Fall 2014

Dual channel-based network traffic authentication

David Irakiza
Louisiana Tech University

DUAL CHANNEL-BASED NETWORK TRAFFIC AUTHENTICATION

by

David Irakiza, B.Sc., M.S., M.S.

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

November 2014

LOUISIANA TECH UNIVERSITY

THE GRADUATE SCHOOL

20th June, 2014

Date

We hereby recommend that the dissertation prepared under our supervision by David Irakiza

entitled

Dual channel-based network traffic authentication

be accepted in partial fulfillment of the requirements for the Degree of

Doctor of Philosophy

[illegible]

Supervisor of Dissertation Research

[illegible]

Head of Department

Computational Analysis and Modeling

Department

Recommendation concurred in:

[illegible]

[illegible]

[illegible]

[illegible] DEXTER CAHOY

[illegible]

Advisory Committee

Approved:

[illegible]

Director of Graduate Studies

[illegible]

Dean of the College

Approved:

[illegible]

Dean of the Graduate School

GS Form 13a
(6/07)

ABSTRACT

In a local network or the Internet in general, data that is transmitted between two computers (also known as network traffic or simply, traffic) in that network is usually classified as being of a malicious or of a benign nature by a traffic authentication system employing databases of previously observed malicious or benign traffic signatures, i.e., blacklists or whitelists, respectively. These lists typically consist of either the destinations (i.e., IP addresses or domain names) to which traffic is being sent or the statistical properties of the traffic, e.g., packet size, rate of connection establishment, etc. The drawback with the list-based approach is its inability to offer a fully comprehensive solution since the population of the list is likely to go on indefinitely. This implies that at any given time, there is a likelihood of some traffic signatures not being present in the list, leading to false classification of traffic.

From a security standpoint, whitelists are a safer bet than blacklists since their underlying philosophy is to block anything that is unknown hence in the worst case, are likely to result in high false rejects with no false accepts. On the other hand, blacklists block only what is known and therefore are likely to result in high false accepts since unknown malicious traffic will be accepted, e.g., in the case of zero-day attacks (i.e., new attacks whose signatures have not yet been analyzed by the security community).

Despite this knowledge, the most commonly used traffic authentication solutions, e.g., antivirus or antimalware solutions, have predominantly employed blacklists rather than whitelists in their solutions. This can perhaps be attributed to the fact that the population of a blacklist typically requires less user involvement than that of a whitelist. For instance, malicious traffic signatures (i.e., behavior or destinations) are usually the same across a population of users; hence, by observing malicious activity from a few users, a global blacklist that is applicable to all users can be created. Whitelist generation, on the other hand, tends to be more user-specific as what may be considered acceptable or benign traffic to one user may not be considered the same to a different user. As a result, users are likely to find whitelist-based solutions that require their participation to be both cumbersome and inconveniencing.

This dissertation offers a whitelist-based traffic authentication solution that reduces the active participation of users in whitelist population. By relying on activity that users regularly engage in while interacting with their computers (i.e., typing), we are able to identify legitimate destinations to which users direct their traffic and use these to populate the whitelist, without requiring the users to deviate from their normal behavior. Our solution requires users to type the destinations of their outgoing traffic requests only once, after which any subsequent requests to that destination are authenticated without the need for them to be typed again.

Empirical results from testing our solution in a real time traffic analysis scenario showed that relatively low false reject rates for legitimate traffic with no false accepts for illegitimate traffic are achievable. Additionally, an investigation into the level of inconvenience that the typing requirement imposes on the users revealed that, since users are likely to engage in this (typing) activity during the course of utilizing their computer's resources, this requirement did not pose a significant deterrent to them from using the system.

DEDICATION

This dissertation is dedicated to my family – my parents, Mr. and Mrs. Ezra and Lydia Ndagije, as well as my siblings, Andrew, Anna and Mike.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

There are numerous people that I owe a ton of gratitude for enabling me to get this far in my educational pursuits. It would be a futile effort for me to attempt to list each of you by name but your support is highly valued and appreciated. Nonetheless, there are some key individuals who in one way or the other have been very instrumental in enabling me to achieve my doctorate and deserve special mention.

First of all, my Ph.D. advisor, Dr. Vir V. Phoha, for his invaluable support and guidance during the entire period of pursuing my doctoral studies. Dr. Md. E. Karim, who has been my immediate supervisor for the greater part of my Ph.D. journey and has proofread and edited all my academic writings.

The other members of my advisory committee – Dr. Weizhong Dai, Dr. Jinko Kanno, Dr. Travis Atkison and Dr. Dexter Cahoy – for their helpful comments and feedback in not only putting this dissertation together, but also throughout the course of my graduate studies.

My lab mates, both past and present – Justin Rice, Abdul Serwadda, Ankunda Kiremire, Shafaeat Hossain, Abir Rahman, Harry May, Zibo Wang, Abena Primo, Jundong Chen, Diksha Shukla, Rajesh Kumar, Saba Ramazani, Andrew Gardner. These folks have always been a source of encouragement throughout this journey.

My American family – Ed, Judy and Nick Charles – for giving me a home away from home and always making sure I am well taken care of. The numerous friends that I have made along the way who have in many ways helped me cope with the challenges and stresses that come along with pursuing a Ph.D. Again, I can only mention but a few – Krystal Corbett, Alicia Boudreaux, Oneka Cummings, Jana Melvin, Sara Hahler, Jacob and Rachel Parks.

Lastly, but perhaps more importantly, my parents Ezra and Lydia Ndagije, my siblings – Andrew Dushime, Anna Mutesi and Michael Mwizerwa – for their sincere and heartfelt words of encouragement, prayers and support throughout this journey. I am blessed to have such a strong support system and I honestly believe I would never have made it without you. Thank you!

CHAPTER 1

INTRODUCTION

1.1 Background

Authentication plays a vital role in securing a system from potential intruders as it provides an avenue to guarantee the privacy and confidentiality of resources on the system. Recently, the term *active* (*continuous*) *authentication* [8] has been contrived in reference to the act of periodically authenticating the users of a system to ensure that no unauthorized users gain access to the system at any point. Although the use of this term is fairly recent in the authentication community, the underlying principle of continuous system monitoring has been in existence for some time in other authentication areas such as in real time network traffic monitoring and/or analysis. In this dissertation, we explore continuous authentication from the perspective of ensuring the data that is continually being transmitted from one computer to another, either in a local network or globally on the Internet, is authenticated. In the rest of this dissertation, we loosely refer to this data as (network) traffic.

In traffic-based continuous authentication, many systems have taken the approach of creating impostor (malicious traffic) templates based off of which to compare unknown traffic during classification. The disadvantage with this approach, though, is that the creation of a complete template is infeasible since the set of potentially malicious traffic is indefinite, thereby resulting in a significant number of malicious traffic going undetected. Although the use of templates that are built based on benign traffic eliminates this problem, this has not been fully explored in many solutions because it usually involves a significant effort on the part of users in the template building process, a scenario that is deemed undesirable to a user.

This dissertation proposes a benign traffic-based template generation approach that minimizes user participation in the template building process. The template used in our solution comprises the destinations to which users send traffic. The template is updated with new destinations after they have been authenticated by a user through typing. This signifies that those destinations are associated with benign traffic. In the authentication process, each destination to be authenticated, is accompanied by an authentication code whose integrity, in the face of possibly malicious traffic, is guaranteed using a recently proposed concept of dual channel approaches that are used for message authentication when one of the channels is assumed to be insecure.

In these dual channel protocols, information transmitted through one channel is authenticated using information transmitted over the other channel. The motivation behind the use of dual channel authentication protocols is the assumption that in a properly designed dual channel protocol, impostors cannot authenticate themselves if, (i) just one of the channels is compromised, or (ii) both channels are compromised but attacks on them are not coordinated. Existing literature details two families of dual channel protocols, namely: non-interactive (e.g., [2, 16, 34, 27, 36]) and interactive (e.g., [37, 43, 28]) protocols.

The concept of using duality-based solutions to strengthen authentication systems is not unique to dual channel protocols. For instance, it has been previously explored in, among others, dual factor based authentication [1, 47, 7] and dual server based password management [5, 48, 24, 23]. The former propose the use of two factors in the authentication process. These factors can be chosen from *something that the user knows* e.g., a password, *something that the user has* e.g., a smart card, or *something that the user is* e.g., a biometric signature. This idea was proposed with the view that since impostors cannot easily gain access to both authentication factors, they (impostors) cannot be falsely authenticated as a genuine user.

The latter, on the other hand, consider scenarios where the server side compromises are likely to occur. Generally in these solutions, the user (client) has access to and communicates with only one of the servers through a secure channel that can be realized via Secure Sockets Layer (SSL) [5]. Taking the example of a password-based authentication scheme, these solutions propose the splitting of a user's password into two parts with each server keeping one of the parts. During authentication, the user is independently verified by both servers which communicate via a protected channel [48]. These solutions also provide for server integrity verification which is done through a series of cryptographic challenges that are sent back and forth between the client and server during the authentication process.

1.2 Research Focus

One of the primary threats of malware infection is the leakage of information from a compromised computer to an external malicious entity cooperating with the malware without the user's knowledge or permission. We refer to this problem as *"information exfiltration"*. This scenario can be visualized for instance when a bot program is installed on a target computer and upon installation, the program accesses specific information (e.g., account passwords, credit card information) which it then remits to a predetermined destination such as the bot's command and control server.

Bearing this scenario in mind, the work in this dissertation aims to avert this type of *information exfiltration* by authenticating the outgoing traffic from a user's computer. By visualizing the user's computer as one entity and a proxy, through which traffic is filtered before it is sent to (and received from) the Internet, as another entity, we design a non-interactive dual channel protocol for authenticating the outgoing traffic. The choice of non-interactive over interactive protocols is made largely with the aim of minimizing additional overhead costs in terms of back and forth communication between entities during authentication, which implies that the protocol operation costs are reduced.

In literature, the non-interactive family of dual channel protocols, e.g., [16, 34, 2, 27, 36, 33, 40, 26] use a narrow-band authenticated channel and a wide-band insecure channel for communication. A remote verifier authenticates the information received through the wide-band insecure channel using a piece of brief information received through the narrow-band authenticated channel that an adversary has no (or limited, for some protocol) control over. These protocols are called non-interactive since, by design, information flow in them is unidirectional, which keeps them lightweight.

However, the existing non-interactive dual channel protocols are only ideal for applications that do not require frequent transmission of information over the authenticated channel because the authenticated channel in these protocols is presumed to be human aided, implying that users are required to be explicitly engaged in each authentication event. An application example is the authentication of a computer to a wireless printer by entering the computer's identification code to the printer, manually. If the manual authenticated channel is replaced by a non-manual authenticated channel, then remotely located verifiers become prone to a channel spoof attack, as explained in Chapter 2, defeating the protocols' purpose.

In contrast, the solution presented in this dissertation infers implicit current or prior approval of a destination from the users before authenticating the traffic sent from the host computer to that destination. The underlying protocol ensures that such inference is not corrupted by the malware residing in the host computer. It authenticates each outbound user-request by analyzing the keys (not the keystroke timings) a user types during the normal course of computer usage.

Our solution uses two communication channels, an insecure one which connects to a host computer and an authenticated one that bypasses the host, and connects to a remote verifier. A string processor sits between these channels and the keyboard. It controls the flow of keys from the keyboard to the channels, but it does not accept any incoming data or execution requests other than the signals necessary for the basic operation of the keyboard. We assume that the keyboard and string processor are not pre-compromised during manufacturing or in the supply chain.

The solution is non-interactive because it does not require the user to respond to any verifier-request for the authentication to take place. However, it requires that the user types a destination IP/domain name for the first time a traffic request is sent to that destination. This can affect user convenience, for instance, if all the external links in a Web page need to be typed. A possible tradeoff between security and user convenience, in this case, can be achieved by letting such links received in response to an authenticated request be considered as legitimate for a temporary period. We propose a protocol extension for this that can be integrated with the core dual channel non-interactive protocol executed in the verification server. We also propose another protocol extension to handle scenarios where exfiltration is done via external third party legitimate servers (e.g. Webmail servers). However, implementation of this extension requires the participation of the associated third party legitimate servers and hence is not covered in this dissertation.

The proposed protocol does require an auxiliary manual channel for synchronizing certain operations between the communication parties. This is, however, a one-step process required during the initial setup. Subsequent continuous authentication does not require this manual channel's involvement.

1.3 Contributions of this Dissertation

The contributions of this dissertation are outlined below:

1. We introduce the first non-interactive dual channel protocol for traffic authentication purposes. This protocol is designed to be resistant to channel spoof

attacks that existing non-interactive dual channel protocols suffer from when applied to continuous authentication scenarios.

2. Through a prototype deployment of the protocol, we empirically demonstrate its effectiveness in preventing *information exfiltration*. Specifically, we show that the protocol is able to achieve, for legitimate requests, a daily average true accept rate of 99.5% and a maximum false reject rate of 6%, while maintaining a 0% false accept rate for illegitimate requests.
3. Unlike existing protocols that typically base the security of the authenticated channel on the fact that it is human-aided, we introduce the concept of using secure, external hardware modules to realize the security of the authenticated channel. This is also in contrast to the use of internal hardware modules such as Trusted Platform Modules (TPMs) that have been relied upon in other security-related schemes, since their security benefits cannot be realized in our application context.

1.4 Definitions

For purposes of clarity to the reader, we provide some definitions of terms used in this dissertation and the context in which they are used.

Information exfiltration: This refers to the leakage of information, by malware residing on a possibly compromised computer, to an external entity cooperating with the malware.

Destination: This refers to either the domain name or Internet Protocol (IP) address to which a Web request is being sent.

Typed requests: These refer to Web requests that are sent after their destinations have been typed by the user.

Non-typed requests: These refer to Web requests that are sent with no typing activity involved, e.g., requests sent after their destinations have been copied and pasted (e.g., from a document into the browser), selected from a drop-down list, auto-completed by the Web browser, etc.

Legitimate requests: These refer to Web requests that are sanctioned by the user. They consist of both typed and non-typed requests.

Illegitimate requests: These refer to Web requests that are not sanctioned by the user and are presumed to be sent by a malicious entity on the user's computer.

(a', b') **is consistent:** In a two-party communication, when two pieces of information are received by the message recipient, (a', b') is *consistent* if the b that is independently computed by the recipient (using the received a') is the same as the received b'.

Embedded links: These refer to destinations that are contained (embedded) in responses to legitimate outgoing Web requests, e.g., links on a returned Web page.

Enforced user response: This refers to a scenario where a user is compelled to type a destination of a previously rejected outgoing Web request.

1.5 Organization of this Dissertation

In Chapter 2, we present a review of the existing dual channel protocols (both the interactive and non-interactive ones), as well as the current traffic authentication schemes that relate to the work in this dissertation. We then introduce our protocol in Chapter 3 and present a prototype implementation of it in Chapter 4. In Chapter

5, we provide discussions on some aspects of the protocol's implementation such as an alternate placement design of the protocol's second channel, and how optimization of the verifier's workload can be achieved. Finally, we present concluding remarks and possible directions for future work in Chapter 6.

CHAPTER 2

LITERATURE REVIEW

2.1 Dual Channel-based Solutions

The notion of using dual channel protocols arose out of a need to provide secure communication over insecure channels. This is especially desirable in ad hoc networks where Public Key Infrastructure (PKI) is non existent and/or undesirable. The general idea behind these protocols is the transmission of a short authentication bit (or string) over a narrow band authenticated channel in order to authenticate the message transmitted over the wide band insecure channel.

In literature, these protocols are categorized as being either interactive or non interactive, depending on whether the communication between the sender and recipient of information is one-way or two-way. We review some of the existing protocols below.

2.1.1 Interactive Protocols

Rivest and Shamir [37] are believed to have introduced the first interactive protocol in which they proposed using human voices in the authentication process when two parties wish to establish a secure key that is to be used in future communication instances. The idea behind their protocol was that the two communicating parties are able to identify each other's voice; hence, an eavesdropper cannot impersonate any of the parties without being discovered. This protocol is also credited for having introduced the idea of human assistance in the realization of the authenticated channel, a common scenario in dual channel protocols.

In general, subsequently proposed protocols have mainly differed in terms of the security tokens used and adversarial models considered. For instance, Vaudenay [43] proposed using extractable commitment schemes to secure the information sent in their protocol and considered the chosen random string adversarial model. Mashatan and Stinson [28], on the other hand, proposed utilizing Interactive Collision Resistant (ICR) hash functions in their protocol, considered the adaptive chosen plain-text attack model, and also allowed adversaries to have online computational power, a scenario that Vaudenay's protocol [43] did not address.

Example applications of these protocols. As mentioned above, the interactive family of dual channel protocols uses the bi-directional flow of information between communicating parties over two channels [49, 1, 44, 22, 11]. In some authentication systems, the second channel is used to transmit additional authentication factors that are required before authentication can be granted. For instance, in a bid to improve security on many online systems today, it has become common practice to use more than just a user password or Personal Identification Number (PIN) as an authentication factor when granting access to the system. Some of the additional factors typically used to authenticate users on these systems include either a user's biometric signature (e.g., fingerprint, keystroke, iris) or something in the user's possession e.g., a secure token or mobile device that can be used to identify the said user as a legitimate user of the system.

For example, Yang *et al.*'s [47] work focussed on boosting the use of smart cards as a second factor in user authentication. In their work, they proposed new security requirements to improve existing smart card-based two factor authentication schemes. Among their proposed requirements is the ability for users to change passwords without involving the authenticating server and the elimination of a password database at the server side.

The choice of additional authentication factors for most systems usually boils down to the cost associated with implementing the chosen authentication factor. For instance, Aloul *et al.* [1] proposed using mobile phones to generate One Time Passwords (OTPs) that the user can enter into an online system in addition to his or her password or PIN. In their approach, the OTP can be generated either locally on the mobile phone or by requesting it, via SMS message, from a remote server. Their choice of mobile phones is driven by their availability to most users, hence reducing the implementation cost associated with their deployment.

Another practical example using mobile phones is Google's 2-step verification that uses the text or voice channel in a phone to respond to an authentication request received over the Internet channel [17]. Upon receiving an authentication request from a host machine over an Internet channel, Google's remote server sends back a PIN to the associated user's phone that the user (reads or listens to and) sends back to the server using the Internet channel. The server authenticates the user upon receiving the exact PIN that it sent.

In other works [44, 33, 40, 26, 22], mobile devices have been used to send private information securely through untrusted terminals. In these studies, the trusted mobile devices establish connection to the terminals and send encrypted information through the terminals to remote servers, or wirelessly bypass the terminals and send this information directly to the remote servers. In one variation of these protocols [11], the authenticating Web server sends back a cryptographic challenge in the form of a two-dimensional picture to the browser and the user takes a picture of that image using a mobile phone. The user then sends a cryptographic response via wireless to the server. In another variation [22], a trusted proxy intercepts the login requests from untrusted machines to Web servers and asks the users, through their trusted mobile devices, to authenticate those requests before they are sent out; the proxy also removes any information from the responses that the users might not want to reveal to the untrusted machines.

Garriss *et al.* [15] and Dodson *et al.* [11] used a mobile device to determine the level of trust a user should place on the terminal before using its resources. In their work, the terminals were assumed to possess hardware security technologies such as a Trusted Platform Module (TPM) [18] that can be used to determine the trustworthiness of software on the terminal. The mobile device established a connection with the terminal, requested for an attestation of the terminal's software and recommended to the user whether or not the terminal could be trusted.

The common thread among all these works is the requirement for users' involvement in every authentication request, hence rendering them unsuitable for applications requiring frequent authentication.

2.1.2 Non-interactive Protocols

In contrast to the interactive protocols, the non-interactive protocols are designed for only unidirectional flow of information, which keeps them lightweight. This makes them a more desirable option for message authentication because of the reduced operational cost of the protocol. Interactive protocols are therefore usually chosen over the non-interactive ones in cases where they (the interactive protocols) increase the security of the protocol such as when replay attacks on the authenticated channel are considered – a scenario which non-interactive protocols are vulnerable to [28].

Below, we provide only a brief review of the existing non-interactive protocols. A tabular comparison of them, based on shared properties (e.g., security features used, assumed security of manual channel, etc.), is also given in Appendix B. For detailed descriptions of the protocols, we refer the reader to the respective cited works. In the accompanying protocol figures, a "′" is added to information received by Bob (indicating that the information sent by Alice may or may not be the same as the information that is received by Bob). The insecure and authenticated channels are represented using "$\rightarrow$" and "$\Rightarrow$", respectively.

Balfanz *et al.* (BSSW'02 in Figure 2.1) [2] proposed the first such protocol in which a message M and its hash h are sent over the insecure and authenticated channels, respectively. In their protocol, they compute h from M using a collision resistant hash function. However, their protocol was seen to be vulnerable to offline attacks (e.g., the birthday attack).

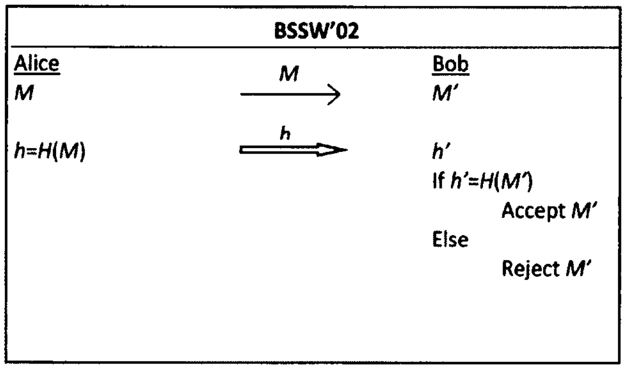


Figure 2.1: Balfanz *et al.*'s protocol.

To address this vulnerability, Gehrmann *et al.* (GMN'04 in Figure 2.2) [16] proposed the MANA 1 protocol. In the MANA 1 protocol, the hash of a message is computed by applying a random key to a universal hash function. The hash and the key are then sent over the authenticated channel and the message is sent over the insecure channel. This protocol assumes confidentiality of the authenticated channel. Further improvements to the MANA 1 protocol, that make different assumptions over the authenticated channel, have also been proposed in [30, 31].

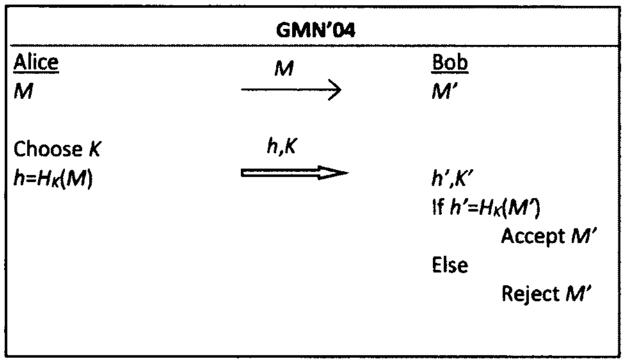


Figure 2.2: Gehrmann *et al.*'s protocol.

Pasini and Vaudenay's protocol (PV'06 in Figure 2.3) [34] assumed an authenticated channel in which an adversary is only restricted from modifying the messages. The security of their protocol is guaranteed by using trapdoor commitment schemes and second preimage resistant hash functions. A commitment is applied to the message and a key to produce commit "*c*" and decommit "*d*" values. The hash of "*c*" is transmitted through the authenticated channel and "*c*" and "*d*" are sent over the insecure channel.

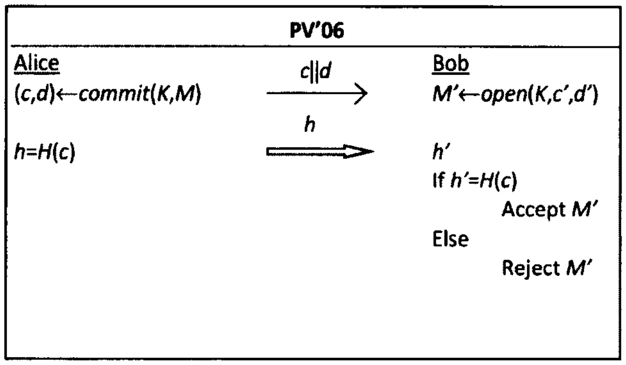


Figure 2.3: Pasini and Vaudenay's protocol.

Mashatan and Stinson (MS'06 in Figure 2.4) [27] and Reyhanitabar *et al.* (RWN'07 in Figure 2.5) [36] proposed two protocols that are somewhat similar to each other. Both protocols send the message together with a randomly chosen key over the insecure channel and a hash of that message computed utilizing the key over the authenticated channel. However, while [27] utilizes a hybrid collision resistant hash, [36] uses an enhanced target collision resistant hash.

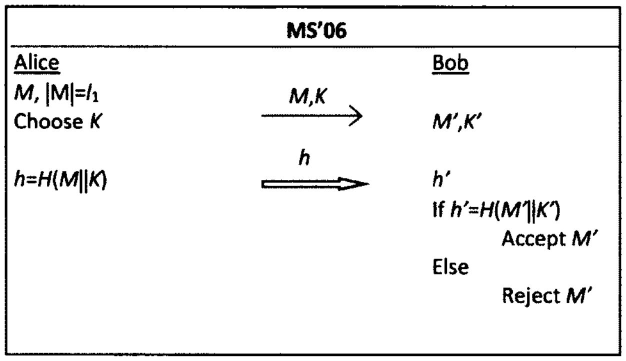


Figure 2.4: Mashatan and Stinson's protocol.

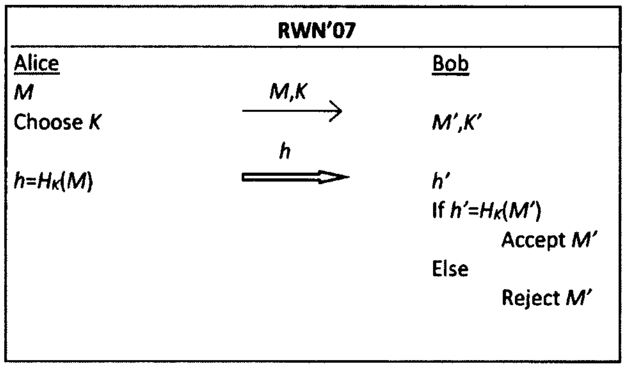


Figure 2.5: Reyhanitabar *et al.*'s protocol.

In order for these protocols to be applicable to scenarios where frequent use of the authenticated channel is necessary, the manual authenticated channel in these protocols needs to be replaced by a non-manual channel. However, even with this modification, remotely located verifiers become prone to a channel spoof attack, as explained below, defeating the protocols' purpose.

Adaptation of existing non-interactive dual channel protocols for frequent authentication. Since it is impractical for a human assisted channel to engage continuously in the authentication process, the manual channel in the existing non-interactive dual channel protocols needs to be replaced with a non-manual channel to facilitate continuous authentication by a remote verifier. Figure 2.6 presents a generic representation of such an adaptation. Bob in this figure represents a remote verifier that authenticates Alice. Alice produces two sets of information, r_1 and r_2, and transmits them respectively over the insecure and authenticated non-manual channels. Bob perceives receiving r'_1 from $source_1$ and r'_2 from $source_2$. Upon receiving r'_1 and r'_2, Bob determines if they are *consistent*, i.e., if r'_2 authenticates r'_1.

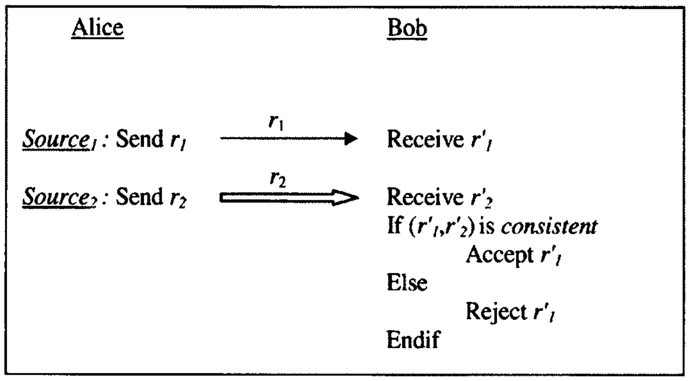


Figure 2.6: A possible adaptation of existing non-interactive dual channel protocols for frequent authentication.

This adaptation, however, opens up the possibility of a channel spoof attack; an adversary, Eve, residing at $source_1$, can compute her own r_1 and r_2, transmit r_1 from $source_1$, and spoof the identity of $source_2$ to make it appear to Bob that r_2 is transmitted from $source_2$, thus defeating the scheme. To avoid this attack, Bob must have a way to know if an r_2 is really generated by Alice, which a simple replacement of the manual channel with a non-manual one cannot ensure.

2.2 Existing Traffic Authentication Schemes

One of the most prevalent security threats is the exfiltration of sensitive information (e.g., passwords, social security numbers, etc.) by malicious programs or entities. A recently reported example of this is the *Heartbleed* bug [6] that exposed a security flaw in the OpenSSL protocol [32] allowing would be attackers to access data (such as usernames and passwords, content of emails, instant messages, etc.) from the memory of servers using this protocol.

Though this bug was not reported to be a deliberate attack on Internet communication, it nonetheless shows the vulnerability of online information to dedicated, unscrupulous individuals. In this work, we specifically address the scenarios where attackers deliberately target their victims' information, and by exploiting compromises on the victims' computers (e.g., through drive-by downloads), they are able to steal and exfiltrate sensitive information to external malicious entities under the control of the attackers.

To prevent this malicious exfiltration of users' information, a number of schemes have been proposed to authenticate traffic being sent out of users' computers. We categorize the existing approaches as: (i) statistical testing based methods [42, 4] in which observed statistical properties of both malicious and benign traffic (e.g., new connection establishment rates, packet sizes, upload/download bandwidth) are used to train classifiers for future classification of unknown traffic; (ii) keystroke/mouse-click association based methods [19] that rely on keystroke/mouse activity to categorize traffic that is sent within a defined interval of such activity as being legitimate (benign) and all other traffic is categorized as being malicious; (iii) packet marking based methods [46] that introduce check points at different levels of the network protocol stack to mark and verify traffic (for the markings) as the traffic goes through the stack. The assumption is that malicious traffic does not go through the entire stack, and hence it will not be marked; (iv) heuristic rule based filtering (e.g., firewalls) [20] that employ a rule set (e.g., based on port numbers) that is used to distinguish between benign and malicious traffic depending on whether the traffic meets the specified rules; (v) blacklist based egress filtering [10, 38] that maintain a set of known malicious traffic characteristics (e.g., IP addresses/domain names) and block traffic whose characteristics match those in the set; and (vi) content sensitivity based filtering [12] that checks the content of data being sent to determine its sensitivity. The assumption made here is that the data is structured, and hence, its sensitivity can be determined based on its content.

However, even with these approaches being deployed, malware is still able to exfiltrate information due to certain limitations that the above approaches face e.g., the first four are vulnerable to malware's adaptation to the associated filtering logics, the fifth one suffers from incomplete coverage of malicious recipients, and the sixth one fails to identify and prevent unstructured sensitive data from exfiltration.

Of the above approaches, the two that are most comparable to our work are [19] and [46]. The difference between these two approaches and ours is that malware's adaptation to our scheme is unlikely to occur since authentication decisions are made based on typing activity on the keyboard in which host-based malware cannot engage. Additionally, unlike [46] that assume limited kernel malware, our solution does not impose any restriction on malware in the host.

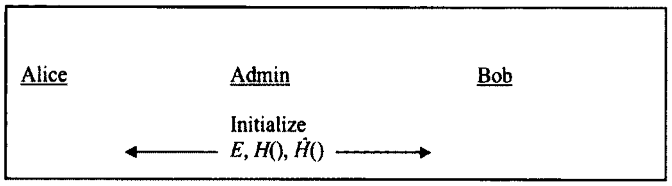
CHAPTER 3

THE PROPOSED PROTOCOL

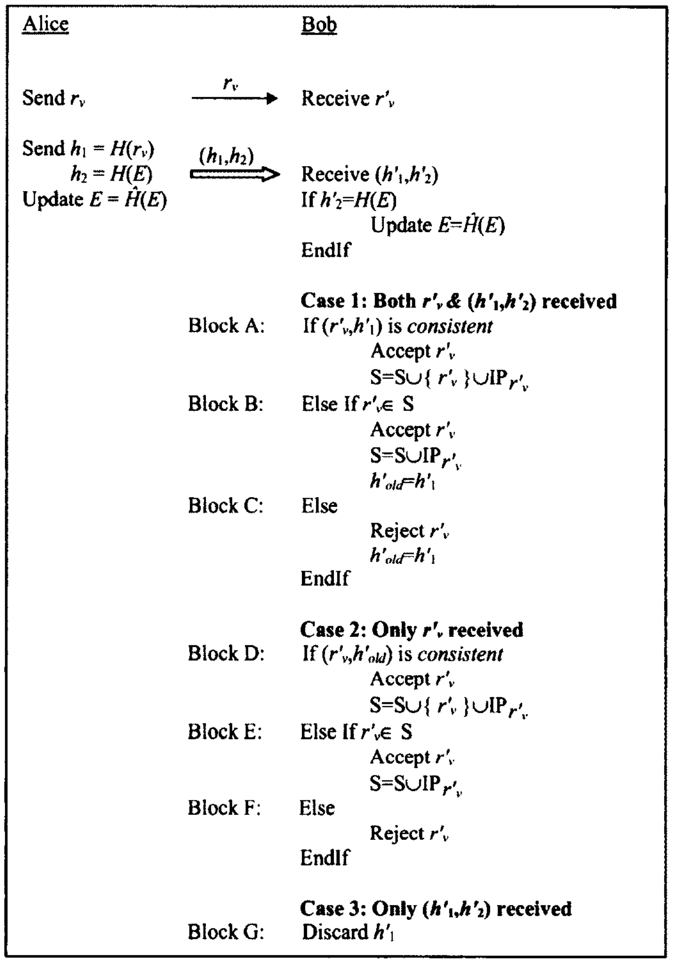
3.1 The Protocol

This dissertation presents, in Figure 3.1, a non-interactive dual channel protocol [21] that resolves the issue of a channel spoof attack, mentioned in Chapter 2, by utilizing a synchronization variable, E which, unlike [16, 27, 36], is not randomly chosen, but generated by a function $\hat{H}$ instead. In a practical deployment of the protocol, a system administrator initializes E to both Alice and Bob. Alice and Bob, thereafter, individually computes the successive values of E using $\hat{H}$. Since for a specific instance of an authentication request Bob is aware of the expected E, Eve can no longer launch a channel spoof attack without knowing which E Bob is expecting. For ease of reference, we present the notations used in the protocol and their meanings in Table 3.1.

Alice in this protocol can be realized as a keystroke parsing program participating in the protocol on behalf of the user and Bob as a verifier program located in a remote computer. Alice parses the keystrokes to extract r_v, an IP/domain name that has been typed and computes $h_1 = H(r_v)$ and $h_2 = H(E)$ using a standard hash function H. Alice releases r_v to the host using the insecure channel and transmits (h_1, h_2) to Bob using the authenticated channel. Not all the typed IP/domain names



(a)



(b)

Figure 3.1: The proposed non-interactive dual channel protocol.

are intended to send a request (e.g., a user may type an IP/domain name while writing a report). If a user intends to send a request to r_v, the host is supposed to forward that request to r_v via the insecure channel.

Table 3.1: Symbols used in the protocol and their meanings.

Symbol	**Meaning**
r_v	request sent to the verifier (domain name or IP address)
E	the synchronization variable used in the protocol
h_1	the hash value of r_v
h_2	the hash value of E
H	a standard hash function applied to r_v to produce h_1 and to E to produce h_2
$\hat{H}$	update function used to produce a new E
IP_{r_v}	a list of IP addresses returned in response to an accepted r_v
S	a set of all accepted r_vs and IP_{r_v}s
h_{old}	an unused and saved value of h_1
x'	Bob's received value of x
$\rightarrow$	insecure channel
$\Rightarrow$	authenticated channel

For this protocol, we assume that a malicious program residing in a host cannot remove, modify or stall information transmitted over the authenticated channel that bypasses the host. The protocol also assumes the following: (i) an h_1 can verify the associated r_v; (ii) an adversary (Eve) can generate and replace the r_v, but it is difficult for her to compute the associated h_1 from a given r_v; (iii) Bob can compute h_1' from r_v'. An r_v' is accepted only if (r_v', h_1') is *consistent*, i.e., the hash Bob computes for the r_v' is the same as h_1'. The protocol works in two phases: (i) the initialization phase (Figure 3.1(a)) and, (ii) the protocol phase (Figure 3.1(b)) as described below.

(i) Initialization phase

The initialization phase is a one step setup process required prior to the establishment of the rest of the protocol. During this phase, a human operator assigns the hash function H, and a pair of secret E and update function $\hat{H}$ to Alice and Bob. H is used to compute h_1 from r_v and function $\hat{H}$ is used to update and synchronize E by both Alice and Bob. Use of $\hat{H}$ prevents an adversary from computing subsequent Es in the unlikely event of randomly guessing a right E. If, for instance, due to some failures E is no longer synchronized, all authentication requests are rejected and E needs to be re-initialized.

(ii) Protocol phase

In the protocol phase, Alice transmits r_v via the insecure channel and (h_1, h_2) via the authenticated channel to Bob. Alice updates her E, using the function $\hat{H}$, after sending (h_1, h_2) to Bob. If $h_2' = H(E)$, Bob assumes that h_2' is computed by Alice and updates his E to have the same value as that of Alice. Bob maintains a set S of accepted IPs and domain names. The set of IPs returned by Domain Name Servers (DNS) for an accepted domain name r_v', denoted as $IP_{r_v'}$, are also included in S (we assume that the DNS are not compromised). Updating S for an r_v' or a set $IP_{r_v'}$ refers to the inclusion of r_v' or elements in $IP_{r_v'}$ to S, if they do not already belong to S. While checking if a given domain name matches with a domain name in S, Bob compares up to the third-level for country code domains, and up to the second-level for other domains.

3.1.1 Protocol Scenarios and Operation

In the three cases presented below, we discuss, in detail, the scenarios that are likely to arise during the operation of the protocol and their respective resolution. Figures 3.2 and 3.3, respectively, provide a summary of these scenarios and their resolution. The correspondence between a case resolution branch in Figure 3.3 and a leaf scenario in Figure 3.2 is represented by the type of their border lines.

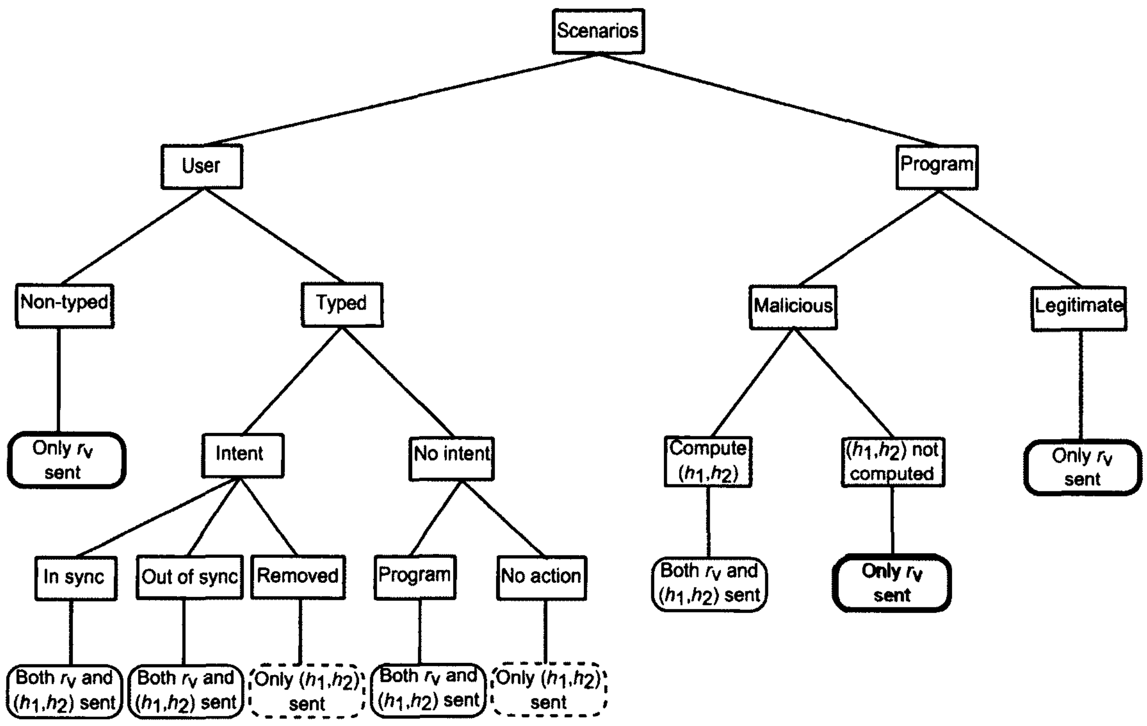


Figure 3.2: Protocol scenario tree showing the different types of outgoing requests that are likely to be encountered during a protocol run and information sent to the verifier in each case.

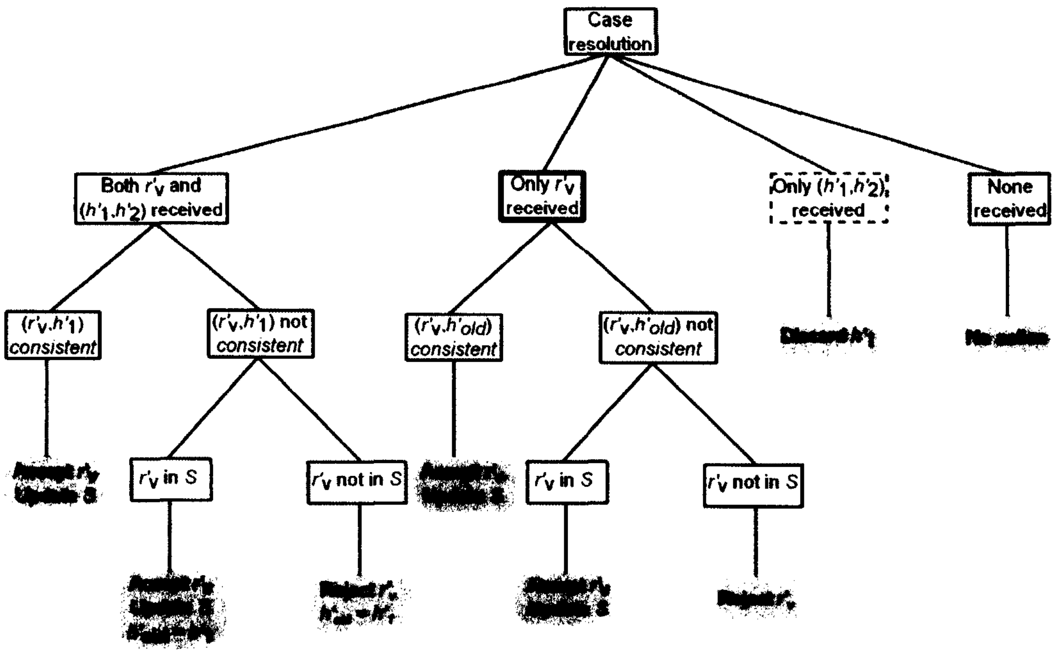

Case resolution
Both r'_v and (h'_1,h'_2) received
Only r'_v received
Only (h'_1,h'_2) received
None received
(r'_v,h'_1) consistent
(r'_v,h'_1) not consistent
(r'_v,h'_{old}) consistent
(r'_v,h'_{old}) not consistent
r'_v in S
r'_v not in S
r'_v in S
r'_v not in S

Figure 3.3: Protocol operation tree showing how the verifier implements the protocol to accept or reject an outbound request based on the information received during a given protocol run.

Case 1: r_v' is received through the insecure channel and (h_1', h_2') is received through the authenticated channel. All the possible scenarios for which this may occur include: (i) a user sends an authentication request by typing on the keyboard leading Alice to transmit (h_1, h_2), and the host to forward an authentication request to the associated r_v; (ii) a user types a domain name or IP without intending to send a request leading Alice to transmit (h_1, h_2), but the host does not forward any authentication request; however, a legitimate program or a malicious program residing in the host sends an authentication request at the same time to an r_v; (iii) a user sends an authentication request but an r_v from a legitimate or a malicious program reaches Bob before the r_v from the user, causing Bob to receive a wrong r_v', (h_1', h_2') pair; (iv) a malicious program computes (h_1, h_2) for her own r_v utilizing a random E and transmits them both through the insecure channel, but she makes it appear that (h_1, h_2) is transmitted through the authenticated channel by spoofing the source IP; and (v) a malicious program replaces the r_v belonging to a user or a legitimate program's request with another r_v.

For scenario (i), the condition in Block A (see protocol diagram, Figure 3.1) is satisfied, the r_v' is accepted and Bob updates S for r_v' and $IP_{r_v'}$, the associated IPs returned by a DNS when r_v' is a domain name.

For scenarios (ii) and (iii), either the condition in Block B is satisfied, or r_v' is rejected in Block C; depending on whether the r_v' is already included in S. Bob saves the h_1' as h_{old}' anticipating the arrival of an associated r_v' due to scenario (iii). If the condition in Block B is satisfied, Bob accepts r_v' and updates S for $IP_{r_v'}$. It may, however, happen that the r_v' that Bob receives from a legitimate or malicious program incidentally matches with the r_v sent for a user's request. In that case, the condition in Block A is satisfied, Bob accepts r_v' and updates S.

For scenarios (iv) and (v), either Block C or the condition in Block B applies depending on whether $r_v' \in S$ or not. Block C rejects r_v'. However, if Eve chooses to use an $r_v \in S$, the r_v' is accepted, and S as well as h_{old}' is updated in Block B. We note that the update of S and h_{old}' does not affect the protocol's operation as explained below.

Acceptance of an r_v' received from a malicious program because it matches the r_v a user sent a request to, or because $r_v' \in S$ does not constitute exfiltration to Eve's choice of destinations because of the assumption that, for the purpose of exfiltration, the adversary's choice of destinations do not include any legitimate destinations.

Case 2: Only r_v' is received through the insecure channel. This is possible in the following scenarios: (i) a user sends an authentication request, but an r_v from a legitimate or a malicious program reaches Bob before the r_v from the user. This causes Bob to pair the user's (h_1, h_2) with the legitimate or malicious program's r_v and subsequently receiving the user's r_v without the corresponding (h_1, h_2); (ii) a user sends a non-typed request to a destination; (iii) a legitimate or a malicious program sends an authentication request; (iv) a malicious program replaces the r_v belonging to a user's non-typed request or a legitimate program's request with another r_v.

For scenario (i), if corresponding h_1' is preserved as h_{old}', the condition in Block D is satisfied, the r_v' is accepted and Bob updates S for r_v' and $IP_{r_v'}$. Otherwise (i.e., if Bob earlier received the same r_v' from another source satisfying the condition in

Block A and thus not preserving h_1', however, updating S for r_v'), the condition in Block E is satisfied, r_v' is accepted and S is updated.

For scenarios (ii), (iii) and (iv), either the condition in Block E is satisfied, or r_v' is rejected in Block F, depending on whether the r_v' is already included in S. If the condition in Block E is satisfied, Bob accepts r_v' and updates S for $IP_{r_v'}$. As explained in Case 1, the acceptance of an r_v' received from a malicious program due to $r_v' \in S$, does not constitute exfiltration.

Case 3: Only (h_1', h_2') is received through the authenticated channel. This case is possible when Bob receives only (h_1', h_2') due to the following scenarios: (i) a user types a domain name or IP without intending to send a request, leading Alice to transmit (h_1, h_2), but the host does not forward any authentication request; (ii) a malicious program residing in the host removes the r_v for a typed request sent by the user; (iii) a malicious program residing in the host randomly generates (h_1, h_2) and transmits them to Bob (making it appear that they are transmitted from Alice), hoping that $h_2 = H(E)$, which would break the synchronization of E between Alice and Bob. In all of these scenarios, Bob receives no r_v' from any sources; Block G applies to all of them and Bob discards the h_1'.

Frequent disruptions of communications due to the replacement or removal of r_v by a malicious program (Case 1, scenario (v); Case 2, scenario (iv); Case 3, scenario (ii)) will expose the presence of that malware, which defeats the malware's objective of operating stealthily, and thus malware is unlikely to engage in such activity under this protocol.

3.1.2 Security Analysis of the Protocol

Given the presented protocol, a malicious destination can only be authenticated if the malware on the host is able to make Bob think that the request sent to that destination was actually sanctioned by the user. We consider that a malicious agent Eve, residing in the host, can transmit her choice of an r_{eve} through the insecure channel and make Bob accept it in the following cases:

Case 1: Spoof attack. As mentioned, our protocol resolves the issue of channel spoof attacks by using the variable E that is maintained separately by Alice and Bob and simultaneously updated by them. Both the Es are initialized with the same value before the protocol is deployed and their subsequent values, at a given instance, are computed independently by Alice and Bob using $\hat{H}$. Since Bob is aware of the expected value of E that Alice is using to compute h_2 for a specific instance of an authentication request, an adversary (Eve), must be able to compute the correct $H(E)$ to make Bob believe that she is transmitting (h_1, h_2) through the authenticated channel, and not just spoofing the identity of Alice.

An attempt at such an attack, therefore, would require Eve to compute h_2 using a randomly chosen E' so that $H(E) = H(E')$ where E and E' may or may not be the same i.e., E' is a preimage or a second preimage of E. To prevent this attack, the hash function, $H()$, is required to be resistant to preimage and second preimage attacks. Since the value of $H(E)$ is not known to Eve, she would launch a one-shot attack and hope for success. The probability of Eve launching a successful spoof attack is $\epsilon_s + \epsilon_p$, where ϵ_s and ϵ_p are the probabilities of successful second preimage and preimage one-shot attacks on $H()$, respectively.

Case 2: Message replacement attack. In this attack, Eve does not compute her own (h_1, h_2). Instead, she replaces Alice's r_v with her own r_{eve} and is successful if r_{eve} is a second preimage, i.e., $H(r_v) = H(r_{eve})$. A second preimage resistant hash function, $H()$ is necessary to prevent such an attack. If Eve has a set of destinations, $\mathbf{d_{eve}}$ from which she attempts to find the r_{eve}, the probability of launching a message replacement attack using an element in $\mathbf{d_{eve}}$ is $1-[1-\epsilon_s]^L$ where ϵ_s is the probability of a successful second preimage one-shot attack on $H()$ and L = $|\mathbf{d_{eve}}|$. If Eve does not check for $H(r_v) = H(r_{eve})$, but simply replaces r_v with a randomly chosen r_{eve}, then the probability of Eve successfully launching a message replacement attack is ϵ_s.

3.2 Protocol Extensions

3.2.1 Protocol Extension 1: Embedded Links

A response to a legitimate request may contain links not listed in S. Although those are returned by a trusted destination, it is not unlikely that some of them may be injected by an adversary and are malicious in nature. Because of this uncertainty, we propose that requests to those linked destinations be accepted for a temporary period as well as the amount of data allowed to be transmitted to those destinations be restricted. An administrator can define these parameters to meet an organization's security needs. Decision on embedded links mostly affects Web-browsing experience. We notice that the majority of these links require only simple connection requests. By restricting the amount of data allowed to be transmitted by the amount that an average connection request requires, we can minimize the "enforced user response" and prevent any significant exfiltration.

Considering this scenario, information can be exfiltrated if the embedded links are malicious. However, given that their acceptance is temporary, exfiltration can only occur during their temporary acceptance period. We can quantify the maximum amount of *information exfiltration* that is expected during this time as $n_h {m_t}^d s$ where n_h is the number of requests that are typed by the user, ${m_t}^d$ is the number of embedded links that are accepted either during time, t or until a certain number, d of successive embedded links is reached, and s is the packet size (in bytes) of each request. These parameters can be adjusted by the system administrator to achieve the desired trade-off between user-experience and acceptable amount of *information exfiltration.*

3.2.2 Protocol Extension 2: Final Destination Not Explicit

The protocol presented in Figure 3.1 can be implemented locally on a single host computer or in a subnet consisting of many computers. However, when the final destination of the outgoing request is not explicitly stated in the request, other entities such as mail servers (for the case of email) need to be integrated into the process of authenticating these requests. The second extension to the protocol, though not locally implementable, is proposed to address such a scenario where malware exfiltrates information to its cooperating entity through legitimate third party services as discussed below.

Adversaries may attempt to exfiltrate information by using legitimate third party services (e.g., webmails). To prevent this, we propose a Bloom filter based extension that requires participation of the associated third party servers. The proposed extension requires that if Alice notices a recipient ID v (e.g., xyz@gmail.com) for a legitimate server r_v (e.g., gmail.com), she sends (h_1, h_2, v, r_v) to Bob. Bob then creates a Bloom filter B_{r_v}, maintaining the records for all associated v's he receives for r_v. A Bloom filter is composed of an m-bit array and a set of w different hash functions [3]. The array entries are initialized to zero. Upon receiving a v, all w hashes of v are computed and the array locations corresponding to the computed hashes are set to 1. If a traffic request is sent to r_v, Bob constructs a new request to r_v, concatenating the original request and B_{r_v}. The associated third party server, upon receiving this concatenated request, verifies if the intended recipients of the associated request have corresponding entries in B_{r_v} before it delivers the message to those recipients. A recipient has an entry in B_{r_v} if the associated array has a "1" in all the positions associated with the w hashes of its ID. A "0" in any of those positions indicates its absence in the filter, although all "1" entries do not guarantee its presence. If we assume that a hash function selects each array position with equal probability and the probabilities of each bit being set are independent, the asymptotic formula, $(1 - e^{-w(n+0.5)/(m-1)})^w$ [13], describes the expected false positive rate for a finite bloom filter with n elements. The advantage of using a Bloom filter is that its size does not grow with the number of entries of the recipients.

CHAPTER 4

A PROTOTYPE IMPLEMENTATION OF THE PROTOCOL

Having developed the protocol, we went ahead to test the suitability of this protocol for a real world network traffic scenario. This chapter details how a prototype of this protocol was developed and deployed. Specifically, the threat model that was considered while deploying the protocol is given, then the trusted external hardware device that was used in the protocol as well as its comparison with internal hardware devices on the market today is presented, and finally the experimental setup that was used and results that were obtained are described.

4.1 Threat Model

The threat model considered in this dissertation assumes that the computers under surveillance connect to the Internet via a secure proxy that hosts traffic filters, which is a common scenario in many organizations. As shown in Figure 4.1, the threat model considers that malware infecting a host is capable of exfiltrating information from that host to an external malicious entity, defeating the known traffic filtering mechanisms. It further considers that exfiltrated information can be transmitted to the external malicious entity in one of two ways: (i) using the malicious entity's domain name or IP address as the destination, (ii) transferring information to a legitimate third party service provider, such as a webmail server, that the malicious entity has access to. In addition, the threat model assumes that malware residing in a host cannot remove, modify or stall the content on a traffic channel that does not run through the host.

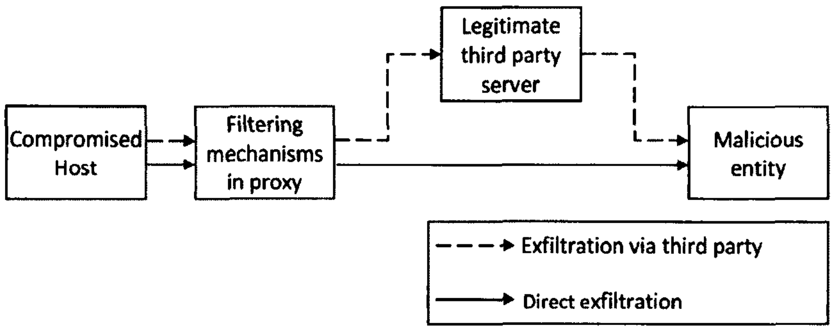

Legitimate third party server
Compromised Host
Filtering mechanisms in proxy
Malicious entity
Exfiltration via third party
Direct exfiltration

Figure 4.1: A diagram of our threat model showing that host based malware can exfiltrate information to a malicious entity either directly or through a legitimate third party server, bypassing existing filtering mechanisms.

4.2 Trusted Hardware

4.2.1 Trusted Platform Modules

The use of Trusted Platform Modules (TPMs) in security solutions has gained traction in recent years because of their ability to satisfy security properties that are difficult to realize through software-only solutions. In addition, their low price and miniature size made them suitable as add-on devices. A TPM is a microcontroller chip that is attached internally to a computer motherboard for the purpose of securely generating and storing cryptographic keys and storing encrypted data either locally or in the hard drive [18]. A schematic representation of the components of a TPM is shown in Figure 4.2.

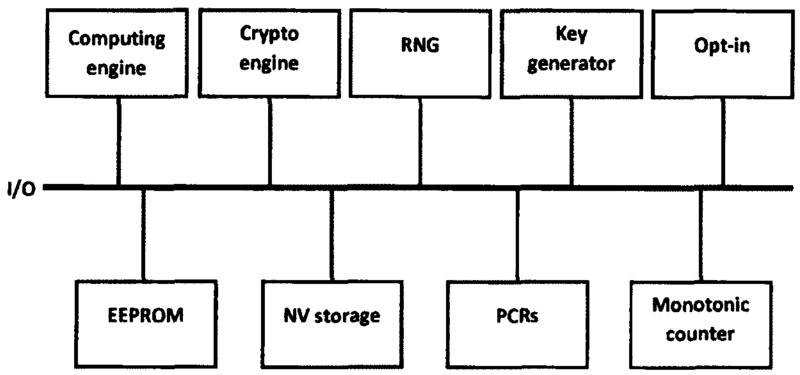


Figure 4.2: A diagrammatic illustration of the TPM's components.

TPMs can ensure a trusted boot procedure by facilitating each component in the boot sequence to attest to the integrity of the next component and thus establish trust on the OS kernel. Such trust can be extended beyond the kernel to applications by attesting the target applications while loading, and using a similar chain of attestations [45]. However, if an application is pre-infected, it would still be attested by this process and be considered as trusted. Additionally, for an application to be trusted, it cannot be loaded whenever it is required; rather, it has to be loaded during the boot process.

A TPM comes with a set of built-in cryptographic key and hash generation functionalities that are operated using a predefined set of commands (or instructions), hard-coded into its execution engine. This prevents adversaries from exploiting the TPM to perform unknown operations since the TPM is accessible from the host, which could be compromised. However, this also limits the TPM's programming flexibility since no new instructions can be added to this set of instructions after manufacturing.

4.2.2 Raspberry Pi

To address the above limitations and to satisfy lightweight operational needs required by continuous authentication applications, we build our solution around a conceptual external trusted hardware module that we realize using a Raspberry Pi (shown in Figure 4.3) as a prototype. The proposed hardware module requires fewer security assumptions than a TPM due to its isolation from the host. It is also programmable, allowing implementation of given logic.

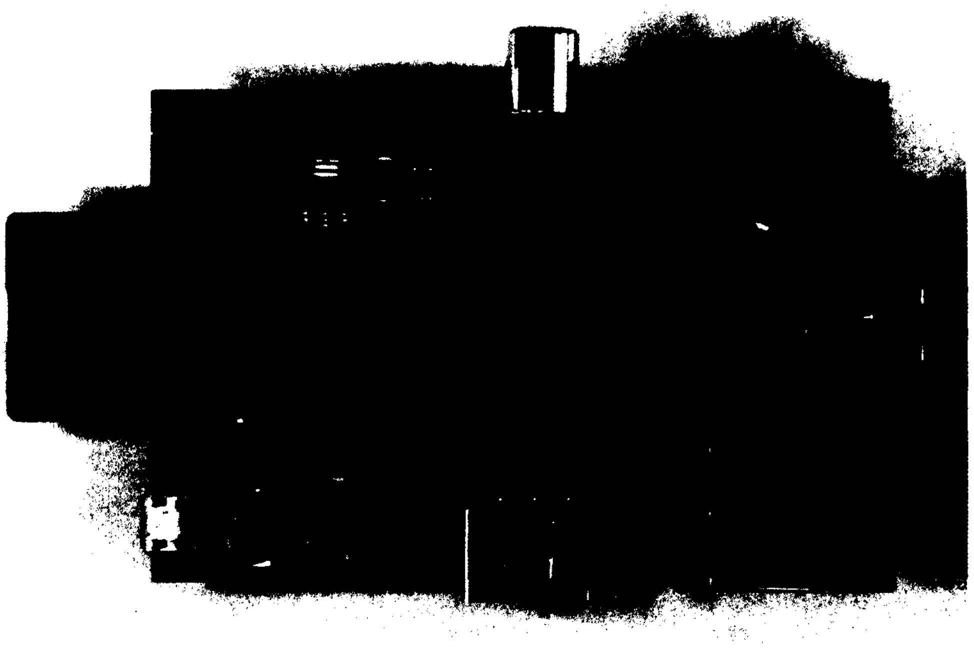


Figure 4.3: A Raspberry Pi

The Raspberry Pi Model B is a miniaturized computing device with a RISC processor based on a 32-bit ARM architecture. It has two USB interfaces and one 10/100 wired Ethernet port that can facilitate networking functionality. It uses a removable Secure Digital (SD) card as the default secondary storage device to store the OS (such as Debian "Wheezy" Linux) and other necessary files. In a conceptual design of a hardware module to be used in this protocol, the storage would be integrated into the module so that it is not readily physically accessible. Write access to this storage would be facilitated via a USB port and restricted through administrative control. The Raspberry Pi is powered through a 5V micro USB.

A typical solution to support the proposed continuous traffic authentication protocol using a TPM would require: (i) attestation of the source of the keystrokes to make sure that the keystrokes are generated by the keyboard driver and not by an unauthorized application [45, 46], (ii) interactive protocol operation to set up a shared key between the client and the remote server e.g., through RSA key exchange protocol [46], and (iii) signing of keystroke events [46]. Steps (i) and (iii) in the above process would be required for each authentication request making it highly impractical for frequent traffic authentication. The proposed external hardware module eliminates the requirement of steps (i) and (ii) since it is connected externally beyond the reach of the malware residing in the host and requires the use of cryptographic hashes to realize the desired security properties.

Additionally, some of the Raspberry Pi's functionalities such as audio, RCA, HDMI and GPIO ports are not required by the proposed design; hence, this should translate to a lower cost realization of the conceptual external trusted hardware device than realizing it using a Raspberry Pi.

4.3 Setup

As shown in Figure 4.4, we designed a prototype implementation of our protocol, realizing the dual channels through the keyboard's attachment to a Raspberry

Pi (Alice). The insecure channel ran through the host to a verifier (Bob) while the authenticated channel was directly connected to the verifier, bypassing the host. Regular communication between the host computer and the Internet was channeled through a proxy in which the verifier was located.

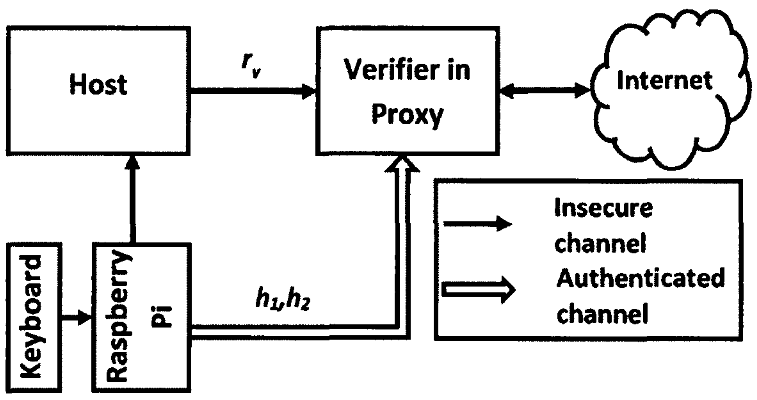


Figure 4.4: A prototype of the proposed protocol showing how a single computer's keyboard (aided by a Raspberry Pi) is connected to both the host computer and the remote verifier through the insecure and authenticated channels, respectively.

In a regular setup, once a user sends an outgoing request by typing or copying and pasting a domain name or IP address (r_v) into a Web browser, a Web request is generated and sent from the host via the proxy to the request's Web server. A response from the Web server, containing the requested information is sent to the proxy, which then forwards the same to the requesting host. Additional processing or filtering of outgoing and/or incoming requests can be done at the proxy if desired. Our scheme modified this setup by capturing the r_v, using the keystroke parsing program, before it was sent through the host. A hashed copy of the r_v was then sent directly to the verifier through the authenticated channel. The verifier in the proxy inspected all outgoing requests it received from the host before forwarding them to the Internet to ensure that a corresponding r_v had also been sent to it from the keystroke parsing program. A detailed description of the protocol operation including the procedure for handling requests that were not typed is given in Chapter 3.

Based on the prototype, we set up a small subnet consisting of a Windows XP and three Windows 7 machines in a graduate student laboratory that seats ten students. The choice of operating systems for the four machines used was arbitrary and has no effect on the experiments or results obtained. The Windows XP machine served as a dedicated proxy (verification server) hosting the remote verifier. The three Windows 7 machines were assigned to three of the students in the lab for daily usage. However, other students in the lab also routinely used these computers during the experimentation period.

Since all HTTP/HTTPS and DNS traffic was channeled to the external world (i.e., Internet) through the proxy, we implemented our protocol logic for traffic authentication as follows. We developed a verification program that was deployed in the user space of the proxy (see Figure 4.5).

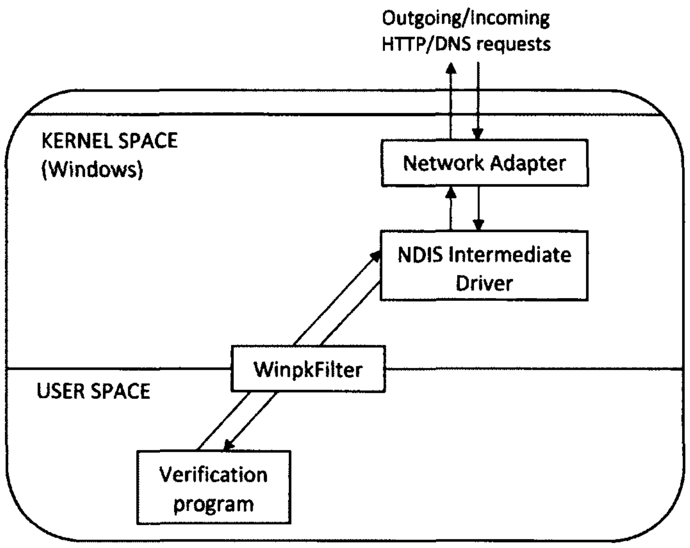


Figure 4.5: A windows-based implementation of the verification server.

We used the WinpkFilter library to set up an interface between the user space verification program and the NDIS Intermediate Driver such that all outgoing traffic received from the three Windows 7 client machines was sent to the verification program before being forwarded to the respective destination(s) if found to be authorized.

In the proxy, we also saved the outgoing packets' session information (i.e., destination IP and port) for later comparison with incoming packets. Outgoing packets for which no (h_1, h_2) was received (via the authenticated channel) were checked to see if the destinations belonged to the set S before their destinations could be authorized. If this were true, the associated destinations were authorized. Otherwise, they were marked as being unauthorized and rejected.

The respective destination servers, when contacted, responded with the appropriate reply messages. At the proxy, the incoming packets' session information was compared with previously stored session information from the outgoing packets before the legitimate responses were sent back to the appropriate requesting client machines. Any response packets that were received without a legitimate request (i.e., no session information recorded) were rejected.

As shown in Figure 4.6, a similar setup can also be achieved in a Linux environment by setting rules through IP tables for the built-in firewall, NetFilter, instructing it to send all packets (both incoming and outgoing) to the user space. Using the Libipq library, the user space verification program would receive packets from the kernel space and inspect them for their authenticity before sending the packets with verified recipient information back to the NetFilter for release to the network.

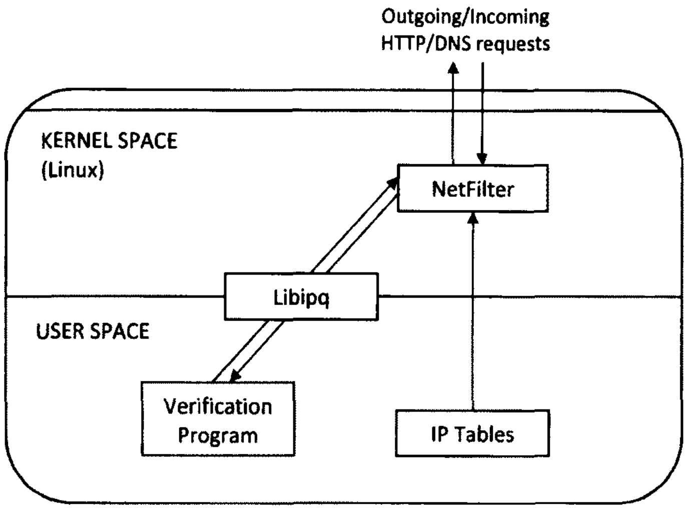


Figure 4.6: A possible linux-based implementation of the verification server.

The keystroke parsing program (on the Raspberry Pi) was used to scan keystrokes as a user typed, identify domain names and IP addresses, compute hashes (h_1, h_2) and release the hashes through the authenticated channel; the normal flow of the keystrokes was released to the insecure channel. A subnet router, upon receiving any traffic through either of the channels, forwarded this traffic to the proxy. The verifier on the proxy continuously monitored the outgoing traffic and extracted the destination domain names and IPs (r_v). It then followed the protocol described in Chapter 3 while granting authentication.

We maintained a shared IP/domain name set S in the verification server for the traffic requests from all three host computers so that prior validations of a request from one host was effective for the other hosts, too. When checking if a given domain name matched with a domain name in S, we compared up to the third-level for country code domains, and up to the second-level for other domains. We also implemented *protocol extension 1* to allow the embedded links only to establish connections. However, we skipped *protocol extension 2* since it required participation of the third party external servers. Over a 90-day period, all the outgoing traffic requests from the Windows 7 machines were stored in the verifier and the collected data during this time was analyzed to investigate the performance of the proposed protocol.

4.4 Experimental Results

To analyze the collected data and measure the protocol's performance, we undertook two studies. In the first study, we measured the general performance statistics when using the protocol such as how many requests were sent, composition of these requests, acceptance (or denial) rates of the requests, among others. In the second study, we measured protocol performance from the users' perspective, i.e., how convenient (or inconvenient) it was for the users to utilize this protocol. We present these two studies and the associated results below.

4.4.1 Accuracy

In this study, our objective was to recognize the outgoing network requests that were sanctioned by the users (through users' activity such as typing, clicking, copying and pasting, etc.) or by non-malicious programs on the users' computers (e.g., legitimate software's auto update programs) as legitimate requests (i.e., accept them) and deny all other unsanctioned (illegitimate) requests.

We computed true accept rates as the ratio of the accepted requests to the total number of requests that had been typed at least once and hence had an associated entry in the set S of accepted r_vs and IP_{r_v}s. We then computed false accept rates by either checking whether any of the embedded links had appeared in databases (e.g., DNSBL [10], SBL [38]) of known malicious sites or by manually inspecting if their

IPs resolved to unverifiable or suspicious entities through *Whois Lookup*. Based on our protocol, false accepts are only likely to occur if S is compromised or some of the embedded links are malicious. Since our threat model assumes that the verification server is secure, we assume S to be secure too, and hence, our test for false accepts only consisted of examining the latter.

We computed true reject rates from those requests which were denied and either had no associated entry in S (implying that they were never typed by a user during our experimentation period), or whose IPs resolved to unverifiable or suspicious entities through manual inspection.

We computed false reject rates from, (i) requests that were initially denied (because their destinations had not been previously typed) and were later accepted after having been typed by the user, meaning that they later had an associated entry in S, and (ii) requests that were denied whose IPs resolved to legitimate entities through manual inspection. Requests in (ii) included programmatic requests from legitimate software and rejected requests that users did not re-attempt to send by typing their respective destinations.

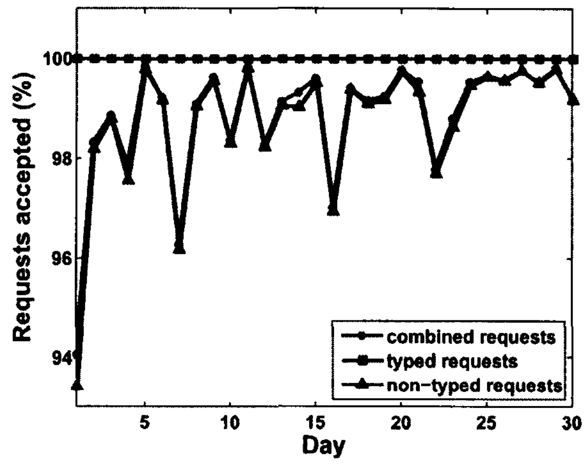
We observed that all illegitimate requests sent during the experimentation period were correctly identified and denied in accordance with the protocol. For the legitimate requests, though their acceptance rate varied daily, it was generally seen to improve with continual usage of the protocol. To better visualize this behavior, we divided the dataset into three subsets in order to show the performance improvement in each subset. The subsets were created as follows: The first subset contained data collected in the first 30 days, the second subset contained data collected in the next

30 days (i.e., from the 31st day to the 60th day), and the third subset contained data collected from the last 30 days (i.e., from the 61st day to the 90th day). In each subset (time interval), we divided the legitimate requests into the typed and non-typed categories in order to observe the daily acceptance rates of each category (see Table 4.1 for a brief description of the dataset).

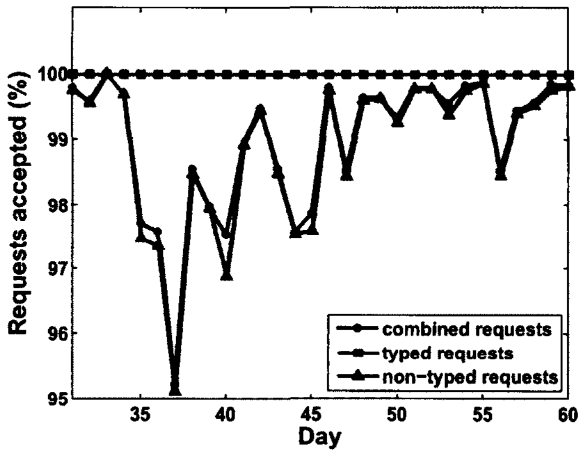
Table 4.1: Characteristics of the dataset used.

	Period of traffic collection	**No. of unique typed requests**	**No. of unique non-typed requests**
1st 30-day period	Nov'12	1488	18269
2nd 30-day period	Mid-Dec'12 - Mid-Jan'13	1529	16372
3rd 30-day period	Feb'13	1475	16736

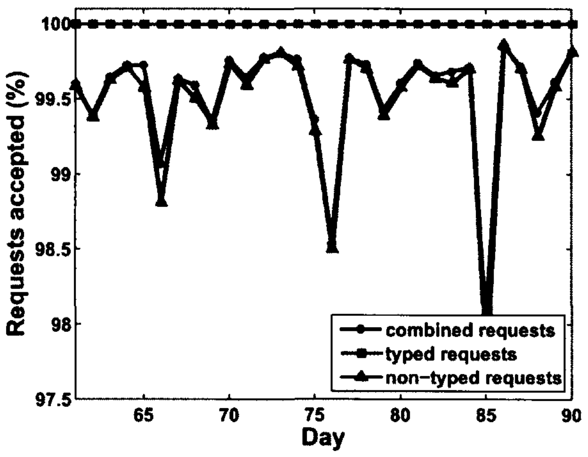
The results from this study are presented in Figure 4.7. These results show that, on each day, every typed request was accepted. Some of the non-typed requests, however, were not accepted since these requests had not been typed previously (hence violated the protocol's authentication requirement). Nonetheless, we observed that on average, the daily true accept rate for the non-typed category was approximately 98.8%, 98.9%, and 99.5% in the three time intervals, respectively. Since this category dominated the typed category (as seen in plots (a), (b) and (c) of Figure 4.8), their acceptance rate greatly influenced the overall acceptance rate of the combined requests (both typed and non-typed) whose daily true accept rate was seen to be approximately 98.8%, 99%, and 99.5% on average, in the three respective time intervals (see Table 4.2 for a summary of the average true accept rates in each period).



(*a*)

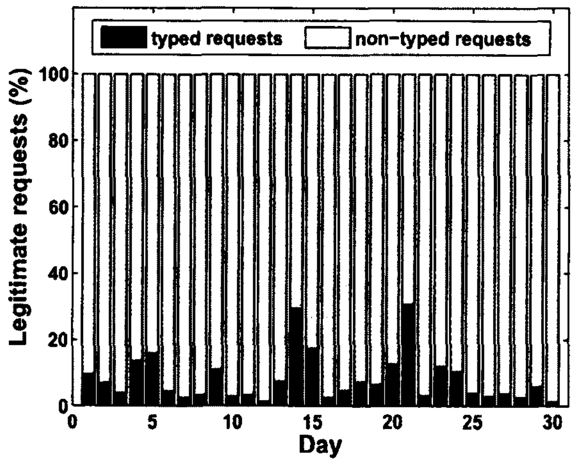


(*b*)

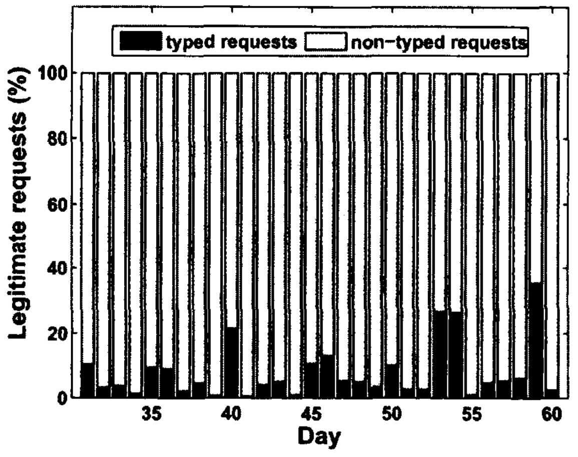


(*c*)

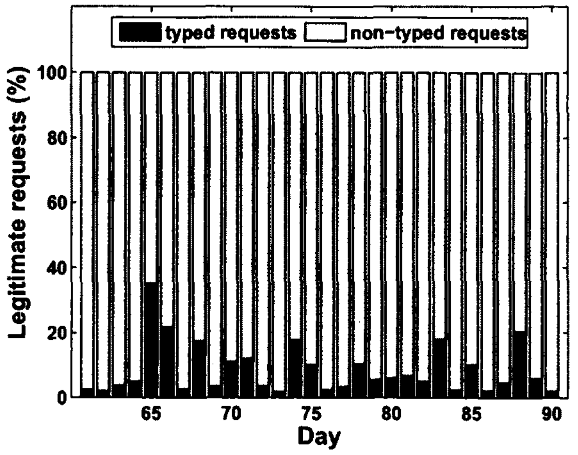
Figure 4.7: Plots (a), (b) and (c) show, in 30-day intervals, the daily percentage acceptance rates of legitimate requests (both the typed and non-typed requests) during the period under study.



(a)



(b)



(c)

Figure 4.8: The bar graphs in plots (a), (b) and (c) show, in 30-day intervals, the percentage-wise contributions of typed and non-typed requests to the overall legitimate requests seen daily during the period under study.

Table 4.2: Percentage results during the 90-day period for, (i) daily averages of true accept rates for the non-typed as well as combined requests, and (ii) maximum false reject rates for all legitimate requests (i.e., both user and program requests) as well as only the users' legitimate requests.

	Average True Accepts (%)		**Maximum False Rejects (%)**	
	Non-typed requests	**Combined requests**	**All legitimate requests**	**Only users' legitimate requests**
1st 30-day period	98.8	98.8	6	1.2
2nd 30-day period	98.9	99	5	0.92
3rd 30-day period	99.5	99.5	2	0.75

As seen from these results, the performance of the protocol improved in each successive time interval, implying that continual usage of the protocol led to an improvement in its accuracy measurements. This improvement is attributed to continual growth of the set, S, allowing non-typed requests (that had been typed previously by at least one of the users) to be accepted. During the entire experimentation period, we did not notice any false accepts.

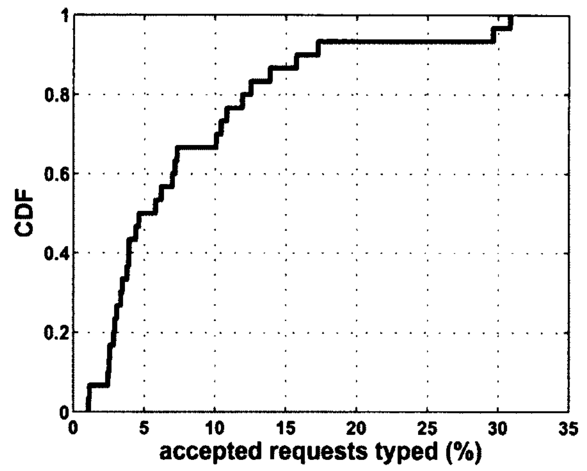
4.4.2 Enforced User Response

In this study, our objective was to examine the impact of the requirement for destination requests to be typed at least once before they could be accepted. Since this requirement is central to the working of our protocol, the number of times a user makes a non-typed request, which is rejected, and the user has to re-send the request after typing its destination, is an important aspect to consider. To investigate this, we asked the users to maintain their natural browsing habits when using the machines. We also advised them to type a destination IP address or domain name if their non-typed requests were rejected. (It is likely that some cases of rejected requests arise due to the associated server being down or offline; however, our dataset did not include such cases.)

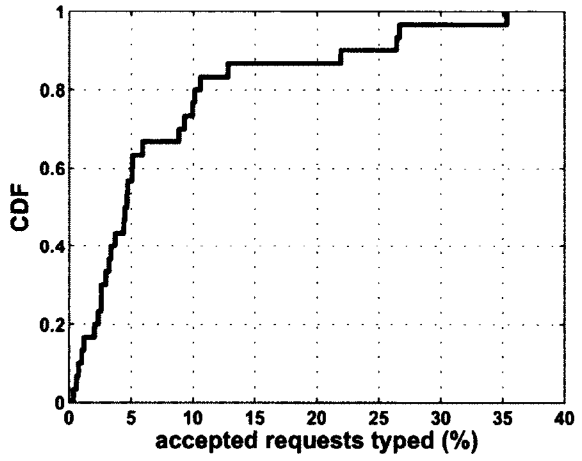
We present the observations from this study using Cumulative Distribution Function (CDF) plots that show the maximum observed quantities for, (a) accepted requests that were naturally typed, (b) legitimate non-typed user and program requests that were denied, and (c) legitimate non-typed user requests that were denied. The results in this study are also shown in 30-day intervals (representing the same three subsets described above) of the entire 90-day period.

Interestingly, we observed that, in a number of instances, the users naturally typed the destinations without being asked. Plots (a), (b) and (c) of Figure 4.9 show that in the three time intervals considered, accepted requests that had been voluntarily typed by users were 30%, 35% and 35%, respectively.

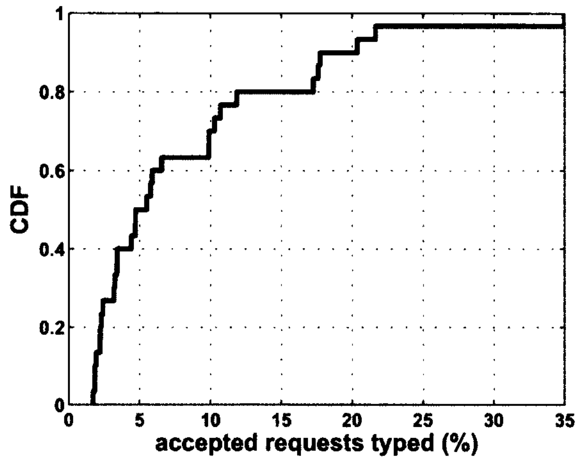
The CDFs of false reject rates for non-typed user and program requests, during the three 30-day intervals representing the entire 90-day period, are shown together in plots (a), (b) and (c) of Figure 4.10. They show that the maximum false reject rates for all non-typed legitimate requests on a given day were less than 6%, 5% and 2%, in each of the three 30-day periods, respectively. We observed that as the set "S" became populated over time, the number of false rejects dropped, leading to daily average false reject rates of approximately 1.2%, 1% and 0.5% in each of the three 30-day periods, respectively. We point out that these false reject rates, however, do not relate to the ability of the protocol to prevent *information exfiltration* but rather simply reflect the usability of the protocol.



(*a*)

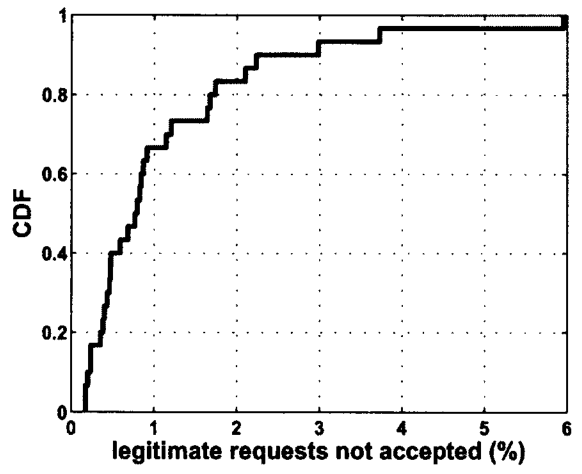


(*b*)

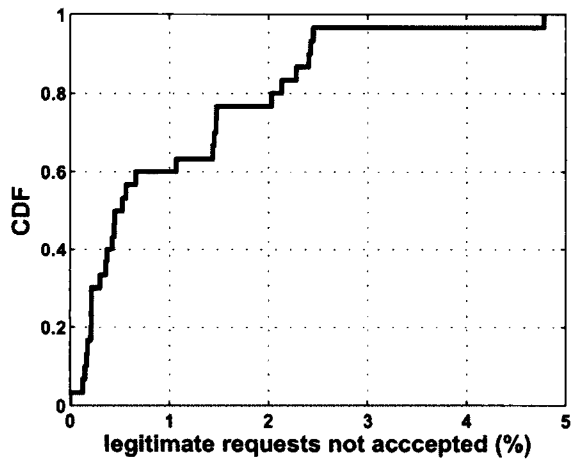


(*c*)

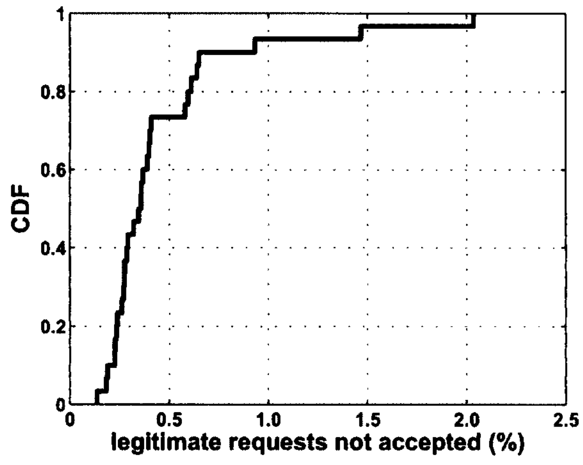
Figure 4.9: CDF plots showing, in 30-day intervals, the accepted requests that were typed during the period under study.



(*a*)



(*b*)



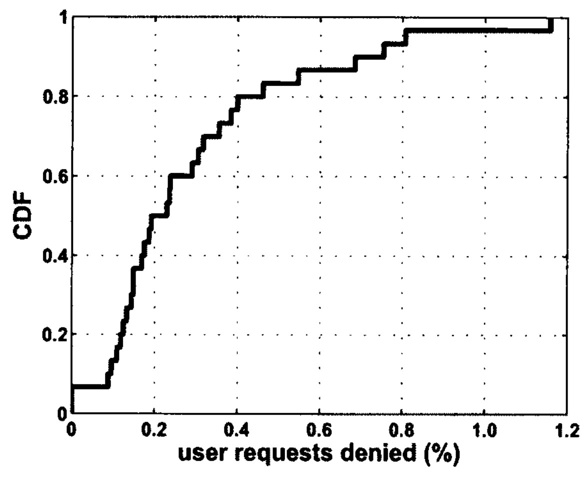
(*c*)

Figure 4.10: CDF plots showing, in 30-day intervals, the legitimate non-typed user and program requests that were denied during the period under study.

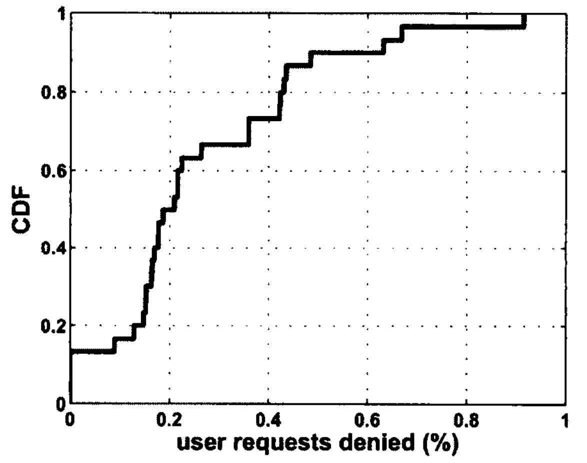
Of the requests that were non-typed, some of these were rejected (because they had not been previously typed) and users were compelled to type these requests in order for them to be accepted. We refer to these requests as the non-typed user requests. Plots (a), (b) and (c) of Figure 4.11 show the CDFs of the rejected non-typed user requests. They show that the maximum false reject rate on a given day for non-typed user requests, during the three 30-day intervals representing the entire 90-day period, were less than 1.2%, 0.92% and 0.75%, respectively (see Table 4.2). When the users typed those requests, all of them were accepted.

These percentages give a measure of the level of inconvenience that the users experienced due to the requirement set by the proposed scheme. However, we found that in comparison to the maximum percentages of requests that were naturally typed by the users (which was 30%, 35% and 35% of all accepted requests), the forced typing (which was 1.2%, 0.92% and 0.75% of only non-typed user requests) was not very significant. In addition, the forced typing requirement is akin to somewhat common scenarios that a user experiences when he/she deletes his/her browsing history, or uses a new Web browser and has to type the domain name/IP addresses before he/she can be assisted by the autocomplete feature of the browsers, and thus is not a significant deterrent from using the protocol.

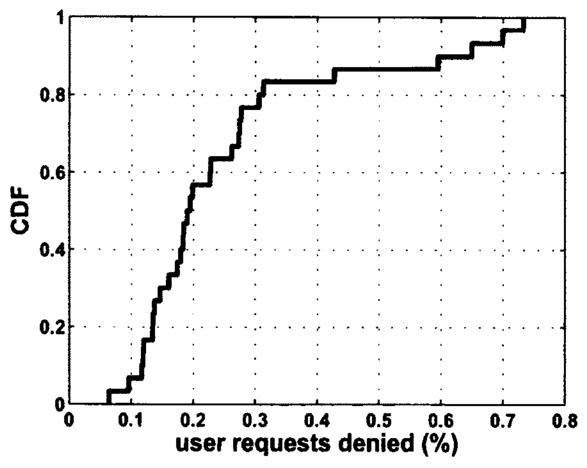
In Appendix A, we provide the daily trends of these results, i.e., accepted requests that were typed, legitimate non-typed user and program requests that were denied as well as the legitimate non-typed user requests that were denied.



(*a*)



(*b*)



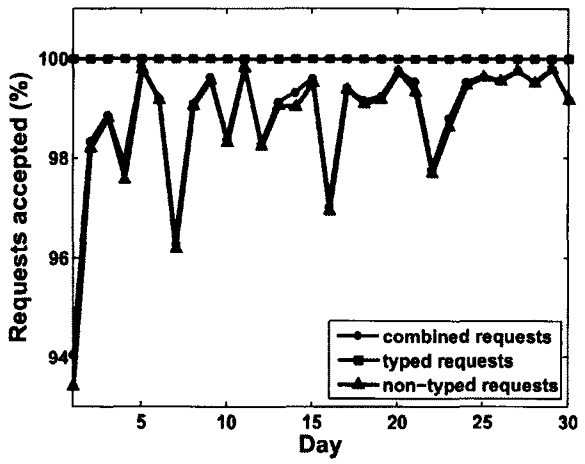
(*c*)

Figure 4.11: CDF plots showing, in 30-day intervals, the legitimate non-typed user requests that were denied during the period under study.

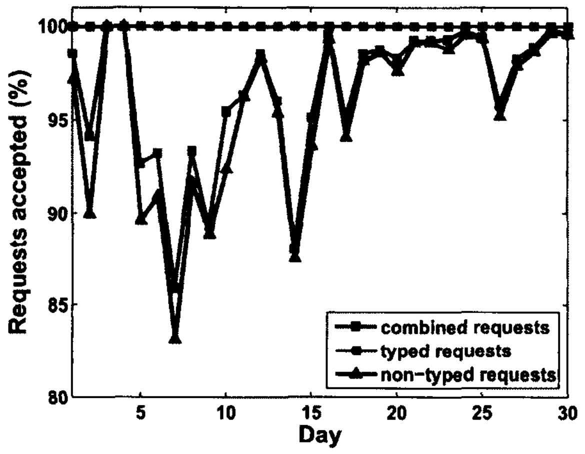
4.4.3 Replicating these Results

Motivated by the results obtained over the entire 90-day period, we sought to investigate whether this performance was tied to the specific time frame of protocol usage or whether it could be replicated, say if protocol usage began on a different day. To achieve this, we repeated our experiments using different start dates of protocol deployment in order to observe the change in performance, if any. From the three subsets created above, we took each subset to represent a unique and independent 30-day period of protocol usage, i.e., for each time period, we considered the protocol to have been in use only during that specific 30-day period. At the start of each period, therefore, the set S was cleared (or emptied) such that there was no history of previously typed requests. This investigation revealed that the behavior of the protocol was fairly consistent irrespective of the initial date of deploying the protocol. For instance, when considering the daily acceptance rates of legitimate requests (see Figure 4.12), we found that the daily average of true accept rates for legitimate typed and non-typed requests during each of these 30-day periods was consistently above 96% with no false accepts being observed.

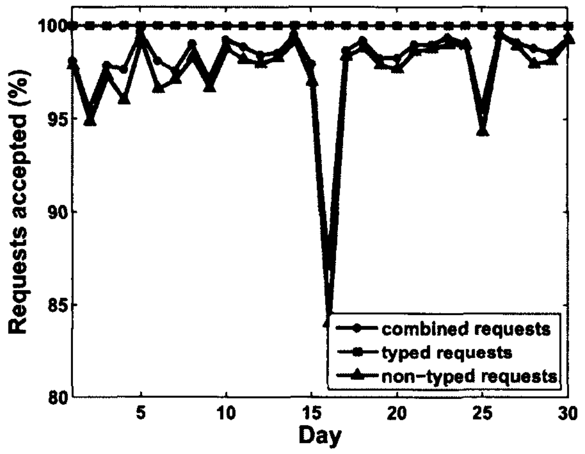
Figures 4.13 and 4.14 provide a summary of the results obtained from this study. They show the averages of the daily percentages of legitimate requests that were accepted, accepted requests that were typed and non-typed, legitimate user and program requests that were denied, and legitimate user requests that were denied. To obtain these results, for each quantity being measured, we averaged the daily percentage results from the three individual periods, e.g., the percentage value on



(a)



(b)



(c)

Figure 4.12: Plots (a), (b) and (c) show the daily percentage acceptance rates of legitimate requests (both the typed and non-typed requests) during the new first, second and third 30-day periods under study.

Day 1 represents the average of the three percentage values obtained on *Day 1* from each of the three 30-day periods.

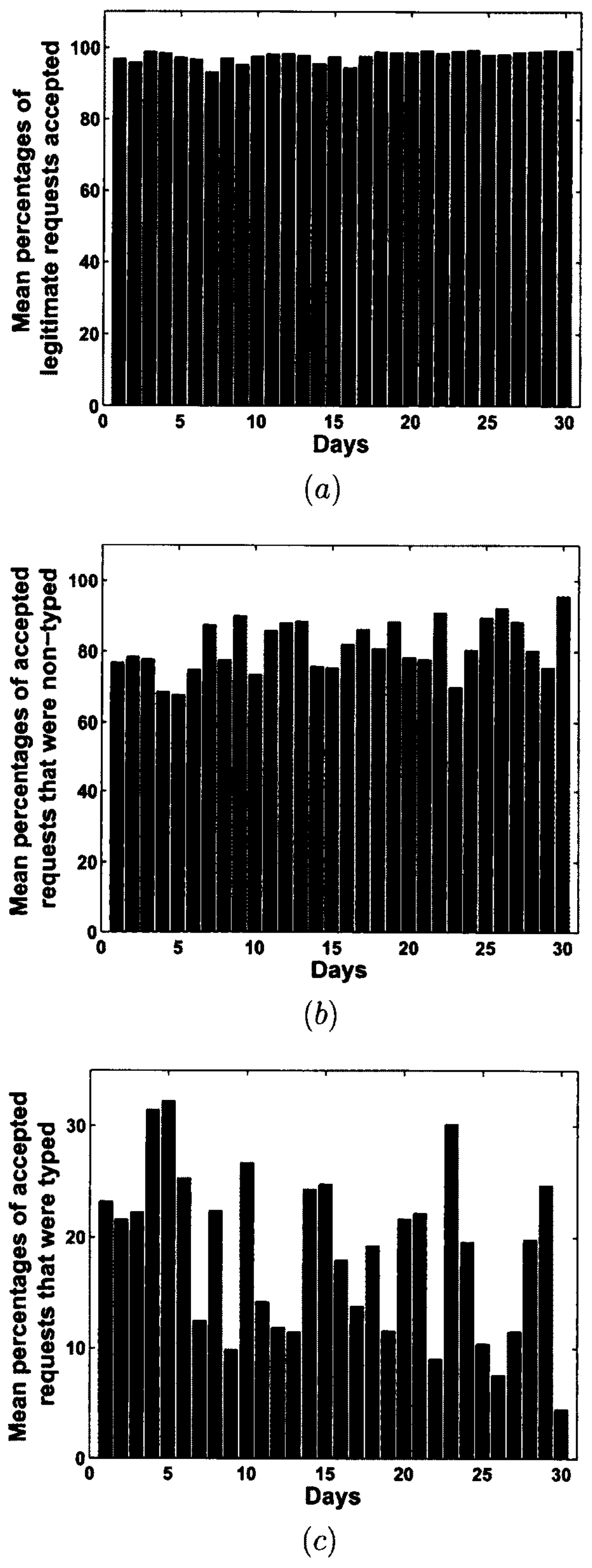
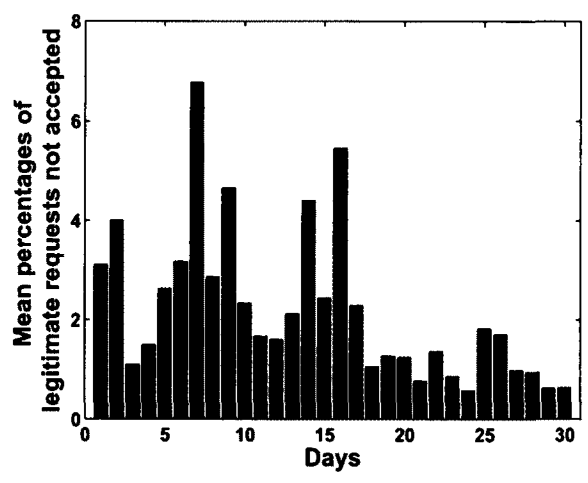
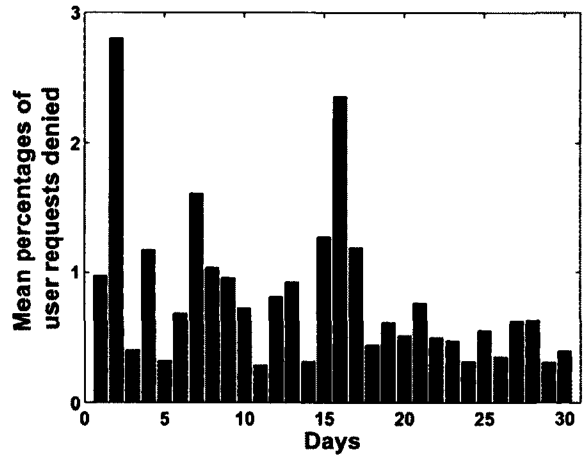


Figure 4.13: Percentages of, (a) legitimate requests that were accepted, (b) accepted requests that were non-typed, (c) accepted requests that were typed, averaged from the daily rates observed during the three 30-day periods under study.



(a)



(b)

Figure 4.14: Percentages of, (a) legitimate user and program requests that were not accepted, (b) legitimate user requests that were not accepted, averaged from the daily rates observed during the three 30-day periods under study.

CHAPTER 5

DISCUSSIONS

5.1 Alternative Placement of the Authenticated Channel

In this dissertation, the presented protocol has utilized an authenticated channel that is placed outside (i.e., bypasses) the host and therefore is presumed to be inaccessible by the malware inside a possibly compromised host computer. This serves not only to guarantee that the information transmitted through this channel is not tampered with, but also that it cannot be blocked (or removed) by the malware.

Before settling for this design, we considered the alternative of running the authenticated channel through the host. In such a design, the security of information transmitted through this channel would be guaranteed by transmitting the information, for example, through commonly used secure sessions for data transmission such as the Transport Layer Security (TLS) Protocol [9]. In this way, the information sent through the authenticated channel would be encrypted using a secret key known only to the hardware module (Alice) and the verifier (Bob). This would guarantee that an adversary (Eve), located on the host, would neither be able to encrypt her own information nor discover the contents of Alice's encrypted information without knowledge of the key.

However, a drawback of such a design for our protocol is, there being no independent communication channel between the hardware module and the verifier (i.e., without going through the host), the adversary could potentially block all information being sent from the host to the verifier. Though this would not be beneficial to the adversary, in terms of exfiltrating information, it is likely to lead to a denial of service scenario for the user(s) since all Web requests will likely be denied. This in turn is likely to negatively impact on the user convenience aspect of our protocol. If such a scenario were to occur, it is likely to go on for a while without being detected, because the user may think that the Web server to which his/her request is being sent, is offline or busy.

By placing the authenticated channel outside the host, therefore, we are able to ensure that in such denial of service cases where the adversary blocks Web requests, the authenticating information for those requests can still be sent to the verifier. As a check mechanism, once the verifier receives several pieces of authenticating information, without corresponding Web requests, it could send a message to the system administrator informing them of a possible denial of service scenario that requires investigation.

5.2 Verifier Workload

In this protocol, the verifier is tasked with such a significant amount of work (i.e., continual updating of the set, S of authenticated destinations, as well as scanning the set to check for the presence of an entry) that it needs to execute hastily so as not to slow down the authentication process. In a practical deployment of this protocol, care must be taken to ensure that the verifier is not bogged down by these operations. An important consideration when choosing a data structure to implement the set, S therefore should be the need to minimize the time complexity of the insertion and searching operations. One way of achieving this is by using a hash table-like implementation of S in which an m-bit array, employing a hash function, h to map an input, i into a position in the array (i.e., $0 < h(i) < m$), acts as a pointer to the possible presence of an element in S. In this setup, all array indices are initially set to 0 and a particular index is set to 1 when the hash value of a given input is mapped to that index. Ideally, the hash values produced by h should be uniformly distributed such that there is an equal chance of an input being mapped to any of the array indices. This reduces the possibility of hash collisions (i.e., more than one input mapping to the same array index) occurring. However, given that, theoretically, there exists an infinite set of inputs that would have to be mapped into a finite set of array indices, hash collisions are still likely to occur. This could result in false positive results (an element being reported to be in the array when it is not) when searching for an element whose hash value corresponds to an index that has previously been set to 1.

An approach to avoid these false positives is to maintain a self-balancing binary search tree (e.g., a red-black tree) for each array index (see Figure 5.1 in which a red-black tree is shown for only index 0 of the array) such that the actual values of i are stored in the tree associated with the array index to which $h(i)$ is mapped.

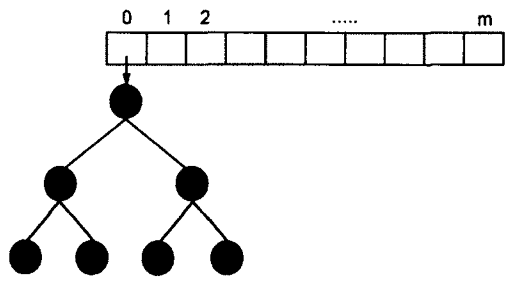


Figure 5.1: Hash table-like data structure in which, for each array index, there is a red-black tree to store the elements whose hash value corresponds to that index.

In this way, when searching for a specific element, computing its hash value only serves to locate the index under whose tree to search for that element. A self-balancing binary search tree is ideal for such an implementation because elements in the tree are always stored in a sorted manner; hence, the time required to perform insertion and search operations for tree elements is not adversely affected by the growth of the tree.

In the context of our protocol, when a destination (domain name or IP address) is to be added to the set S, the destination's hash value would first be computed, after which the array index corresponding to the destination's hash value would be set to 1. The specific destination itself would then be stored in the red-black tree under the associated array index. When searching whether a given destination is in S or not, the hash value of that destination would first be computed to determine which index of the array the destination's hash value maps to. If the corresponding index is set to 0, this would directly imply that the destination is not in set S (since there is no likelihood of false negative results). On the other hand, if the index is set to 1, this would imply that the destination is possibly present in S. The red-black tree associated with that index would then be traversed to determine whether the element is actually stored or if it was simply a case of a false positive result.

Obviously, given that the expected number of entries (destinations) to be stored is bound to influence the anticipated size of the array (in order to further minimize the chances of hash collisions), choosing an appropriate size of m is key to the use of such a data structure. Though the exact number of destinations that will be stored cannot be known a priori, we posit that in our protocol, the astronomical growth of set S is not likely to occur since S is a global set, implying that a destination authenticated for one user is considered authenticated for all users in the same network and, therefore, is only stored once. Additionally, since studies on user browsing behavior have shown that users tend to revisit the same pages (e.g., see [25], [41], [29]), we expect that this will in turn limit the number of new entries to be added and subsequently the growth of S over time. The choice of m, therefore, is mainly dependent on the application scenario (i.e., number of users for whom S is maintained) and could be determined from the observation of the number of unique destinations to which users send requests in a given period of time (e.g., a week or a month).

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this dissertation, we have proposed a new non-interactive dual channel protocol that, unlike the existing protocols, is resistant to channel spoof attacks launched against the authenticated channel. This is especially important when applying the protocol to frequent or continuous authentication solutions that require replacement of the manual channel with a non-manual channel that is able to accommodate frequent transmission of authentication codes. We have shown how this protocol can be applied for authenticating network traffic and demonstrated its effectiveness in addressing *information exfiltration* with minimal user participation and/or inconvenience. Though our protocol requires that users type the destinations of their requests at least once before these requests can be authenticated, any subsequent requests to these destinations are accepted without the need for them to be typed again.

In the presented protocol, a malware can remove the traffic requests sent through the host, causing a temporary denial of service. This, however, does not affect *information exfiltration* as defined in this dissertation. In addition, since the protocol assumes that a legitimate user does not type a malicious IP or domain name, in cases where such typing may be required (e.g., during investigation of IPs by a malware analyst), the verifier should be made aware of such destinations.

To realize this protocol in practice, we developed a small scale laboratory prototype comprised of four computers. When this prototype was deployed for a 90-day period of daily usage, we were able to achieve, on average, a 99.5% daily true accept rate for legitimate requests (i.e., requests sanctioned by the user) with 0% false accepts. During the same period, we also observed that the maximum percentage of false rejects was less than 6%, of which less than 1.2% of these were user requests that were denied for not having been typed.

Since our protocol requires that users type a request's destination at least once before it can be accepted, we investigated the likelihood of users voluntarily typing destinations of their requests and discovered that this happened approximately 35% of the time. This was an encouraging discovery in the sense that the protocol's typing requirement did not pose a significant encumbrance to the users (since typing was a part of the their normal browsing behavior). These results, though obtained from a small scale deployment of the protocol, offer encouraging insights that can be leveraged on for a large scale deployment of the protocol, if needed.

This dissertation has also proposed two extensions to the protocol in order to address unique scenarios that may arise during the implementation of the protocol. In the first extension, we proposed to address scenarios where *information exfiltration* occurs through links (which may be malicious) embedded in responses to legitimate requests. Since these links' destinations may not have been typed previously, their authenticity cannot be verified and are only presumed to be safe because they are contained in the responses to legitimate requests. We propose the acceptance of these requests for a defined period of time (set by the network administrator) in order to achieve the desired trade-off between user convenience and prevention of *information exfiltration.*

The second extension considered exfiltration via legitimate third party services (e.g., webmail servers). This is a more complex scenario since the final destination of the request(s) can only be determined by the third party services. Our proposal to address these scenarios, therefore, requires the participation of these third party services and, as a result, is not implemented in this dissertation.

In this dissertation, we were able to realize the dual channels using a Raspberry Pi, which merely served as an example of an affordable option to realizing the dual channels. We postulate that the dual channel aspect of this dissertation can be incorporated into future designs of keyboards that support dual channel functionalities. In such designs, the functionalities of the Raspberry Pi that were utilized in this work (such as the Ethernet port, processing unit) could be built in the keyboard itself such that there is no need to attach an external hardware module to the keyboard.

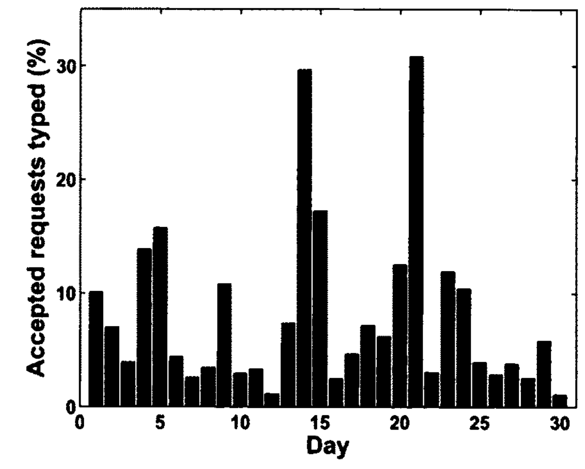
6.2 Future Work

The work presented in this dissertation can be further explored in several ways. For instance, the typing requirement specified in this protocol is hinged on the assumption that a physical keyboard exists to which a secure hardware module can be attached to extract keystroke contents before they are passed on to the host operating system. While this assumption holds mainly for desktop computers and laptops, the proliferation of mobile devices (such as tablet computers, cell phones, etc.) that do not typically have physical keyboards, and yet are also connected to the Internet and thus are likely to be sources of *information exfiltration*, begs for the exploration of how the protocol presented in this dissertation can be applied to such devices since access to their keystroke contents before they are passed on to the operating system may not be as straightforward.

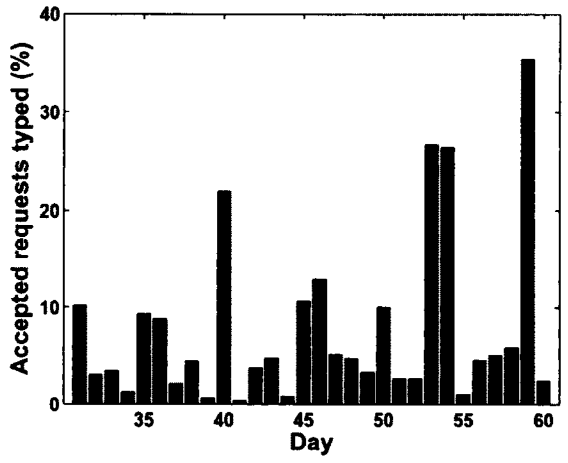
Another aspect worth investigating is how this protocol can be further adapted for continuous user authentication systems, specifically those that are based on behavioral biometrics (e.g., users' keystroke patterns) since these systems, much like the traffic authentication ones, also face the risk of adversaries compromising the authentication efforts if the host is compromised (e.g., see [35], [39]). To the best of our knowledge, no defense mechanisms to prevent host-based compromises against continuous user authentication have been proposed to date. This is perhaps due to the belief that in a compromised host scenario, the user has already lost to the impostor [14]; hence, continuous authentication has no merit under such a scenario. Contrary to this view, we postulate that with further research, the protocol presented in this dissertation can be a stepping stone for designing defense mechanisms that ensure successful continuous authentication in the face of possible host-based compromises.

APPENDIX A

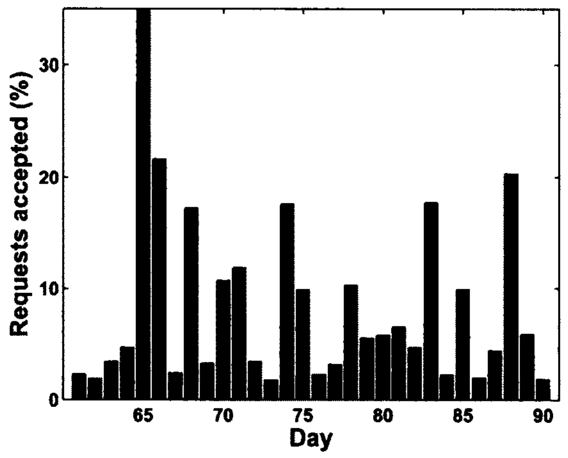
ADDITIONAL RESULTS SHOWING DAILY TRENDS OF ACCEPTED AND DENIED REQUESTS



(*a*)

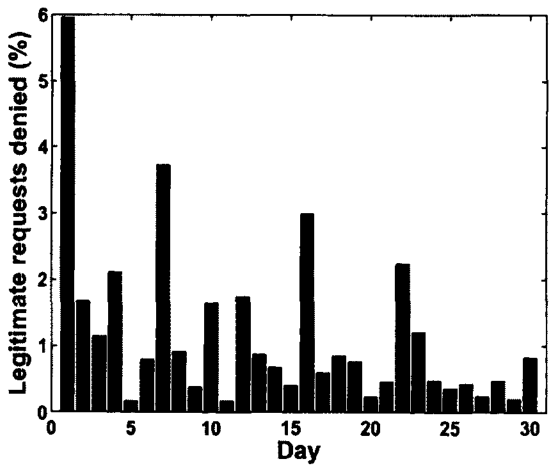


(*b*)

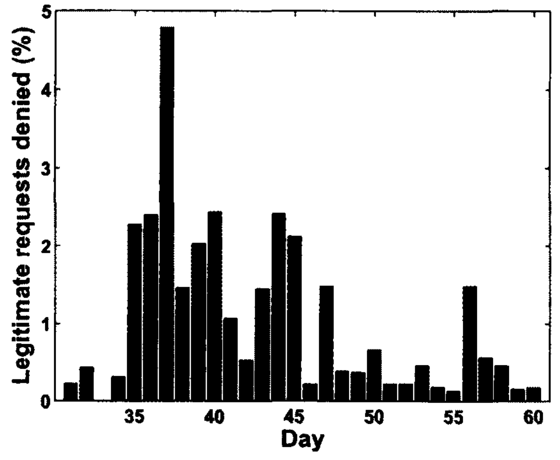


(*c*)

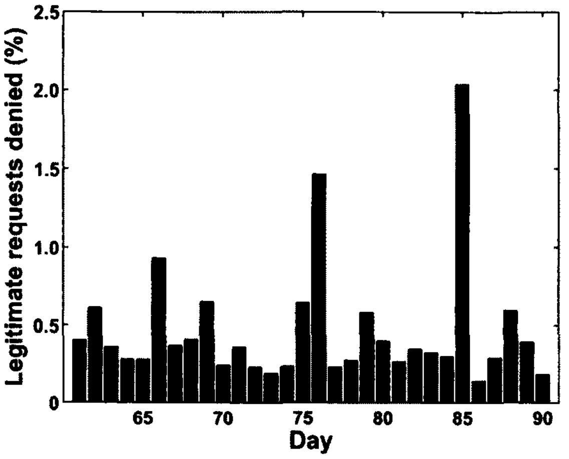
Figure A.1: Bar graphs (a), (b), and (c), respectively, show the daily percentages of accepted requests that were voluntarily typed during the first 30 days, the next 30 days, and the last 30 days of the period under study.



(a)

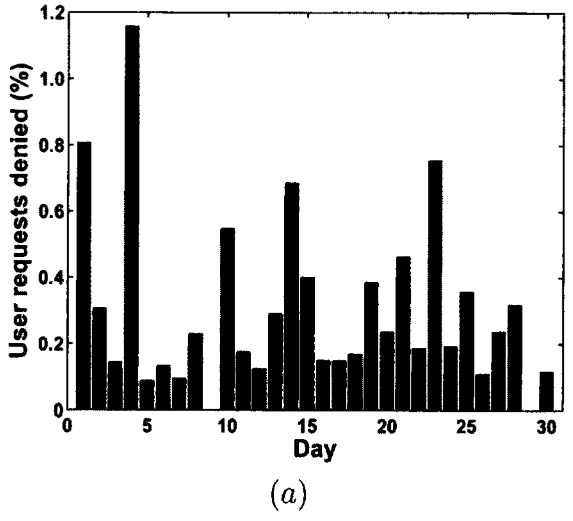


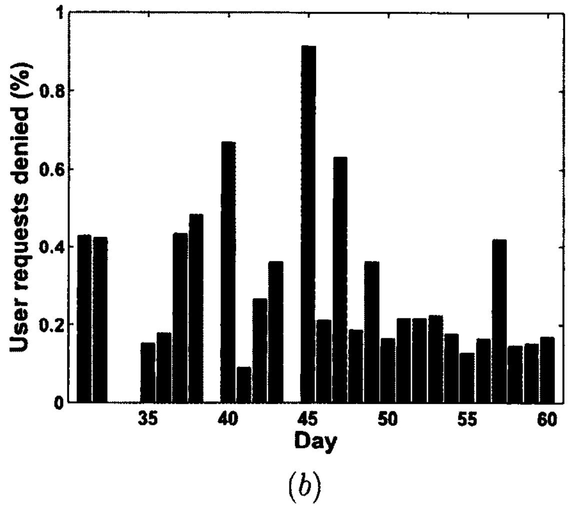
(b)



(c)

Figure A.2: Bar graphs (a), (b), and (c), respectively, show the daily percentages of legitimate non-typed user and program requests that were denied during the first 30 days, the next 30 days, and the last 30 days of the period under study.





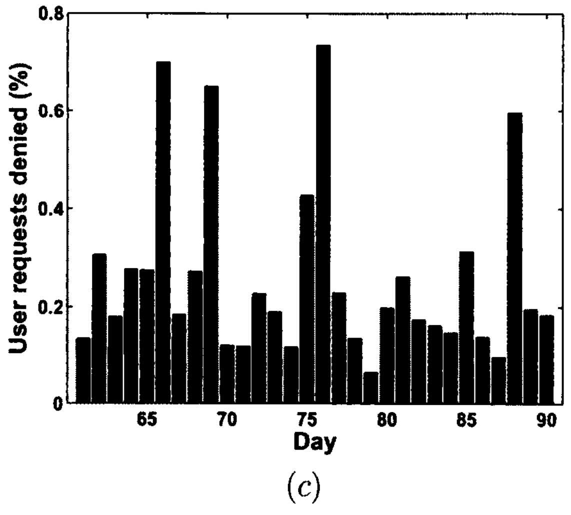


Figure A.3: Bar graphs (a), (b), and (c), respectively, show the daily percentages of legitimate non-typed user requests that were denied during the first 30 days, the next 30 days, and the last 30 days of the period under study.

APPENDIX B

TABULAR COMPARISON OF EXISTING AND RELATED PROTOCOLS

Table B.1: Characteristics of existing non-interactive message authentication protocols.

Property	BSSW'02	GMN'04	PV'06	MS'06	RWN'07
Assumed security of manual channel	weak	strong	weak	weak	weak
Security feature used	collision resistan-t hash	universal hash	second preimage resistant hash and trapdoor commitment scheme	hybrid collision resistant hash	enhanced target collision resistan-t hash
Key-based hash function used?	no	yes	no	yes	yes
Input to hash function	message	message	commit value	message and key	message

BIBLIOGRAPHY

[1] F. Aloul, S. Zahidi, and W. El-Hajj. Two factor authentication using mobile phones. In *Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications*, pages 641–644, Rabat, Morocco, 2009.

[2] D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of the Network and Distributed System Security Symposium*, pages 7–19, San Diego, CA, USA, 2002.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] K. Borders and A. Prakash. Webtap: detecting covert web traffic. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 110–120, Washington, DC, USA, 2004.

[5] J. Brainard, A. Juels, B. Kaliski, and M. Szydlo. A new two-server approach for authentication with short secrets. In *Proceedings of the 12th Conference on USENIX Security Symposium*, pages 14–14, Washington, DC, USA, 2003.

[6] Codenomicon. The heartbleed bug, 2014. http://heartbleed.com/. Last Accessed in April, 2014.

[7] D. Coffin and D. Coffin. Two-factor authentication. In *Expert Oracle and Java Security*, pages 177–208. Apress, 2011.

[8] DARPA. Active authentication. http://www.darpa.mil/Our_Work/I2O/Programs/Active_Authentication.aspx. Last Accessed in January, 2014.

[9] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol, Version 1.2. RFC 5246, August 2008.

[10] DNSBL. Spam database lookup, 2013. http://www.dnsbl.info/. Last Accessed in October, 2013.

[11] B. Dodson, D. Sengupta, D. Boneh, and M. Lam. Secure, consumer-friendly web authentication and payments with a phone. In *Mobile Computing, Applications, and Services*, pages 17–38, 2012.

[12] D. Duncan and D. Myers. Method and apparatus for automatically detecting sensitive information, applying policies based on a structured taxonomy and dynamically enforcing and reporting on the protection of sensitive data through a software permission wrapper, March 2006. US Patent App. 10/930,173.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[14] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE Transactions on Information Forensics and Security*, 8(1):136–148, 2013.

[15] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Trustworthy and personalized computing on public kiosks. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, pages 199–210, Breckenridge, CO, USA, 2008.

[16] C. Gehrmann, C. J. Mitchell, and K. Nyberg. Manual authentication for wireless devices. *RSA Cryptobytes*, 7(1):29–37, 2004.

[17] Google. About 2-step verification. http://support.google.com/accounts/bin/answer.py?hl=en&answer=180744. Last Accessed in September, 2012.

[18] T. C. Group. Trusted platform module, 2014. http://www.trustedcomputinggroup.org/developers/trusted_platform_module. Last Accessed in March, 2014.

[19] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-bot: improving service availability in the face of botnet attacks. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, pages 307–320, Boston, MA, USA, 2009.

[20] P. Gupta. *Algorithms for routing lookups and packet classification.* PhD thesis, Stanford University, 2000.

[21] D. Irakiza, M. Karim, and V. Phoha. A non-interactive dual channel continuous traffic authentication protocol. *IEEE Transactions on Information Forensics and Security*, 9(7):1133–1140, 2014.

[22] R. C. Jammalamadaka, T. W. v. d. Horst, S. Mehrotra, K. E. 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, Miami, FL, USA, 2006.

[23] H. Jin, D. S. Wong, and Y. Xu. An efficient password-only two-server authenticated key exchange system. In *Proceedings of the 9th International Conference on Information and Communications Security*, pages 44–56, Zhengzhou, China, 2007.

[24] J. Katz, P. MacKenzie, G. Taban, and V. Gligor. Two-server password-only authenticated key exchange. *Journal of Computer and System Sciences*, 78(2):651–669, 2012.

[25] R. Kumar and A. Tomkins. A characterization of online browsing behavior. In *Proceedings of the 19th International Conference on World Wide Web*, pages 561–570, Raleigh, NC, USA, 2010.

[26] M. Mannan and P. C. Van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *Proceedings of the 11th International Conference on Financial cryptography and 1st International Conference on Usable Security*, pages 88–103, Scarborough, Trinidad and Tobago, 2007.

[27] A. Mashatan and D. R. Stinson. Noninteractive two-channel message authentication based on hybrid-collision resistant hash functions. *Information Security, IET*, 1(3):111–118, 2007.

[28] A. Mashatan and D. R. Stinson. Interactive two-channel message authentication based on interactive collision resistant hash functions. Technical report, University of Waterloo, Canada, 2007.

[29] A. L. Montgomery and C. Faloutsos. Identifying web browsing trends and patterns. *Computer*, 34(7):94–95, 2001.

[30] L. H. Nguyen and A. W. Roscoe. Separating two roles of hashing in one-way message authentication. *IACR Cryptology ePrint Archive*, 2009.

[31] L. H. Nguyen and A. W. Roscoe. Authentication protocols based on low-bandwidth unspoofable channels: A comparative survey. *Journal of Computer Security*, 19(1):139–201, 2011.

[32] OpenSSL. Openssl cryptography and ssl/tls toolkit, 2014. https://www.openssl.org/. Last Accessed in March, 2014.

[33] A. Oprea, D. Balfanz, G. Durfee, and D. K. Smetters. Securing a remote terminal application with a mobile trusted device. In *Proceedings of the 20th Annual Computer Security Applications Conference*, pages 438–447, Tucson, AZ, USA, 2004.

[34] S. Pasini and S. Vaudenay. An optimal non-interactive message authentication protocol. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, pages 280–294, San Jose, CA, USA, 2006.

[35] K. Rahman, K. Balagani, and V. Phoha. Making impostor pass rates meaningless: A case of snoop- forge-replay attack on continuous cyber-behavioral verification with keystrokes. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 31–38, Colorado Springs, CO, USA, 2011.

[36] M. R. Reyhanitabar, S. Wang, and R. Safavi-Naini. Non-interactive manual channel message authentication based on etcr hash functions. In *Proceedings of the 12th Australasian Conference on Information Security and Privacy*, pages 385–399, Townsville, Australia, 2007.

[37] R. L. Rivest and A. Shamir. How to expose an eavesdropper. *Communications of the ACM*, 27(4):393–394, 1984.

[38] SBL. The spamhaus block list, 2013. http://www.dnsbl.info/. Last Accessed in October, 2013.

[39] A. Serwadda and V. V. Phoha. Examining a large keystroke biometrics dataset for statistical-attack openings. *ACM Transactions on Information Systems and Security*, 16(2):1–30, 2013.

[40] R. Sharp, J. Scott, and A. R. Beresford. Secure mobile computing via public terminals. In *Proceedings of the 4th International Conference on Pervasive Computing*, pages 238–253, Dublin, Ireland, 2006.

[41] L. Tauscher and S. Greenberg. How people revisit web pages: empirical findings and implications for the design of history systems. *International Journal of Human-Computer Studies*, 47(1):97–137, 1997.

[42] B. Thuraisingham. Data mining for security applications: Mining concept-drifting data streams to detect peer to peer botnet traffic. In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics*, pages 29–30, Taipei, Taiwan, 2008.

[43] S. Vaudenay. Secure communications over insecure channels based on short authenticated strings. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, pages 309–326, Santa Barbara, CA, USA, 2005.

[44] M. Wu, S. Garfinkel, and R. Miller. Secure web authentication with mobile phones. In *Proceedings of the DIMACS Workshop on Usable Privacy and Security Software*, Piscataway, NJ, USA, 2004.

[45] G. Xu, C. Borcea, and L. Iftode. Satem: Trusted service code execution across transactions. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, pages 337–338, Leeds, UK, 2006.

[46] K. Xu, H. Xiong, C. Wu, D. Stefan, and D. Yao. Data-provenance verification for secure hosts. *IEEE Transactions on Dependable and Secure Computing*, 9(2):173–183, 2012.

[47] G. Yang, D. S. Wong, H. Wang, and X. Deng. Two-factor mutual authentication based on smart cards and passwords. *Journal of Computer and System Sciences*, 74(7):1160–1172, 2008.

[48] Y. Yang, R. H. Deng, and F. Bao. A practical password-based two-server authentication and key exchange system. *IEEE Transactions on Dependable and Secure Computing*, 3(2):105–114, 2006.

[49] H.-N. You, J.-S. Lee, J.-J. Kim, and M.-S. Jun. A study on the two-channel authentication method which provides two-way authentication in the internet banking environment. In *Proceedings of the 5th International Conference on Computer Sciences and Convergence Information Technology*, pages 539–543, Seoul, Korea, 2010.