt it, and display it to the user.
2. Allow the user to compose content, encrypt it, and send it to the relay server.
3. Manage the keys and other details associated with encryption and decryption.

4.6.1 Alternatives

SMS

One way to transfer the necessary keys and secure content could be through SMS (Short Messaging Service). The advantage of this approach would be allowing existing non-Internet-enabled cell phones to participate in the system. The length restriction (160 characters) and lack of guaranteed delivery may pose problems, though.

Secure Content Barcode

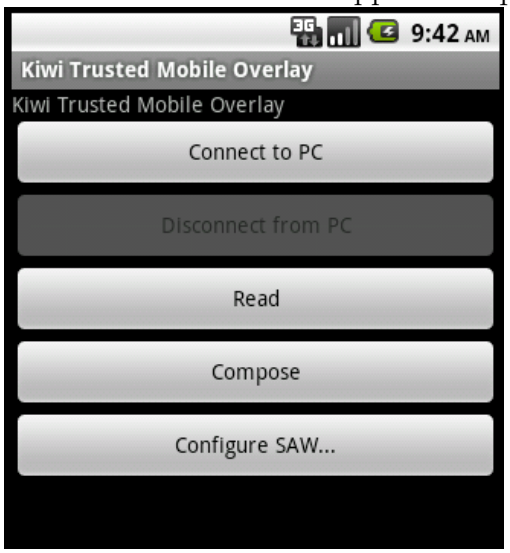
Another alternative that was seriously considered for the prototype was the use of barcodes to transfer the secure content. The code running in the untrusted browser could look for secure

content, and use a web service to convert it into a high-capacity barcode. This could then be captured by the users mobile phone camera. One possible problem with this approach is that even a system leveraging modern 2D barcodes and sophisticated mobile phone cameras would have a limit to the amount of data that could be communicated. One way to alleviate this concern would be to cycle through a series of barcodes [12], each containing a portion of the secure content.

Ultimately it was decided that the system would assume Internet-connected mobile devices, making barcodes for data transfer less relevant.

4.6.2 Trusted Mobile Device Solution

Figure 4.2: Main screen of mobile application prototype.



The mobile application component of the prototype is a program written for Android's Java-like SDK. It uses the Internet to transfer secure content and other messages necessary for encryption and decryption.

The application's main screen (see Figure 4.2) contains buttons allowing users to do the following:

Connect to PC Clicking on the button brings up a camera view that allows users to cap-

ture a barcode containing a session id from the untrusted client. When the session id is captured, the session service sends a confirmation message to the relay server, and the session id is then used in all communication with the relay server.

Disconnect from PC Causes the application to not use the current session id and enables the *Connect to PC* button, allowing the user to establish a new session.

Read Allows the user to see the available read content. The list updates when new content arrives.

Compose Here users can create secure compose content, and choose for whom it is encrypted.

Configure SAW Users enter email credentials, allowing them to authenticate to the Key Server with SAW. They can manage multiple users so they can create and view secure content with multiple email addresses. The prototype only supports Gmail email addresses for SAW authentication.

The mobile phone application took advantage of existing ideas from the Kiwi system. It was also able to leverage a significant amount of existing Java code created for other applications of the Kiwi system.

Simple Authentication for the Web (SAW)

Simple Authentication for the Web [20], which is part of the Kiwi system, is a lightweight authentication mechanism. Many online services already place a great amount of trust in email providers [8]. They allow their users to either retrieve or reset their passwords via email, indicating they trust email providers not to impersonate their users. Basically, if a user can demonstrate they can access a message sent to the email address associated with their account, that is often enough to convince services that the user is the rightful owner of the account. SAW leverages this trust by allowing users to log into services directly by proving ownership of an email address, without requiring additional passwords.

A user authenticates to a SAW-enabled service by providing only their email address to the service (no password is necessary). The service generates an authentication token, which it splits into two shares, giving part back to the browser, and sending the other part to the email address provided. If the user is able to obtain the message, they can reconstruct the token and use it to log into the service.

This prototype takes advantage of SAW in order to authenticate users to a key server, allowing them to obtain the keys required to encrypt and decrypt secure content. All of the network traffic for SAW will occur on the phone's Internet connection and should be protected with TLS or SSL. The key server authentication will certainly occur over TLS or SSL, but whether or not the interactions with the individual email providers are protected depends on the email providers' capabilities.

Packaging and Key Distribution

Another component of the Kiwi [19] system developed previously by the ISRL at BYU allows for simple key distribution. This method overcomes many of the difficulties of public key systems by providing key escrow. For example, lost keys don't mean forever-undecipherable messages, because keys can be regenerated. Another benefit of the Kiwi system, like other encryption systems which leverage user identity (e.g., [3]), is that when Bob wishes to send a secret message to Alice, he doesn't need to know her public key. In fact, at the time Bob sends the message, Alice doesn't need to have a key at all or even be aware that the Kiwi system exists. She can retrieve the necessary key after she receives the message.

For Bob to send secure content to Alice, he:

1. Composes a message (M) for Alice
2. Gets an authentication token (AT_B) from the Auth Server (AS) (the prototype uses SAW)
3. Uses AT_B to authenticate to the key server (KS)

4. Requests his creator key (K_B) from KS
5. Generates Alice's shared viewer key ($K_{B,A}$) from K_B
6. Protects M using $K_{B,A}$ (generating $\{M\}_{K_{A,B}}$)
7. Sends $\{M\}_{K_{A,B}}$ to Alice

For Alice to read the secure content, she:

1. Receives $\{M\}_{K_{A,B}}$
2. Authenticates to AS, receiving AT_A
3. Uses AT_A to authenticate to KS
4. Requests her viewer key associated with Bob $K_{B,A}$ from KS
5. Uses $K_{B,A}$ to decrypt $\{M\}_{K_{A,B}}$
6. Reads M, the original message from Bob

Although other message encryption systems could have been used for the prototype of Trusted Mobile Overlays, these components of the Kiwi system are a good fit because of the benefits of key escrow and the relative ease of key distribution.

Another interesting aspect of the Kiwi system leveraged by the prototype is its package format. Kiwi packages contain metadata, including the creator's id, which is helpful for decrypting secure content.

Chapter 5

Informal Cognitive Walkthrough

Polson et al. [15] describe a user interface analysis technique called cognitive walkthrough. Wharton et al. published a follow-up paper [22] refining the process and giving details about how the technique can be applied. The main purpose of this technique is to evaluate a user interface design's learnability by exploration. A cognitive walkthrough was famously employed in "Why Johnny Can't Encrypt," [23] a foundational paper in usable security.

A cognitive walkthrough, which incorporated many of the suggestions of Wharton's 1994 paper, was conducted during the design and implementation of the Trusted Mobile Overlays prototype. It resulted in improvements and identified areas for future improvements.

This thesis includes a proof-of-concept prototype for Trusted Mobile Overlays. A cognitive walkthrough was performed to evaluate the prototype's user interface. The walkthrough presented here analyzes an earlier version of the prototype, rather than the current iteration. Presenting the walkthrough in its original form provides insight into the improvements brought about as a result of the exercise. These changes are described at the end of the chapter. The walkthrough description below follows many of the suggestions of Wharton's 1994 paper.

5.1 Inputs to the Walkthrough

Users of the System Users of the prototype will be comfortable with the Web and with using applications on their mobile phones. They are presumed to be aware of the risks of losing sensitive information, but are not necessarily security experts.

Tasks to Analyze The prototype enables users to perform two related tasks, both of which will be analyzed. The first task is retrieving secure content from the Web and viewing plaintext on their mobile device. The second is composing plaintext on their mobile device and posting the secure content to the Web. Many of the initial steps for reading and composing secure content are the same, so they are only examined once in the analysis.

User Interface Definition The prototype contains a client and mobile phone component, which serve as the interface definition.

Correct Action Sequence for Each Task

- Actions Required for both Reading and Composing
  - (client) Click on the bookmarklet, for each web site where they wish to use the system
  - (phone) Establish a session, if not already established, by capturing the barcode on the client with the mobile phone camera
  - (phone) Create a user account, if not already created
    - Touch *Configure SAW...* button
    - Touch *Add New Account* button
    - Enter correct Gmail credentials
- Reading Secure Content
  - (client) Click *Send to Phone* button

- (phone) Touch *Read* button, if not already in read mode
- (phone) Touch correct secure content menu item

- Composing Secure Content
  - (phone) Touch *Compose* button, if not already in compose mode
  - (phone) Choose desired identity for composer
  - (phone) Enter identity of receiver (viewer)
  - (phone) Enter plaintext
  - (phone) Press *Send to Browser* button
  - (client) Click on *message #* with the correct number to post content to HTML form
  - (client) Submit the HTML form as usual (e.g., click *send* in email)

5.2 Walking Through the Actions

We will now step through the actions required to complete the desired tasks. For each step, we either tell a credible story relating how the user could correctly complete the action or a story where the user may fail to do so. We make design recommendations to remedy some issues.

5.2.1 Actions Required for both Reading and Composing

Click on the bookmarklet This assumes that the bookmarklet has already been "installed" by saving it as a bookmark. Since installation and use of a bookmarklet is not a common task, we will assume that the user has received some training as part of their introduction to the system. One difficulty users will face is that each time they want to use the system, and a new page has been loaded, they will need to click on the bookmarklet. Compounding the problem, some sites taking advantage of AJAX techniques may appear to have loaded a new page, but, in fact, have just replaced the contents of some elements of the

current page. In these cases the bookmarklet should *not* need to be clicked. The problem of whether or not to click on the bookmarklet may be confusing to users. One way they can reliably know if they need to load the bookmarklet is to look for the presence of the controls to send or retrieve content. If they are not there, users should try clicking on the bookmarklet again.

The bookmarklet may be one of the more difficult parts of the system to use, but the difficulties arise from the nature of bookmarklets, rather than this specific implementation. The choice to use bookmarklets for the untrusted client is a known compromise of usability, but helps satisfy the design goal of ease of deployment. Thus, there are no specific design recommendations, but instead an acknowledgement that user training will probably be necessary.

Figure 5.1: Browser session-establishment element.

Establish a Session (client) The first time a user starts using the prototype in a browsing session, they will need to establish a session between their phone and the untrusted computer. Once the session is established, it will persist until the user closes the browser window. In order to make the user aware of the need to establish a session, part of the client browser's display area is overlayed with an HTML element containing a session barcode (see Figure 5.1). This barcode takes up a large portion of the web page, and would be hard for the user to ignore. In small font there is a prompt, "Please confirm session id on the phone:

1A2B3C. . . ."

The first problem with the barcode prompt in the browser is that the font prompting the user to confirm the session id on the phone is probably too small, it should be emphasized so the the user notices it. As 2D barcodes become a more common way to communicate data to mobile phones, users may need less prompting: they may see a barcode and quickly recognize they can capture it with their phone. If this is the case, we need to make sure they know to capture the barcode using our application, and not their normal barcode capturing program.

The next problem is the hexadecimal session id. It's presence allows users to manually type the session id into their phone, but the phone's application currently doesn't support that behavior. Perhaps it would be better to display it in a lighter-colored smaller font, to satisfy advanced users curious about what is being transfered to their phones, but to avoid drawing attention away from more important instructions.

Another serious issue is that the session HTML element doesn't always properly overlay the page, but instead gets combined with existing elements on the page. Similar problems also occur with other elements which we insert into the page. With some engineering effort, the elements could probably be made more resilient to differences in the pages into which they are being loaded.

Establish a Session (phone) Once a user understands that they need to capture the barcode to establish a session, they can use their mobile phone to do so. First, they need to start the prototype phone application. This understanding could either come from training or could be part of the prompt in the browser of the untrusted client.

Once the phone application is started, the *Connect to PC* button is available on the main screen. When the user clicks the button, the phone enters camera mode and begins scanning for barcodes. When the user points the phone at the computer screen and is able to successfully capture the barcode, both the camera screen on the phone and the barcode

element in the untrusted browser go away, and it should be clear to the user that the system is ready to use.

One improvement would be to disable buttons on the main screen, like *Read* and *Compose*, that cannot be used without a valid session. This should quickly direct the user to establish a new session.

Although it is theoretically easier to capture a barcode than to manually enter a large session identifier, sometimes it is difficult to capture the barcode. The software doesn't always recognize it quickly, even though it appears to be in the right place in the view finder. Also, users with shaky hands may have a difficult time getting the camera to work correctly. Thus, while the barcode-capture approach makes sense as the primary way to establish a session, the phone application should probably provide an option to enter the barcode manually for situations where it is difficult to capture the barcode.

Another problem with the barcode capture process is that when the barcode is recognized, the message "Found plain text" is printed to the phone's screen before the program returns to the previous screen. This is not accurate, and should be removed, if possible.

Create a User Account (phone) In order to encrypt or decrypt messages, the phone application needs to check users' email accounts to retrieve auth token shares for SAW. The prototype supports only Gmail, but could be extended to support other email providers. If there are no users set up, buttons on the main screen which require user information, like *Read* or *Compose*, are disabled. This is beneficial, so that users know they need to create a user account to continue. Users access the user management functionality by pressing the *Configure SAW...* button on the main screen. Here they see a list of user accounts, and a button allowing new user accounts to be created.

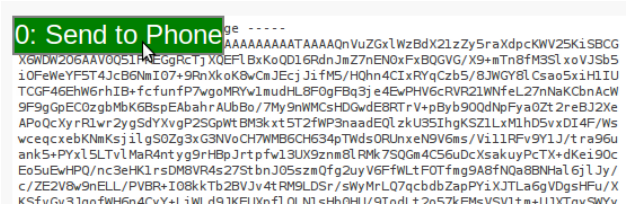
While having multiple user accounts on the same phone is technically possible, it is somewhat confusing. This functionality could still be available, but optional. The default user management screen would just prompt a user to enter their Gmail username and

password, rather than have information about multiple user accounts.

Another issue with user accounts is that people need to understand why they need to provide their Gmail credentials. The user of this system is assumed to be security conscious, so they might be concerned when they are asked to divulge their email username and password. This screen should probably have a button labeled *why do you need my Gmail address and password*, which explains SAW at a high level.

5.2.2 Reading Secure Content

Figure 5.2: Mouse hovering over *Send to Phone* button.



Click *Send to Phone* Button (client) After the user enables the bookmarklet, secure content on the web page is overlaid with a blue element, which in white lettering says, *Send to Phone*. This seems to be an accurate label for what the user wishes to accomplish. When the user clicks this button, the text is altered to include the secure content id; for example, *Send to Phone(12)*. This id helps them match the content they just sent to the phone with the content that arrives on the phone: they will both have the same id.

One problem with the the *Send to Phone* button is that it is probably too transparent until a user hovers over it, at which time it becomes opaque and the background turns green (see Figure 5.2). This transparency helps to avoid obscuring the underlying text, but the underlying text is unintelligible base64, so ensuring the button is seen is more important than not obscuring the text. Also, it should probably be animated when it is put on the page to draw a user's attention.

Another problem is with the secure content id: a user has little chance of guessing what the number means. The number could instead be rendered in small font along with

an explanation of how it can be used. Or, perhaps, the id could be removed from the client altogether; it may cause more confusion than it provides help.

Touch *Read* Button (phone) It makes sense that once the user has clicked the *Send to Phone* button they would turn to their phone to find the message. Because they are intending to read the message, it also makes sense that they would then touch the phone application's *Read* button.

Touch Correct Secure Content Menu Item (phone) After the user enters read mode, they are presented with a list of messages they have sent to the phone, identified by their secure content id (the same id displayed by the browser on the client). The user can then press the line with the correct id to view the plaintext.

This approach allows users to keep a collection of secure content to review later (although the phone application does not attempt to persist read content between invocations of the application anyway). A better approach would probably be to just go directly to the newest message when the user clicks the *Read* button. This would reduce the number of steps for the user to accomplish their goal (read secure content), and would also eliminate the need for secure content identifiers in the user interface. If they want to read a message they had previously read, they could click the *Send to Phone* button again.

The application could even go a step further, and simply decrypt and display secure content for the user when it is retrieved.

5.2.3 Composing Secure Content

Touch *Compose* Button (phone) The label for this button makes sense: it helps the user get a step closer to accomplishing their goal of composing secure content.

Choose Identity for Composer (phone) The compose screen has a control that allows the user to select which identity they want to use to compose the message. This concept may

be confusing. Instead of a dynamic control, a static label reminding them of their current email address could be used. Advanced users could turn on an option allowing selection from among multiple personas.

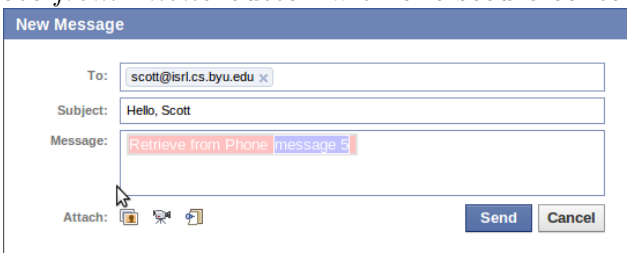
Enter Identity of Receiver (phone) This step allows the content to be encrypted for the specified user, and is relatively clear for the case of an untrusted service provider (e.g. webmail). They simply enter the email address of the person to whom they want to send the message. This may be somewhat frustrating for a user though, because they likely also have to enter the same address on the browser screen so the service provider knows to whom to route the message. It would be convenient if this address didn't have to be entered again.

Having to enter the address at this point is even more problematic for trusted service providers (e.g. banks). They would likely need to include instructions near the HTML box with what identifier to use to encrypt the message. This may be confusing because people may not associate an email address with the identity of their service provider. The service provider could provide this information in such a way that the client could inform the mobile application of possible email addresses to use for the current page. The problem with this approach though, is that the untrusted client could spoof the message, and encourage the user to encrypt the content in such a way that the client could read it. Thus, the security implications do not allow for any specific recommendations presently, but this is an area of the interface which could be further examined in the future.

Enter Plaintext (phone) The user should recognize that their task is to use their phone to protect plaintext, and the text box seems like a natural place for it to be entered. One problem with the control for this interface is that the input box is only one line tall, and grows as text is entered. The initial height should be increased to make it clear that more text can be entered here. Another problem is the label for this text box reads "Secure content." Making this label more explanatory may give users a better chance of recognizing that they need to enter their plaintext in the text box.

Press *Send to Browser* Button (phone) Users send secure content by clicking the *Send to Browser* button. Text below the button reflects that the content is being sent, and then echoes the secure content's id. This feedback could be more effective by indicating that the operation completed successfully, rather than only showing an id. Another suggestion is to use another thread for operations that freeze the user interface when clicking this button. Additional threads are used in other parts of the mobile prototype, but clicking this button sometimes causes the UI to freeze.

Figure 5.3: *Retrieve from Phone* button with one secure content item available.



Click on *Message #* (client) Once the user has successfully sent secure content using their phone, they need to use the untrusted client to post the content to the web. The phone application doesn't prompt the user to switch to the computer, which it probably should.

Until compose content is available to the untrusted client's web browser, the *Retrieve from Phone* element is gray. Once compose content is available it turns red, and inside of it elements with a blue background containing *Message #* appear (see Figure 5.3). The "#" is the secure content id, associated with the message, which was also shown on the phone. The user can thus choose which message to post to the form.

Again, the secure content ids may be too complicated. Instead, secure content could simply be available or unavailable, rather than forcing users to choose which content to use by matching ids from their phone in their browser.

Submit the Form as Usual (client) After the secure content has been placed in the HTML form, it needs to be posted back to the server. This is done by clicking the *submit*

button, or whatever mechanism the user would usually use to submit the form. This step seems natural, because the contents of HTML forms are not usually submitted to the server until the user takes an explicit action. Here, users take the same action as they usually would to submit a form.

5.2.4 Improvements

Performing the cognitive walkthrough revealed several areas of improvement for the prototype of Trusted Mobile Overlays. As a result, the following improvements have been made:

- Session Establishment
  - Session id in HTML overlay is de-emphasized (smaller font)
  - HTML element prompts user to use phone to capture barcode
  - More buttons in phone application are disabled until session is established
- Creating User Account
  - "Why do you need my Gmail address and password" link, explaining that the credentials are used to validate the email address and referring them to the SAW paper [20]
- Reading Secure Content
  - *Send to Phone* button is less transparent
  - Secure content id is de-emphasized, with a tool-tip explaining its use
- Composing Secure Content
  - Plaintext text box is taller than one line
  - Label for plaintext text box is more helpful
  - Phone UI indicates compose content sending success

- After compose content is sent, phone prompts user to use untrusted computer

Other suggestions resulting from the walkthrough are left as future work.

Chapter 6

Threat Analysis

This section contains a threat analysis of the Trusted Mobile Overlays architecture and the prototype implementation. A threat analysis of another implementation of the architecture would likely not overlap exactly with the analysis for the prototype described herein.

6.1 Secure Content Disclosure

The system uses end-to-end encryption, which means that untrusted networks, clients, service providers, and even the relay have no access to the plaintext. For secure content to be disclosed, an attacker would need to gain access to a user's keys.

One way this could be accomplished would be to compromise the key server. To mitigate this attack the key server could use a hardware token that doesn't allow remote extraction of keys. In this way, the attacker should only be able to generate keys for the time that they have access to the server.

Malicious individuals could also gain access to keys, and thus plaintext, by compromising the key server authentication process or the phone itself, which are discussed in the next two sections.

6.2 Phone Compromise

The architecture used for Trusted Mobile Overlays assumes a secure trusted mobile device. This section explores the case in which the phone, itself, becomes an "untrusted client."

Phone Stealing If an attacker steals the device, they could read and compose content as if they were the device's owner. One difficulty an attacker who has stolen a phone may experience is actually getting to the secure content, which would often be only available after providing account credentials (e.g. username and password for a social networking site). We can assume, though, that it is possible for an attacker to gain access to secure content (for example, if the credentials are stored on the phone).

One approach a mobile application implementation could take would be to encrypt keys, auth tokens, and email account details using a key generated from a password or PIN. The system could lock this sensitive information whenever the device is turned off or put in sleep mode, or after a period of time.

Remote Attack The mobile phone is connected to the Internet, so it is vulnerable to network-based attacks. If such an attack were successful, encrypting sensitive information stored on the phone, as described above, would help to alleviate the risk.

Malware The assumption, for now, is that mobile phones tend to be less susceptible to attack than desktops PC's. If an attacker does gain control over a phone with malware, though, there is great risk to a user. The malware could wait until sensitive data is unlocked, and thereby gain access to all keys and plaintext. This could be mitigated by using the anti-virus software that is emerging for mobile phones.

6.3 Authentication Vulnerabilities

SAW was chosen as the key server authentication mechanism because of its ease-of-use. Not all email providers support TLS, so the half of the auth token sent by the auth server to the email provider may be captured. This would not provide an attacker much benefit, though, without also obtaining the half of the auth token sent directly back to the user. A more serious weakness of SAW is the ability of a malicious email provider to impersonate

their users. See van der Horst and Seamons's paper [20] for more discussion on the security considerations of SAW.

If the risks of SAW are determined to be too great, an implementation of Trusted Mobile Overlays could provide an alternative mechanism, like usernames/passwords or certificates, to grant access to a key server. If key escrow - having a trusted 3rd party host keys - is itself too risky, an entirely different key distribution mechanism could be employed.

6.4 Creator Email Address Disclosure

A primary goal of the system is to minimize the trust required in the relay server. In the current implementation, the email address of the secure content creator is not encrypted so that the reader can know which viewer key to use to decrypt the message. This sensitive data is available to the relay server, which is a problem because the relay could generate a one-stop list of users of the system. This list would also make a good collection of valid email addresses for a spammer.

This threat could be averted by applying an extra layer of encryption on top of the existing Kiwi packaging format. The code on the untrusted client could generate a symmetric key, which could be communicated to the mobile device via the same barcode that contains the session id. Using this shared key, the client and mobile device could apply one more layer of encryption, thereby protecting creator email addresses from the relay server.

Another remedy for the loss of creator email addresses would be to prevent them from passing through the relay server at all. If the addresses were displayed as text or in a barcode on the client screen, they could be used directly on the mobile device without passing through the relay. The disadvantage of this approach would be requiring users to capture barcodes more often.

6.5 Content Swapping

An untrusted service provider or client could cause confusion by swapping secure content. For example, an email provider could swap the contents of two secure emails, one containing the response "yes", and the other containing, "no". Or, a malicious client could perform the same attack by rearranging a web page's Document Object Model. This risk is inherent in the design of the system, because it integrates into the existing framework of the web. Also note that this attack is not unique to our system: malicious entities can already swap plaintext content. If the risk from this type of attack is too high, an implementer could consider techniques suggested by Fang et al. [7].

6.6 Denial of Service

Each untrusted part of the Web could effectively perform a denial of service attack on the system. For example, a service provider could simply delete all data it identifies as secure content, or a client could simply refuse to forward secure content to the relay server. This type of risk is inherent in the system because it tries to co-exist with the existing Web infrastructure.

6.7 Session Hijacking

Another possibility is an attacker stealing a user's session id to try to impersonate them. The attacker could be near the untrusted computer and capture the same barcode as the user. This would allow the attacker to request the user's secure content from the relay server, but they could not read the content because they should not be able to access the user's keys.

Forged Secure Content Although an attacker who has stolen a session id cannot read secure content, they can *forge* secure content by encrypting a malicious message with their own creator key. Then, they could use the stolen session id to send the content to the relay server as part of the user's session. At this point, the secure content would be available on the

untrusted client. If a user wasn't careful about checking content ids, they may accidentally submit the forged content to their service provider.

One way to prevent this attack would be to require the user to enter their email address at the untrusted client. This would allow the attack to be thwarted by either (1) the software on the untrusted client, or (2) the reader's software trying to decrypt the message.

When the phone creates secure content it includes the creator's email address, unencrypted, in the content's metadata. The code on the untrusted client could then verify that the email address entered into the client and the email address in the metadata match, otherwise it would reject the content.

If an attacker realized that the addresses needed to match, they could insert the legitimate user's email address into the forged content's metadata. This would cause the software on the client not to reject the message as fraudulent. The discrepancy would be caught, however, by the software reading the content. When the receiving party attempted to decrypt the message, they would try to use the key associated with the legitimate user's email address. Because the content was actually encrypted with the attacker's key, however, the decryption would fail and the forged content could, again, be rejected.

6.8 Shoulder Surfing

Trusted Mobile Overlays is designed for public places, like libraries or Internet cafés, where there is an increased risk for shoulder surfing. The system reduces the risk of shoulder surfing because it encourages users to read and compose secure content on their mobile phones, which are more resistant to shoulder surfing than large desktop computers.

Chapter 7

Conclusion

7.1 Contributions

This thesis presents the architecture of Trusted Mobile Overlays, a system allowing users to enjoy the conveniences of a full-size public computer, while being protected from malicious clients and service providers. The architecture achieves this with components for untrusted computers and trusted mobile devices. Trusted Mobile Overlays also includes a low trust relay which allows it to overcome some of the deployment hurdles of previous work. In addition to the general architecture, this thesis provides the description of a prototype implementation and a threat analysis.

7.2 Directions for Future Work

7.2.1 Usability Study

One promising area of future work is studying the usability of Trusted Mobile Overlays. Such research could begin with the prototype and its Cognitive Walkthrough presented herein. Studying usability would help to improve the design of the system and gauge users' interest in this type of security. Some questions that might be considered are:

- How much interaction should occur on the client versus the phone?
- How do users respond to bookmarklets?
- Which mobile platforms are the most important to users?

- What is the point at which the difficultly of using such a system outweighs the security benefits?

7.2.2 Alternative Key Distribution

The prototype described in this paper takes advantage of the convenient Kiwi system for key distribution. While the prototype currently employs SAW to authenticate users to the key server, other authentication methods like certificates, usernames/passwords, or OpenID could be considered. Additionally, a Trusted Mobile Overlays implementation could not rely on a central trusted key server, using instead distributed key management methods like PGP or S/MIME certificates. Future research could investigate how the use of other techniques affects the implementation or usability of the system.

References

[1] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and delegation with smart-cards. *Science of Computer Programming*, 21(2):93–113, 1993.

[2] D. Balfanz and E. Felten. Hand-held computers can be better smart cards. In *Proceedings of USENIX Security*, volume 99, 1999.

[3] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. 2001.

[4] B. Borchert. Open sesame.

[5] B. Borchert. Sichere fenster.

[6] D. Clarke, B. Gassend, T. Kotwal, M. Burnside, M. Van Dijk, S. Devadas, and R. Rivest. The untrusted computer problem and camera-based authentication. *Lecture Notes in Computer Science*, pages 114–124, 2002.

[7] C. Fang and E.-C. Chang. Securing kiosk using mobile devices coupled with visual inspection. *CoRR*, abs/1003.0723, 2010.

[8] S. L. Garfinkel. Email-based identification and authentication: An alternative to pki? *IEEE Security and Privacy*, 1:20–26, 2003.

[9] I. Hickson. The websocket api. Technical report, World Wide Web Consortium (W3C), 9 2009.

[10] R. Jammalamadaka, T. van der Horst, S. Mehrotra, K. Seamons, and N. Venkasubramanian. Delegate: A proxy based architecture for secure website access from an untrusted machine. In *Proceedings of the 22nd Annual Computer Security Applications Conference*, pages 57–66, 2006.

[11] M. Mannan and P. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. *Financial Cryptography and Data Security*, pages 88–103, 2007.

[12] J. McCune. *Reducing the Trusted Computing Base for Applications on Commodity Systems*. PhD thesis, Carnegie Mellon University, 2009.

[13] J. McCune, A. Perrig, and M. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. *International Journal of Security and Networks*, 4(1):43–56, 2009.

[14] B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. *Lecture Notes in Computer Science*, 4107:1, 2006.

[15] P. Polson, C. Lewis, J. Rieman, and C. Wharton. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies*, 36(5):741–773, 1992.

[16] R. Sharp, J. Scott, and A. Beresford. Secure mobile computing via public terminals. *Lecture Notes in Computer Science*, 3968:238, 2006.

[17] R. Sharp, A. Madhavapeddy, R. Want, and T. Pering. Enhancing web browsing security on public terminals using mobile composition. In *Proceedings of the Sixth International Conference on Mobile Systems, Applications, and Services*, pages 94–105, New York, NY, USA, 2008.

[18] G. Starnberger, L. Froihofer, and K. Goeschka. Qr-tan: Secure mobile transaction authentication. In *2009 International Conference on Availability, Reliability and Security*, pages 578 –583, 2009.

[19] T. W. van der Horst and K. E. Seamons. Kiwi. Work in progress as of December, 2010.

[20] T. W. van der Horst and K. E. Seamons. Simple authentication for the web. In *3rd International Conference on Security and Privacy in Communication Networks*, pages 473–482, 2007.

[21] A. van Kesteren. Cross-origin resource sharing. Technical report, World Wide Web Consortium (W3C), 7 2010.

[22] C. Wharton, J. Rieman, C. Lewis, and P. Polson. The cognitive walkthrough method: A practitioner's guide. In *Usability Inspection Methods*, pages 105–140. 1994.

[23] A. Whitten and J. Tygar. Why Johnny cant encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium*, pages 169–184, 1999.

[24] M. Wu, S. Garfinkel, and R. Miller. Secure web authentication with mobile phones. In *DIMACS Workshop on Usable Privacy and Security Software*, 2004.

[25] C. Yue and H. Wang. Sessionmagnifier: a simple approach to secure and convenient kiosk browsing. In *UbiComp09: Proceedings of the 11th ACM International Conference on Ubiquitous Computing*, pages 125–134, 2009.