Trust Models for Remote Hosts

by

Alexander Cowperthwaite

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Computer Science
Carleton University
Ottawa, Ontario, Canada
September 2011

Your file Votre référence
ISBN: 978-0-494-83189-2
Our file Notre référence
ISBN: 978-0-494-83189-2

Abstract

Many of the attacks on the internet today are a result of unintentional communication with a malicious host. Many current security efforts revolve around authenticating and verifying hosts based on their name as reported by the Domain Name System (DNS). These efforts either focus on securing the name translation (DNSSec and DNSCurve), or providing end to end security (SSL). Regardless of these efforts, users can still be misdirected to a malicious host.

We analyze current and proposed technologies for securely identifying and communicating with remote hosts and compare them with respect to their trust model. Based on this analysis we conclude a new trust model is needed. We describe such a trust model based on trusted referrals. We also present SSL-based Trusted Referrals, a proposed system design for securely identifying remote hosts on the web based on trusted referrals.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

One of the basic principles of navigation is arriving at the correct destination. Successfully identifying the destination—particularly the intended host—can be a challenge. On the Internet navigating to the wrong destination can have serious consequences including a loss of personal or sensitive information or dissemination of false or incorrect information. This can be the result of simple human error, computer error or a malicious attack.

Secure Sockets Layer (SSL) was created in the mid 1990s as a tool to assure the identity of a host and protect communications with that host. It uses a public key binded to a hostname to ensure the identity of the remote resource. This public key and hostname binding is verified and cryptographically signed by a Certificate Authority (CA), a trusted third party. Any user's browser can then verify the binding using the public key of the CA. SSL was the first tool for secure communications widely adopted by web browser manufacturers. It is credited with enabling e-commerce and is one of the foundations of the online economy.

Since its development, SSL has seen continued success in securing communications on the Internet; however, it has not seen universal use on web servers. The reasons for not deploying SSL on a web server vary depending on the organization's needs, but several potential limiting factors include: lack of professional resources to support

SSL, lack of computational resources to support additional processing load, and lack of financial resources to purchase a CA signed certificate. Organizations also do not necessarily feel the need for the security SSL provides unless they clearly deal in personal or sensitive information.

While SSL is not universally adopted among web servers, the use of the Domain Name System (DNS) is. DNS is the global namespace of the Internet; each host on the Internet has a unique name. It provides a hierarchical structure through which hostnames can be translated into machine routable IP addresses. DNS was originally developed to be lightweight and robust, but it was not designed to provide data integrity or authenticity. This has resulted in a number of widely known DNS vulnerabilities that can allow attackers to redirect traffic to the site of their choice. DNS Security Extensions (DNSSec) has been proposed to add public key cryptography into DNS to provide the security it originally lacked.

DNSSec and SSL in many ways share a common architecture, hierarchical public key cryptography anchored by a trusted third party. In many cases, the trusted third parties are actually the same: Verisign is both a CA and DNS root operator, for example. The security provided by DNSSec and SSL can be circumvented in similar ways, for example by providing a victim with a malicious or incorrect hostname a user may become securely connected to a host other than they intended. They both also require that a user have absolute trust in all third party trust anchors installed in their browser. These systems have similar trust models.

A trust model describes which components, actors, or relationships (elements) must fulfill their roles in order to maintain the security properties of a system. A trust model has three distinct elements:

The Security Characteristics The security properties that a system may or may not provide: confidentiality, integrity and authenticity.

The Architecture of Trust This is how trust is assigned and shared between elements. This includes how trust chains are built, between which elements trust links may be formed or anchored, and the attacks to which each relationship is vulnerable.

The Technical Mechanism This is the mechanism used to convey trust between two elements. The most common of these mechanisms are asymmetric key pairs, symmetric keys and username-password combinations.

We hypothesize that many of the technologies for secure communication share one or few trust models which makes systems vulnerable in similar ways. We perform an analysis of many secure communication technologies, with respect to their trust model to verify this hypothesis.

Our analysis of existing systems finds that the vast majority of current systems are built using two trust models and therefore are vulnerable to many of the same attack strategies. These two trust models are manual verification and an authoratative trusted third party. Existing secure Internet communication systems generally use a public key binded to a DNS hostname to identify a remote host. The verification that a public key does belong to a remote resource identified by a hostname can be performed manually or delegated to trusted third party. A user can manually verify that the public key belongs to a given resource, securely, through an out-of-band mechanism. A trusted third party can also perform this verification and sign a digital certificate attesting to its correctness. Other parties can then verify the authenticity and integrity of the digital certificate if they have the public key of the trusted third party.

Systems grouped together based on their trust model face similar limitations, that an attack can exploit. Manual verification methods provide strong security when a user has the tools and knowledge to verify the binding of a public key to a hostname;

however, this is frequently not the case for an average user. Authoratative trusted third party methods rely on two assumptions that do not always hold. First, that the trusted third party is in fact trusted, and second that the trusted third party performs sufficient verification. Usability is also a challenge across both techniques because users do not understand hostnames and many of the concepts that provide underlying security technologies (e.g. encryption and asymmetric cryptography).

To move beyond these limitations, we look to a new trust model: a trust model where the trust anchor is directly involved in the transaction and the user can simply and unintrusively make a trust decision for each transaction. When implemented, the technical mechanisms must be simple and not overbearing to a non-technical user. Finally it must provide strong security characteristics.

These characteristics lead us to a trust model based on trusted referrals, a trust model where trust in a remote host is established based on who referred a user to that host. In such a model the user can make a trust assessment about each referral, based on their relationship with the referrer. This trust model is aligned closely with the navigation model of the Internet: a user following web links is analogous to a trusted referral.

To describe the feasibility of such a trust model, we suggest a simple system of including public key hashes with HTML web links to create a trusted referral. A user implicitly makes a trust decision about the source of a link by choosing to follow it. This proposal only illustrates the direction of future research; it is not a complete solution. It potentially provides end-to-end security but uses a different trust model that builds trust on referrals, not arbitrary third parties (such as certificate authorities). Usability limits many of the systems we analyzed and we believe it is an important consideration. While we do consider usability in the design of our system, explicitly verifying the usability is beyond the scope of this thesis.

We evaluate the potential effectiveness of such a system by comparing it with the

other analyzed systems against several well known attack scenarios.

A system based on trusted referrals has serveral advantages to end users. The primary advantage of trusted referrals is that it eliminates the need for a trusted third party in which many current systems rely upon. In the case of SSL and DNSSec, the trusted third partied is relied upon to verify identity information, which is a potential weakness that can been exploited.

With trusted referrals, trust decisions are also being made by the user following the link, not a third party. The user makes the decision to trust a destination by following that link. A user is certainly still vulnerable if they make a bad choice, but the decision is back in control of the user. Further, this decision is leveraged without requiring complicated user actions such as managing installed root certificates.

Trusted referrals provide a limited scope of trust, it only verifies that the page a user arrives at through a referral (web link) was the destination intended by the link. It protects against many network based man in the middle attacks. If a page is compromised, including any trusted referrals, only those referrals are compromised. A user can potentially arrive at thier intended destination through another referrer, or can continue to go to any destination that is not on the compromised page securely. This is in contrast to SSL where if a single CA is compromised, all SSL connections based on CA signed certificates cannot be trusted, until the compromised CA is identified and removed as a source of trust.

In this thesis, our primary contribution is a thorough survey and theoretical analysis of trust models of existing systems for securely connecting to remote hosts. From the analysis we identify common weaknesses caused by the non-diversity of trust models. We propose a new trust model for securing Internet communication, trusted referrals. Finally, we describe the design of a simple system, SSL-based Trusted Referrals, that suggests the feasibility of systems based on trusted referrals. The system design we suggest still has many deployment challenges, limitations and lacks

any usability evaluation so it can only be viewed as a potential direction of future research.

We proceed by describing our initial motivation for this work, DNS and DNS Security Extensions in Chapter 2. We then describe SSL in Chapter 3. We follow this with other less common technologies for secure communications, SSH, Perspectives, DNSCurve, PGP, IPsec, Trust Management for Humans and ITrustPage in Chapter 4. In Chapter 5 we provide a thorough analysis of the discussed technologies trust models. In Chapter 6 we propose SSL-based Trusted Referrals. Chapter 7 discusses limitations, potential directions for future research, and concludes.

Chapter 2

The Domain Name System

In this chapter we describe the modern Domain Name System (DNS). We describe the motivations for DNS, how it has evolved including the current architecture. Then we describe the attack vectors in DNS which motivated DNS Security Extensions (DNSSec). We describe DNSSec and its architecture, including its trust model. Finally, we discuss the attacks against DNSSec and security it provides.

2.1 DNS

The Domain Name System (DNS) is the primary system used to translate human readable hostnames such as `ccsl.carleton.ca` to machine routable Internet Protocol (IP) addresses such as `134.117.225.13` on the Internet. It is a global distributed hierarchical database which performs hostname translations for almost every transaction on the Internet. It uses aggressive caching and lightweight communications to achieve efficiency, robustness and high availability.

2.1.1 History

The ARPA Internet (or ARPANet) was one of the original networks that grew into the modern Internet. It was the first packet switched network connecting research

centres in the United States and grew rapidly through the 1970s and 1980s [1].

Large networks of many hosts have a fundamental challenge: identifying the resource to which a user wishes to connect. On the network layer, the identification is performed through a unique binary address called an IP address. IPv4 addresses are four byte values usually displayed as dot separated, unsigned numbers between 0 and 255 such as `219.42.0.195`. To simplify the identification, hostnames were created. Hostnames are standard character strings, such as `alex-desktop` which are more meaningful and easier for people to remember. The mapping between hostname and IP address was originally maintained in a "hosts" file stored on each machine. Hostnames became the de facto method to identify resources on the Internet.

The Network Information Center was responsible for maintaining a complete hosts file for the entire ARPA Internet. In the early 1980s as the network continued to grow, this centralized repository became unmanageable; the hosts file would need constant updating. To replace the centralized model, Paul Mockapetris designed the distributed Domain Name System in 1983, published in RFC 882 and 883 [2,3]. This system used a tree hierarchy that corresponded to dot separated hostnames such as `mail.carleton.ca` and also loosely corresponded to the dot separated IP addresses. The original system received a redesign in 1987 to create the base DNS used today, the initial specifications which were published in RFC 1034 and 1035 [4,5]. This redesign maintained the same concepts but allowed DNS to scale with the rapid growth of the Internet. Since then there have been several further updates; however, overall it is the same system that performs the hostname to IP address translation for nearly every transaction on the modern Internet.

2.1.2 Architecture

The Domain Name System is a distributed database for storing information on domain names. While the primary purpose of DNS is to map domain names to IP addresses,

A records specify an IPv4 address for a host.

AAAA records specify an IPv6 address for a host.

NS records point to the authoritative name servers for a zone (domain).

CNAME records allow the specification of host aliases or canonical names (e.g. `www.example.com` refer to `berners-lee.example.com`).

MX records identify mail servers that can receive email for a domain.

Figure 1: The primary types of DNS records

it actually supports several kinds of records. The primary record types are shown in Figure 1.

The DNS system is composed of two types of systems: name servers and resolvers. Name servers return information, shown in Figure 1, about domains (also known as zones) when queried by DNS resolvers. DNS resolvers send queries to one or more name servers in order to service DNS requests from applications.

When a name server is configured as authoritative for its domain, its resource records are always interpreted as correct. To improve availability and reduce upstream server load, resolvers may also cache resource records until they have expired; while some domains specify time-to-live values on the order of minutes, it is more common for domains to permit their records to be cached for days. Cached records are not considered to be authoritative, i.e., they may contain incorrect or stale information; most consumers of DNS information, however, do not distinguish between authoritative and non-authoritative results.

Applications normally send queries to simple stub resolvers implemented as library functions. Stub resolvers forward requests to standalone recursive resolvers that perform the multiple name server queries normally required to answer any request. Recursive resolvers work best when most queries can be answered using cached responses (i.e. overtime, they see multiple requests for the same information). ISPs

normally operate DNS servers with recursive, caching resolvers for their customers; alternately, users can configure their systems to connect to other open DNS servers.[1]

DNS Hierarchy and Lookup

DNS name servers are structured as a hierarchical tree where the tree levels correspond to the dot-separated components of a domain name. A complete lookup follows a chain starting from the DNS root servers to a top level domain (TLD) down to any number of domains and subdomains below. We review the steps in a typical DNS lookup in Fig 2.

Each DNS query and response is normally sent in a single UDP packet. The query and response use the same structure; the query is sent with the response field blank. A 16 bit query ID field is used to distinguish outstanding requests [4, 5].

Trust Model

DNS does not have any strong cryptographic mechanisms to build trust. Note however, that it does delegate the duty of response through the DNS hierarchy. A response would only be accepted if it contained the query ID of an outstanding query. This field, intended to keep track of multiple outstanding queries, can also provide a weak authenticity check. The 16 bit field does not provide sufficient entropy to be secure, as we will see in Section 2.2.

[1] Traditionally recursive resolvers and name servers were both implemented as part of the same application (e.g., BIND 9 and earlier [6]). Newer DNS server designs, however, divide these functions into separate programs (e.g., `tinydns` and `dnscache` of djbdns [7]).

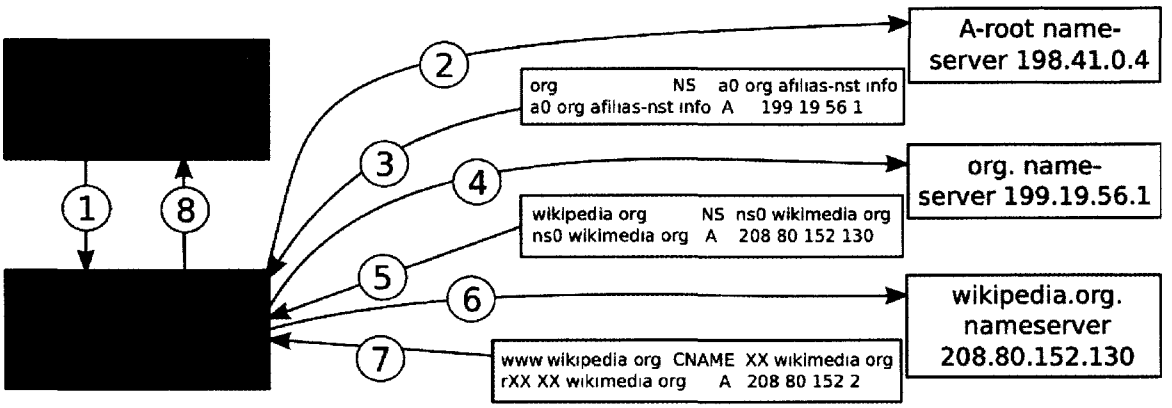

Figure 2: Example of a recursive DNS lookup for www.wikipedia.org

1. An application (e.g., a web browser) invokes the local stub resolver which sends the request to a local DNS server (a recursive, caching DNS resolver).

2. The DNS server will check its cache for an A record (IPv4 address) for `www.wikipedia.org`. If it is does not have an entry it will request a lookup from a randomly chosen root server. The IP addresses of standard root servers are built in to DNS servers.

3. The root server will reply with the NS record describing the name of the authoritative name server for the Top Level Domain (TLD) `.org`. To reduce traffic, it will also append "the glue": the A record for the `.org` TLD name server.

4. The recursive DNS resolver will request a lookup for `www.wikipedia.org` from the `.org` authoritative name server (as specified in the returned glue A record).

5. The `.org` name server will reply with the authoritative name server record for `wikipedia.org`. It will also append the A record for the `wikipedia.org` authoritative name server.

6. The recursive DNS resolver will then ask the authoritative name server for `wikipedia.org` for the lookup of `www.wikipedia.org`

7. The name server for `wikipedia.org` will reply with an authoritative answer for `www.wikipedia.org`, a CNAME record and an A record.

8. The recursive DNS server optionally adds this record to its cache, then returns a non-authoritative answer (an A record) to the requesting application's stub resolver.

2.2 DNS Attacks

2.2.1 Cache Poisoning

While DNS has many positive characteristics, it is well known that it has no integrity, authenticity, or confidentiality guarantees. DNS responses can be modified or completely forged, almost trivially, by an intruder-in-the-middle attack. Indeed, because DNS normally uses UDP, attackers do not even need to hijack TCP connections. The weaknesses of DNS are actually even worse, however, because the existing DNS infrastructure can be used to deliver malicious responses.

Specifically, malicious responses can originate from legitimate, otherwise uncompromised servers through the use of cache poisoning attacks [8]. With standard DNS cache poisoning attacks, a victim can be redirected to a malicious host through the injection of a forged A record. The key idea behind such attacks is that a caching resolver has no way to authenticate received data; it will accept any query response that "looks right." An attacker even has many chances to generate authentic-looking responses as unexpected responses will simply be discarded. This attack is described in Figure 3. An attacker submits a false A record for `www.wikipedia.org` before the DNS hierarchy is traversed and the correct entry is found. The false record will be cached by the resolver until it expires.

While such attacks are problematic enough, in the summer of 2008 Dan Kaminsky described a direct extension to known DNS cache poisoning techniques that made them much more dangerous and reliable [9]. Kaminsky's primary observation was that instead of spoofing the final A record of the destination, an attacker could spoof any one of the query responses in the lookup process. By forging an NS record, an attacker could take over an entire domain, up to and including a TLD such as `.com`. This attack is shown in Figure 4. In the Figure, an attacker triggers a lookup for a non-existent domain `xyz123.wikipedia.org`, while this lookup is being performed,

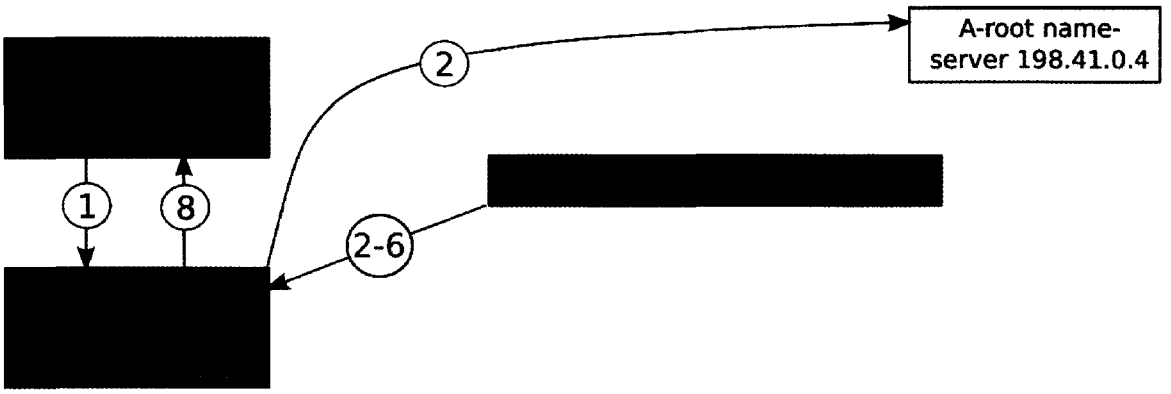

Figure 3: Example of a DNS cache poisoning attack

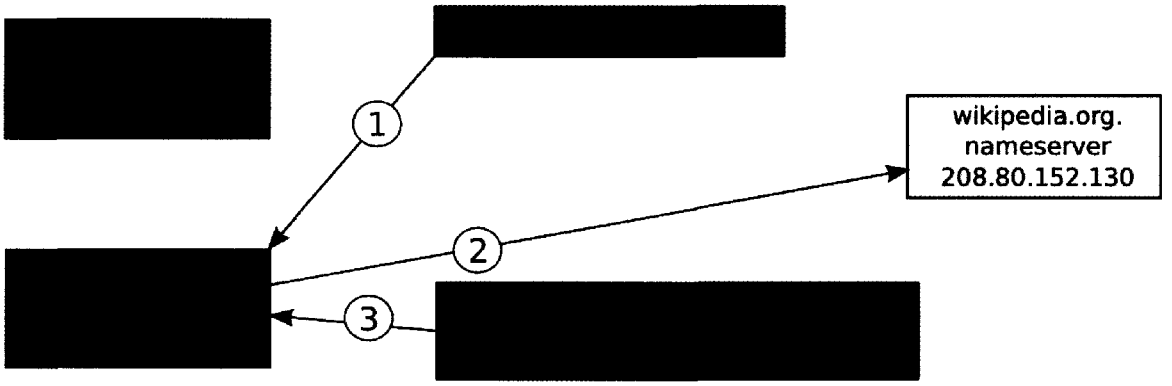

Figure 4: Example of Kaminsky's DNS cache poisoning attack

the attacker submits a false reply and includes two glue records updating the NS server for all of `wikipedia.org` to the attacker controlled server.

Kaminsky also improved the reliability of cache poisoning attacks. Previously, if an entry was cached, attackers would have to wait for the cached entry to expire before they could attempt to replace it; thus, it was relatively hard to poison the cache of popular domains. Kaminsky, however, noted that by querying an obscure subdomain that was not already cached (i.e. `reallyobscure- subdomain.example.com`) an attacker could force a lookup. The lookup would go directly to the authoritative name sever for the domain, assuming the nameserver address was cached. This creates an opportunity for an attack to forge a reply with no answer for the obscure subdomain

but include an additional update for the A record of the authoritative nameserver. The opportunistic caching resolver will update their cache with the false record. This method allows an attacker to repeat the attack many times giving them a greater chance of returning an answer that will be accepted, successfully poisoning the cache. [9]

While response forging is the most common form of cache poisoning attack, another attack vector exists when the reverse lookup tree is the identity verification mechanism. The reverse lookup tree is a special TLD that stores pointer records to hostnames based on IP addresses; it allows the reverse mapping from IP address to hostname. An attacker with an IP of `11.22.33.44` who also controls authoritative name server NS_b can poison the cache of nameserver NS_a. The attacker advertises himself to NS_a as host H_a, then NS_a attempts to verify the IP of `11.22.33.44` as belonging to H_a through the reverse lookup tree. Since the attacker owns NS_b, the reverse lookup of `11.22.33.44` will lead to NS_b, which the attacker can map to hostname H_a. The verification will be complete and nameserver NS_a will resolve hostname H_a to the attacker's IP of `11.22.33.44`. To avoid this attack vector, a variety of alternate mechanisms are used for verification in modern DNS. One common technique to verify the identity of an entity requesting a DNS record update is to verify their ability to receive email at that domain. Many DNS registrars also require that a user login to a web interface with a username-password combination associated with the account.

2.2.2 Other DNS attacks

While cache poisoning is the most common attack on DNS, there are other less common attack mechanisms.

DNS name servers and resolvers are software, and like any other software they contain software vulnerabilities. Specifically, buffer overflows in common DNS server

and resolver implementations have been found and have potentially allowed remote code execution [10, 11].

DNS servers are also frequent targets of Denial of Service (DoS) attacks. A DoS attack at common domains, TLDs or even root servers can affect large numbers of Internet users, and have been common targets [12,13]. Since an IP address is necessary for any IP communication, a denial of hostname translation to IP address effectively stops all IP communication. DoS attacks against DNS servers are typically attempted through resource exhaustion, frequently targeting limited bandwidth.

DNS rebinding is an alternate attack mechanism; however, it does not attack the DNS system specifically [14]. The web browser security model only allows script code to communicate with its source, or same origin, domain; DNS rebinding allows this restriction to be circumvented. Malicious code is executed on a machine, generally in a web browser page; however, a page can only communicate with its source domain. Since the attacker can own the domain, they can then modify the DNS entry, mapping the source domain of the malicious code to a new target IP address. The web browser then rebinds the malicious code's source domain to the new IP address (the attacker's chosen target IP), allowing the communication with that address. The target is frequently an internal IP address which an attacker could not access directly. This attack is frequently performed by setting a short TTL value, requiring a frequent lookup of the IP address and opportunity to select a new target.

2.2.3 Hardening DNS

To prevent chache poisoning attacks, DNS must provide sufficient randomness so that an attacker cannot feasibly forge DNS replies. In recent years DNS has added other sources of entropy to strengthen the very weak response authenticity checks. There are two primary and a third optional source of entropy used by modern DNS servers since Kaminsky's cache poisoning was disclosed in 2008. The first source is

Field	Calculation	Bits of Entropy	Total	
Query ID	$log_2(65536)$	16	2*31	3*43
Source Port	$log_2(32768)$	15		
Capitalization	12 letters providing one bit	12		

Table 1: Possible sources of entropy for a lookup of `www.google.com`

the 16-bit query ID, which is randomized on modern DNS servers. Previously, DNS servers used a predictable sequence of query IDs. The second is the source UDP port; only responses to the query's originating port will be accepted [9]. The entropy gained depends on the implementation and the number of ports available. On most machines the first 1024 ports are reserved; all remaining ports would provide 15.977 bits of entropy. Many common DNS resolver implementations use approximately 32000 ports, providing 15 bits of entropy. The third potential source of entropy is the query string. A resolver can randomly change the case of each letter in a query, adding one additional bit of entropy for each letter. Queries will only be accepted if the capitalization of each letter matches the original query [15]. We have anecdotally observed that many servers permit this behavior, but few resolvers implement it. A example of common entropy values is shown in Table 1.

2.3 DNS Security Extensions

DNSSec is a set of extensions to DNS that are designed to authenticate and protect the integrity of DNS responses through the use of public key-based digital signatures. The design of DNSSec started in the mid-1990s, motivated by the growing recognition of the dangers of DNS cache poisoning [8]. The first version of DNSSec was officially published as RFC 2065 in January 1997 [16]; an improved version followed in RFC 2535 (March 1999) [17]; after trial deployment experience, the current version of

RRSIG records contain digital signatures for other resource records. Each regular DNS resource record should have its own RRSIG record (except for the DNSKEY record).

DNSKEY records contain public keys for verifying RRSIG records. Note that a zone may have multiple DNSKEY records.

DS or designated signer records, store cryptographic hashes of the public keys of a zone's children. DS records are what allow `.com` to specify what key should sign `example.com`'s records, thus enabling trust chains. DS records should be signed with a corresponding RRSIG record.

NSEC and NSEC3 records provide authenticated denial of existence. NSEC provided this information by specifying two hostnames between which no other hostnames exist. NSEC allows attackers to enumerate the hosts in a domain; NSEC3 [21] addresses this problem by specifying the ranges in terms of hashes of domain names rather than the domain names themselves.

Figure 5: The additional resource records for DNSSec

DNSSec was finalized in RFC 4033-4035 in March 2005 [18–20].

2.3.1 Architecture

DNSSec allows domain owners to digitally sign resource records and resolvers to verify authenticity and integrity. The additional information is stored in new types of resource records. The decision to extend DNS's set of resource records ensures DNSSec's backwards compatibility with standard DNS and allows for incremental deployment across the Internet. The additional resource records are described in Figure 5.

To authenticate a resource record, a resolver first checks that the RRSIG signature on the requested record was generated by the zone's key, as specified by its DNSKEY record. To make sure the DNSKEY record has the right value, the corresponding DS record of the zone's parent is checked to see if it has the hash of the DNSKEY key. Then, to verify the DS record is genuine, its RRSIG has to be checked against

the parent zone's DNSKEY public key. This process repeats until we get to the root zone; here, we assume the resolver already knows the authentic public keys belonging to the root name servers.

Note that DNSSec clearly defeats cache poisoning attacks since the responses' authenticity and integrity can be cryptographically verified. Responses can only be forged if the underlying cryptography is broken. Because resolvers will only forward or cache authentic responses, the resolver's cache would remain uncorrupted.

2.3.2 DNSSec Trust Model

DNSSec uses a trust model that leverages the hierarchical structure of the domain name system. Trust is delegated from parent to child entity according to their dot separated hierarchy. For instance, the pubic key (DNSKEY) for `example.com`, is signed by `.com`; the DNSKEY for `.com` is signed by the root servers. The root servers' public key should be distributed to each resolver through some secure out of band mechanism, likely with DNS resolver software.

DNSSec even allows for hierarchical trust delegations within a zone. A zone may have as many signing keys (DNSKEY records) as they wish, and those keys can be divided into separate functions, e.g., Zone Signing Keys (ZSK) and Key Signing Keys (KSK). A single key signed by a parent entity (with a DS record) may be used to sign many KSKs, which in turn may sign many ZSK, which in turn will sign many records.

DNSSec thus follows a strict hierarchical trust model with multiple rooted trees [22]. Each lookup begins at one of many roots for which the public key is known, then trust is delegated down through the DNS tree hierarchy. Trust cannot be delegated across distinct domains. One notable case occurs with CNAME records. An entity may own the same name under different TLDs such as `google.com` and `google.ca` where one points to the other with a cross domain CNAME record. For example,

the canonical name for `google.ca` could be `ca.google.com`. This still follows the strict hierarchical model, after the lookup of `google.ca` revealed the name under a different TLD, a second hierarchical traversal for `ca.google.com` would be necessary.

DNSSec provides two characteristics of secure communications: integrity and authentication. The integrity is provided through the use of cryptographic signatures for each resource record. The strength of the integrity verification is the same as the strength of the cryptography used.

The second characteristic of secure communication is authentication, the verification that the update to a DNSSec resource record is provided by the entity we believe. Authentication is built by chaining trust from a third party parent entity, likely a domain registrar. Our confidence in authentication depends on the third party, both on our trust in that entity, and in their process for verifying DNS updates are from the correct entity. The verification process for DNSSec updates remains to be seen as it is slowly seeing real world deployment. However, registrars have been verifying identity for updates to DNS records for many years. One common technique to verify the identity of an entity requesting an update to a DNS record, is to verify their ability to receive email at that domain. Many DNS registrars also use a remote login system where a user provides a username-password combination for identity verification.

In July 2005 the Internet Corporation for Assigned Names and Numbers (ICANN), released a report investigating the high occurrences of domain hijacking [23]. Their conclusion contained two immediately relevant findings:

- "Failures by registrars and resellers to adhere to the transfer policy have contributed to hijacking incidents and thefts of domain names."

- "Registrant identity verification used in a number of registrar business processes is not sufficient to detect and prevent fraud, misrepresentation, and impersonation of registrants."

Since that report was released in 2005, there have continued to be numerous high profile domain hijacking incidents, including of ICANN itself [24–26]. The result of a domain hijacking is more severe than an effective cache poisoning attack; not only will many users be redirected to an unintended host but the entity whose domain was hijacked will also be negatively affected. The vast majority of domain hijackings are caused by insufficient identity verification when updating DNS information. While we don't have a clear definition for sufficient identity verification, clearly the inability to prevent domain hijacking is evidence of insufficient verification.

DNSSec records are one more instance of DNS records which attackers modify during many domain hijacking attacks. Without changing the system in which identity verification of users updating DNS records is performed, domain hijacking attacks will continue. Further these attacks may be more effective, because the user could have higher confidence in the DNS result because it is "secured" with DNSSec. Trust third parties such as Domain Registrars are a potential source weakness in DNSSec if they fail to consistently provide sufficient identity verification.

2.3.3 DNSSec Security Limitations

When properly deployed, DNSSec does guarantee integrity and authenticity of DNS data. Even if we assume that DNSSec is used optimally, however, that does not mean that we get what we really want, namely secure communications. Attackers have two basic strategies for circumventing DNSSec: they can attack from below or from above.

Attackers can get "below" DNSSec by targeting other aspects of network communication. DNS, at best, tells a computer the IP address of another computer.

An attacker who mounts an intruder-in-the-middle attack (using for example, ARP spoofing on a local network [27] or BGP route hijacking [28, 29]) can potentially take over any IP address they desire. Thus, DNS data can be completely secure yet attackers can still make victim machines connect to malicious hosts. Targeting the actual IP communications with the host instead of the DNS lookup, an attacker easily circumvents any security gained through DNSSec.

Of course, all of this assumes that the DNS hostname we are attempting to resolve *is the right hostname.* Attackers can attack "above" the DNS layer. Phishing attacks have demonstrated that users are easily fooled into visiting malicious sites via links containing fraudulent hostnames [30]. Users do not understand the significance of hostname components and their ordering. Indeed, why shouldn't an average user visit a URL with `www-mybank.secure.com` when their bank is really located at `www.mybank.com`? A proper explanation requires a basic understanding of the hostname hierarchy—something that we cannot expect regular users to know about.

We also believe there will be further usability issues with DNSSec, along the same lines as SSL (discussed in Section 3.2.4). DNSSec administration requires careful generation, management and storage of keys. This is a difficult task for which many DNS administrators are not qualified, so there will inevitably be many DNSSec validation failures. How will the status of DNSSec validation be conveyed to the user? The two most likely options are a soft or hard failure. A hard failure would prevent a user from accessing the resource, which would result in a DoS condition. A soft failure would provide users with error messages and allow them to choose the appropriate action. This is the same situation as we see with current SSL verification failures. Users are prompted with error messages (shown in Figure 7), the majority of whom do not understand or will choose to ignore them, reducing the effectiveness of SSL.

2.3.4 Discussion

DNSSec only secures one small aspect of communications, the hostname to IP address translation. It provides no guarantees about the integrity or authenticity of all subsequent communications. DNSSec takes the approach of securing the naming translation of a remote host, while identifying the correct name in the first place can even be problematic (i.e. in phishing attacks). It also secures them based on third party trust delegators which have struggled to sufficiently verify identity when updating DNS records, compromising the trust in authentication. Further, DNSSec will likely exacerbate security usability issues with more warning messages.

Chapter 3

Secure Sockets Layer and Transport Layer Security

In this chapter we discuss how hosts are identified in SSL protocol and its origins. We analyze its trust model and discuss vulnerable points in SSL.

3.1 SSL

Secure Sockets Layer is a protocol for securely communicating with remote hosts. SSL uses cryptography to provide integrity and confidentiality and a system of trusted third parties to provide authentication with public key certificates.

3.1.1 History

Secure Sockets Layer is a protocol developed by Netscape Corporation to provide end-to-end security for any application layer network protocol. One of the primary motivating factors behind SSL was the rapid growth of the Internet and the desire for secure communications enabling online commerce. Secure communications were a prerequisite to the development of most e-commerce.

SSL 0.2, known as version 2, was the first version of the protocol publicly released

in 1994 [31]. Version 2 however had multiple security vulnerabilities [32] and was soon replaced by version 3 in 1996 [33]. SSLv3 has seen wide adoption and became the primary mechanism to secure web traffic. In an effort to achieve a more open, non-proprietary and community supported standard, the Internet Engineering Task Force (IETF) designed Transport Layer Security (TLS) based very closely on the SSLv3 protocol [34]. Achieving their goal of community support, the TLS protocols have received many revisions including two version updates [35, 36]. Despite many revisions, the TLS protocols very closely resemble the original SSLv3 protocol; the primary differences are upgrades to the cryptography supported. Thanks to the similarity of the protocols, all four are interoperable. The negotiation of a secure connection chooses the highest supported version by both client and server. For the remainder of this thesis, we refer to the suite of protocols SSLv3, TLS 1.0, 1.1 and 1.2 generically as SSL.

3.1.2 SSL architecture

An SSL connection consists of two phases. The first establishes the parameters with which the client and server will communicate and is called a handshake. The second, after a successful handshake, is the communication stage in which application layer data is encrypted and transmitted.

The handshake is used to establish the parameters through which the two parties will communicate; the first parameter is the protocol version to be used. The client sends the server a list of the versions (e.g. SSLv3, TLS1.0) it supports then, the server chooses the highest version it also supports to continue communication. The server shares its public key, encoded in a certificate with the client. After authenticating the public key of the server, the client and server establish a shared secret. The client may also provide a public key to allow two-way authentication, though this is rarely utilized in practice. The final step of the handshake is to use the shared secret to

generate session keys. Session keys are symmetric cryptographic keys which will be used to encrypt the application data during communication. Symmetric cryptography is used to improve performance and also security because unique keys are generated for each session.

SSL operates at layer 6 in the OSI model [37] and therefore requires a reliable transport layer to operate on top of, frequently TCP. One of the desirable characteristics of SSL is that it is independent of the application layer. Our discussion of SSL assumes HTTP web traffic (HTTPS), however other application layer protocols are also applicable.

3.1.3 SSL Certificates

A digital certificate is a document binding identity information to a public key. Typically a third party digital signature is used to prevent tampering with the certificate. Note that the verifier must have an authentic copy of the signer's public key to detect tampering. Certificates are usually encoded using the X.509 standard [38]. Identity information contained in a certificate varies however common fields include the organization name, country, city and critical for web traffic, the domain name, typically stored in the Common Name (CN). Web browsers use this domain name as the identity information when communicating to remote hosts. There are currently three types of SSL certificates widely in use, described in Figure 6.

3.1.4 Trust Model

SSL, as with DNSSec, uses a authortative trusted third party model with multiple roots. SSL CAs can delegate signing authority, however the the depth of delgation is rarely more than 2. One survey of SSL sites on the Internet found 44% had a depth of 1 (signed directly by a CA) and 55% had a depth 2 [40].

Self-signed - This certificate provides no inherent authentication, the certificate is signed by the private key corresponding to the public key in the certificate itself. In the absence of an out-of-band mechanism for establishing trust in the certificate, a self-signed certificate is equivalent to a certificate signed by a non-trusted third party. There is no universally agreed upon process for obtaining trust in the corresponding public key.

Standard CA Certificate - This certificate is signed by a certificate authority (CA), a trusted third party who is paid a fee for their signing services. The process used to verify the identity of a requestor depends on the CA. Some CAs offer different classes of standard certificates with different levels of verification.

Extended Validation (EV) Certificate - Extended Validation is a set of more stringent guidelines for verifying the identity of an entitiy in an SSL certificate requestor. The verification process and signing is still performed by a CA. [39]

Figure 6: List of common SSL certificate types

A user's confidence in the authentication provided by SSL varies and is partly decided by the trust in the signing party. Self-signed certificates are not signed by trusted third parties; they do not provide authentication unless they can be verified by some out-of-band means. SSL connections to unverified self-signed hosts are only secured in terms of confidentiality and integrity, not authenticity, making them vulnerable to intruder in the middle attacks. Standard and EV certificates are signed by trusted third parties and therefore provide authentication.

Our confidence in authentication depends on trust in the third party CA, and the probablility that a signed certificate is erroneous or fraudulent. Standard and EV certificates are signed by CAs whose root certificates are generally distributed with web browsers; therefore, the initial decision to trust a CA is made by the browser manufacturer. CAs may be indifferent or actively hostile to the interests of the user. Recently, organizations accused of distributing malware and spying on Chinese citizens have been granted a root CA certificate distributed with Mozilla's Firefox browser [41,42]. This raises the question, should we really consider CAs to be trusted

third parties?

A user's trust in a third pary CA may also vary over time, especially if some event such as a compromise of the CA occurs. It can be a difficult task for a user to manage the significant list of trusted certificate authorities; even if a user's trust in a CA diminishes they may not be able to remove the root certificate from their trusted set. Moxie Marlinspike has described this inability to modify sources of trust as a (lack of) *trust agiligy* [43].

As with DNSSec, the other factor in the confidence of SSL authentication is the verification, by the signing CA, of the identity presented in a certificate. Certificate Authorities have processes for verifying identity on standard CA certificates [44]. Many CAs use a test verifying the capability to receive email at the domain in question, and, some reportedly perform no verification at all [45]. The Extended Validation certificate specification defines a set of guidelines for verifying the identity of an organization. Some of the verifications include "the legal, physical and operation existence", "the entity has exclusive rights to the domain" and "the entity has properly authorized the issuance of an EV certificate" [39]. As with any process, it can be subject to human error or social engineering [46, 47]. However, these verifications should provide a higher confidence in the authentication provided by an EV certificate.

Many domain registrars are also certificate authorities, such as Verisign [48] and GoDaddy [49]. In Section 2.3.2, we discussed how domain registrars do not consistently perform sufficient verification when updating DNS entries, which result in domain hijacking. Given that many domain registrars are also CAs, we believe that the same can occur during the certificate verification process; this is supported by the numerous publicized occurrences [44–46, 50].

While the authentication phase of SSL can have varying levels of assurance, the communication phase is generally secure. Once a public key is available, the encryption parameters are negotiated. For all types of certificates, data will be encrypted

between client and server ensuring confidentiality and integrity as strong as the cipher in use. However, there are attacks against the negotiation phase and the some ciphers which are described in Section 3.2.1.

In summary, SSL can provide strong end-to-end security when the confidence in authentication is strong. Authentication, however, can be difficult, when attempting to establish trust in a CA or having confidence in their verification process.

3.2 SSL Vulnerabilities

Apart from insufficient identity verification, there are several vulnerable aspects to the SSL system, among them being usability. There are also several technical vulnerabilities in the SSL system and the underlying cryptography.

If an attacker can compromise SSL through any of these vulnerabilities the most likely attack vector is an intruder in the middle attack. In this scenario, an attacker may listen to or inject communication into the SSL connection; this compromises authenticity, integrity and confidentiality.

3.2.1 Cryptographic Weaknesses and Downgrade Attacks

SSL and its multiple versions support a wide variety of cryptographic algorithms and parameters. One must consider the key exchange algorithm, the key size, the hash function, and the symmetric encryption function (or cipher) used. Determining which combinations of algorithms and parameters is secure is beyond the scope of this thesis, but we refer the reader to *Cryptographic Strength of SSL/TLS servers: Current and Recent Practices*, Lee et al. [51]. A weakness in any of the cryptographic components could lead to a compromise of the entire connection. Beyond choosing secure cryptography, the choice of algorithms and parameters must be supported by both the client and server. This choice is performed in the negotiation phase of the

SSL connection.

A class of attacks known as downgrade attacks focuses on attacking the negotiation phase. Many SSL clients and servers support a wide variety of algorithms, likely to ensure wide compatibility; however they often include some which are not secure [40]. A downgrade attack attempts to force a client or server to use a weak choice of algorithm or cryptographic parameter so that an attacker may compromise the communication.

3.2.2 Null Prefix Attacks

One of the most significant recent attacks against SSL was the null prefix attack.

In 2009 Moxie Marlinspike disclosed a vulnerability in the way many browsers and operating systems handle SSL certificates. The error occurs when the browser or operating system misinterprets the common name (CN) field, the field used to identify the domain for which the certificate is valid. By inserting a null character (\0) in the CN field, only the prefix before the null will be read. For instance, a field with a value of `paypal.com\0mydomain.com` will evaluate to `paypal.com`. Such a certificate could be signed and verified by CA if an attacker owned `mydomain.com`. [52]

There are two possible extensions to this attack. The first extension is to use a wildcard certificate, which was only found to work on some platforms. Such a certificate would look like `*\0mydomain.com` and the wildcard character (*) would be valid for **any** domain. The second extension is to attack the certificate revocation protocol. Most browsers verify that a certificate is still valid through an Online Certificate Status Protocol (OCSP). The effectiveness of this protocol can be marginalized though an intruder in the middle attack [53].

3.2.3 Software Vulnerabilities

As with all software, SSL client and server implementations have contained vulnerabilities. These vulnerabilities, such as buffer overflows, can be exploited to compromise an SSL connection or potentially execute arbitrary code.

3.2.4 User Interaction

There are several reasons why average users struggle with SSL. They have difficulty understanding, or are simply uninformed about many of the concepts relating to SSL including certificates, authentication and encryption [30]. They also have trouble interpreting the relevant security information presented to them within a web browser.

Browser Security Cues

SSL can provide strong security, however the state of that security needs to be conveyed to the end user; this is the browser's responsibility. The browser must communicate whether a page is secured with SSL and also the type of certificate used. To convey the state of a connection, browsers use visual cues. There are three primary cues which are common across most browsers.

The Lock Icon its presence in the browser chrome indicates a secure connection.

https preceding the url (as opposed to http) indicating a secured (HTTP+SSL) connection.

Colouring Indicators in the URL bar; these vary from browser to browser. An example can be seen in the red "https" in Figure 7 warning of a self-signed certificate.

These cues have not been successful at conveying security information. One of the underlying reasons appears to be *the unmotivated user property* as described by

Witten and Tygar [54]. This property states that security is a secondary goal for most users. When a user is browsing the web, their primary task is to access a web page and it is easier for them to assume security is sufficient and present than to explicitly verify it. Users may also lack the knowledge to understand the browser cues or levels of security; acquiring that knowledge is also less likely for an unmotivated user.

A users' lack of attention to security cues has also been directly demonstrated by Dhamija et al. as they investigated why users fall victim to phishing attacks [30]. They found three reasons directly related to browser cues.

Lack of knowledge of security indicators when users do notice the browser cues, they often do not understand the level of security they indicate

Lack of attention to security indicators users simply do not pay attention to the browser cues available to them

Lack of attention to the *absence* of security indicators when users should check for browser cues, they do not

Similar results were demonstrated by Schecter et al. in the context of online banking, particularly, that users do not notice the absence of indicators [55].

The lock icon is particularly problematic because users do not understand the difference between the browser chrome and the content of the page. A lock icon (or other security claim) in the content of the page is easily manipulated by the page creator and is not a browser indication of security. The page owner may also choose a favicon resembling the SSL lock icon, which will be placed in the url bar of the browser chrome, further confusing the matter. Also, the placement of the lock icon within the browser varies between browsers and version. [30, 56]

Two separate user studies have found that the visual cues for Extended Validation certificates are not effective. Jackson et al. found that EV certificates do not help

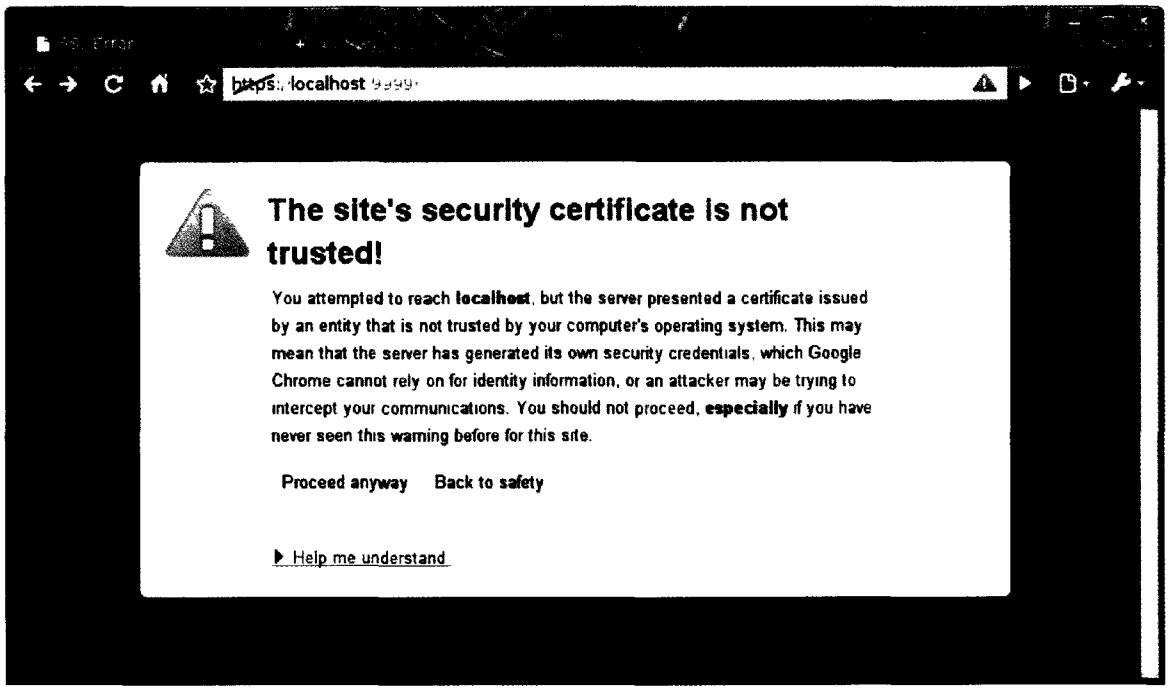

Figure 7: A SSL warning message presented to a user's a web browser

users detect picture in picture phishing attacks, in part because they do not notice a difference between page content and browser chrome [57]. Another user study by Sobey et al., found that users do not notice some browser cues that convey the use of an extended validation certificate [58].

Browser SSL Warnings

Web browsers also typically present warning screens when accessing a site that has a self-signed certificate as shown in Figure 7. All these warnings have an option to cancel navigation to the page or an option to continue. The option to continue to the self-signed page often requires a series of steps in order to discourage the user. Despite a variety of phrasing and design of the messages, the content of these messages is largely irrelevant. Most users do not read or understand the messages; they simply click though to the page. The frequent appearance of SSL warnings which a user does not understand or read has conditioned users to ignore them all together. [30, 59]

3.3 Discussion

SSL is the most widely used secure communications protocol on the Internet. It provides strong cryptographic security in terms of confidentially and integrity. For authentication, it uses certificate authorities which a user may or may not trust. The CA identity verification process appears to be improving due to EV certificates, however recurring events show that the identity verification process is still vulnerable to attack.

The larger problem appears to be users' interaction with SSL. They do not understand the concepts providing the security and are further confused, unaware or negligent of security cues in the browser, including the significane of error messages. They also do not understand the hostnames used to identify remote host in certificates. There has been progress in this area as some studies [58, 60] have shown users will take more notice of improved browser cues.

Chapter 4

Related Security Technologies

In this chapter we discuss other technologies for securely connecting to remote hosts. Some of these technologies: SSH, DNSCurve, PGP and IPsec, are less common than SSL and DNS(Sec) however they have seen some level of use. Other technologies such as Perspectives, ITrustPage, Web-keys and Trust Management for Humans, have seen little or no use, and are primarily research tools. We provide an overview of these focusing on their trusts models.

4.1 Secure Shell

The Secure SHell (SSH) is a tool to provide a secure network tunnel to a remote host over an insecure network. Its primary usage is for remote administration of servers; however, it can also be used to tunnel protocols and forward TCP ports or X11 sessions [61].

SSH was developed to replace insecure protocols such as rlogin, remote shell (RSH) and telnet, which had been the norm for connecting to and administrating remote machines. These protocols did not natively include any sort of encryption. Due to this insecurity, attackers could easily take over remote administration connections and gain complete control of a machine. SSH uses strong cryptography to secure

communications.

SSH is very much like SSL architecturally; it operates on top of a standard TCP/IP stack and allows encapsulation of many different application layer protocols or types of data. It also operates in a client-server model, which the client initiates connections to the server.

4.1.1 Trust Model

SSH provides secure, end-to-end communication: authenticity, confidentiality and integrity. As with SSL, the confidentiality and integrity are provided through strong encryption and integrity checksums. The session is initiated by the client, and parameters are negotiated with the server include cipher and compression algorithms. Also as with SSL, the public key of the server is presented for authentication. This public key is then used to securely generate a symmetric session key with which the data will be encrypted. SSH performs two way authentication: the client presents proof of some secret, generally a password or private key, to the server which is also authenticated.

Unlike SSL, when a public key is presented for authentication it is not in a signed certificate. The verification of the public key is left to the discretion of the user, as shown in Figure 8 This verification is performed manually, if at all. Once the public key for a given host is verified, the user may then store this key, binding it to the hostname. Any subsequent connection to that host will not require any further user interaction during to validate the key. The public key and hostname pair may also be delivered via an out of band mechanism and installed in the SSH client by an administrator.

This trust model is known as Trust on First Use (TOFU) and is analogous to a self signed certificate with SSL [62]. TOFU provides strong authentication once a key

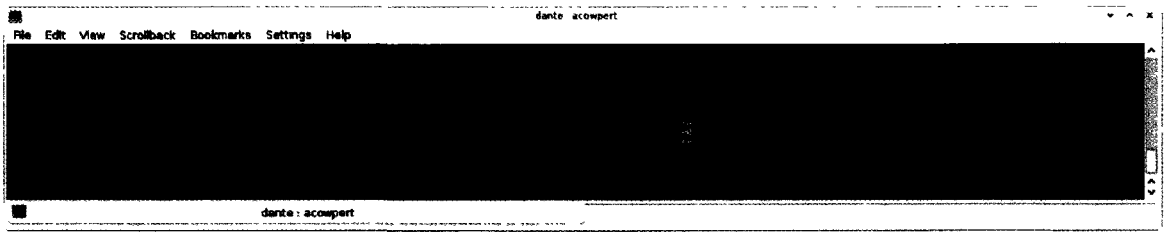

Figure 8: A user is prompted to verify an SSH public key fingerprint

has been established. The weakness, however, is upon first delivery; an intruder-in-the middle could easily intercept the transmission and replace it with their own key compromising not only the current session, but all future ones.

SSH typically presents a public key and not a signed certificate for authentication; however, this does not need to be the case. SSH could easily be extended to accept certificates signed by certificate authorities such as those used by SSL. This would give SSH the same trust model as SSL, both positive and negative aspects.

4.2 Perspectives

The problem of establishing trust on the first connection of SSH, self signed SSL or other TOFU systems, in the face of an intruder in the middle was tackled by Wendlandt et al. in their paper *Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing* [62]. Their primary observation is that the public key offered by a service should remain the same over time and an intruder in the middle will likely only have one network vantage point from which they can attack. To leverage this, they proposed a decentralized model of notaries which observed public keys of TOFU network services. They called this system *Perspectives*. The notaries observe how a public key offered from a TOFU network service changes through time and space (network topology). The notaries were diverse in network location and would probe network services for their public key. This is based on the assumption

that an intruder in the middle would only be able to compromise a limited number of paths, and notaries from diverse network locations would observe this difference. Perspectives also probed network services and noted the changes in public keys over time. When a client wished to connect to a new TOFU system, they could query a number of notaries to verify the public key offered by the TOFU service. Perspectives does not perform the key verification, but provides a user or user defined policy with information to make an informed decision on the initial trust; however in the initially implemented system, the decision is generally automatically delegated to a user created policy.

4.2.1 Trust Model

Perspectives is not a complete system; it is a potential addition to other TOFU secure communication systems such as SSL with self signed certificates and SSH. The trust model, when evaluating Perspectives must be considered in the context of the TOFU system being supported. SSL and SSH both provide strong cryptographic mechanisms, data integrity and confidentiality are as strong as the underlying chosen ciphers. Authentication is also strong, once a user has verified that a public key belongs to the intended target. As with other systems, the verification is challenging and this is the aspect Perspectives aims to improve.

Perspectives uses multiple notaries to observe how a public key changes over time and space. This is a distributed trust model; no single notary is trusted more than others. The public key is essentially endorsed by multiple third parties, although they make no assertions of correctness. A user, or user defined policy, then uses the collective endorsements to decide on trust. This is a distributed trusted third party model.

There are some similarities between Perspectives notaries and traditional SSL Certificate Authorities. Communication with both notaries and CAs are secured by

pre-installed public keys. Perspectives has not seen wide deployment and usage, so the operation of notaries remains an open question. However as with CAs, one must question whether they trust the operator of a notary.

However unlike CAs, notaries do not provide verification; they simply report the public key they have observed. The observation and recording of a public key is a simpler task and is less prone to the errors occasionally made by CAs while verifying identity. A notary also makes no assertions that the public key they have observed is correct.

The system Perspectives has proposed is a decentralized trust model. Information is gathered from many diverse and equally (un)trusted notaries. If this truly is the case, then it provides users with diverse information to verify a public key, which is a powerful tool. There also exists risks for this trust model to stray during implementation. Companies who have traditionally controlled CAs would surely have an interest in maintaining that position, and it could be easy for many notaries to be controlled by very few entities, eroding the decentralized nature. Further, because communication with notaries is secured by pre-installed public keys, the risk is heightened; the entity who pre-installs the public keys will control the default set of notaries. It would not be obvious, even to diligent users, if the trustworthiness of the notaries change.

Perspectives relies on its underlying secure system, and in the case of SSL, Perspectives may further complicate the landscape. Users are already confused by SSL error messages, and this adds another option to the "levels" of verification [30, 59]. The correctness of hostnames, whether a user is connecting to the intended resource, remains a problem in both SSL and SSH.

The Perspectives system improves the strength of the TOFU model by providing users with additional information before deciding to accept TOFU credentials. Although there are many potential pitfalls if this system were to see wide deployment,

it can provide useful information when trying to establish a secure connection.

4.3 DNSCurve

DNSCurve is an alternative to DNSSec, for bringing security to the DNS layer. As we have previously discussed, DNS provides an efficient and highly available lookup system to translate hostnames into IP addresses (Section 2.1). DNS has many positive characteristics, but it does not provide any data integrity or authenticity. DNSCurve addresses this problem by signing and encrypting all DNS packets, queries and replies. DNSCurve uses high speed, high security elliptic curve cryptography [63].

4.3.1 Advantages of DNSCurve

DNSCurve arguably provides better DNS security than DNSSec. DNSSec only provides integrity and authentication to DNS replies. DNSCurve provides confidentiality, authenticity and some integrity to both queries and replies.

DNSCurve also strives to improve availability. A validation failure for DNSSec may result in some sort of error message, or a dialog where a user may choose to accept the false entry; the valid DNS reply will likely be lost. DNSCurve quickly discards forged replies and continues to wait for the valid reply, improving availability in the face of attempted forgery.

DNSSec also provides a powerful platform for Denial of Service (DoS) amplification attacks, where small size DNS queries have relatively large replies. A standard DNS query provides a small amplification factor, while DNSSec greatly increases amplification factor to the order of hundreds [64]. DNSCurve packets are larger than standard DNS, but they are so in both the query and response so the amplification factor is smaller than even standard DNS.

4.3.2 Disadvantages of DNSCurve

DNSCurve, while arguably providing better security than DNSSec, does still share in many of the same issues. These including deployment and maintenance challenges for administrators and significant increases in resource usage.

DNSCurve is designed as extensions to the standard DNS system, specifically public keys are stored in the hostname hierarchy, and all packets are appropriately encrypted. This will require substantial effort for deployment, arguably more than DNSSec due to fewer available resources. Administrators will also be charged with generating key pairs, securing the private key while maintaining its accessibility for cryptographic functions, and developing policy as to who can access the which private keys. DNSCurve and DNSSec administrators will need to become Public Key Infrastructure (PKI) administrators, a task for which most do not have the time or training.

DNSCurve also requires significantly more resources than DNS. DNSCurve uses high speed elliptic curve cryptography (ECC) to encrypt DNS packets. Each query and response must be encrypted and decrypted before it can be consumed by a user, and there may be many queries and responses if a full recursive lookup is required (i.e., Figure 2). Although ECC is faster than more traditional RSA, each cryptographic operation will still introduce some latency and additional CPU cycles into each DNS lookup [63, 65]. DNSCurve packets also will consume more bandwidth because each packet is larger, due to encryption and padding. All these increases in resources appear to be manageable. DNSSec does not require encryption and decryption of each packet, only verification of the final trust chain.

4.3.3 Trust Model

DNSCurve uses the hierarchy of DNS to provide a hierarchical, authoratative trusted third party trust model. Much the same as DNSSec, parent entities verify the children based on the DNS dot hierarchy. For instance, the authenticity of the pubic key for `example.com`, will be vouched for by a `.com` registrar through a digitial signature; that `.com` registrar's key is similarly vouched for by the DNS root. The difference with DNSCurve is due to the shorter ECC key lengths, keys are stored *as part of* the hostnames. A host with the name `ns1.example.com` and the public key `uz5x...hyw`, would then be known as `uz5...hyw.example.com`. The DNS record from the parent should be delivered securely, with confidentiality, authenticity and integrity checks, so the key of the shares the same security guarentees.

As with DNSSec and SSL, the confidence in the authentication provided by the parent entity or registrar is dependent on two things, the trust for that entity and their verification process. Our trust for the registrar will be the same as with DNSSec and SSL as they are the same entities. It may also be dependent on the particular zone, e.g., `.com` may be more trusted than `.org`.

The process for updating a DNSCurve record with a registrar or parent entity is exactly the same as the process for updating a DNS record, because the hostname contains the DNSCurve key. As we have previously discussed, domain registrars have not been successful at consistently performing sufficient verification when updating DNS records, which have resulted in domain hijacking. As with DNSSec, we do not believe that DNSCurve will provide motivation for registrars to fix this issue and consistently provide sufficient verification for all DNS updates.

DNSCurve provides confidentiality by encrypting DNS queries and replies; however, the effectiveness of this measure has been questioned. Because only the payload is encrypted, an attacker can easily compromise the confidentiality by infering the

authoritative nameserver from the IP address if a recursive query is being performed. For example an attacker that sees an encrypted DNS reply (determined by UDP source port 53), from an IP of `134.117.1.11` can determine that that IP is the primary authoritative name server for `carleton.ca`. Because of the heavy caching in DNS, the recursive lookup necessary to compromise the confidentiality man not occur.

DNSCurve does not provide a strong integrity verification mechanism. All packets are encrypted, which makes controlled modification difficult however an attacker can possibly corrupt the payload in a way that cannot be detected. There are no message digests, checksums or signatures to verify integrity.

4.4 PGP/GnuPG

Pretty Good Privacy (PGP) is a suite of multipurpose cryptographic software applications which can be used on many types of data and includes a wide variety of cryptographic algorithms and operations.

PGP was originally developed in the early 1990s by Phil Zimmermann [66]. Due to the increasing popularity of PGP and worries about the licensing of proprietary cryptographic algorithms, a compatible open standard was developed called OpenPGP [67]. As a result of the OpenPGP standard, GnuPG (GPG) was then developed and licensed under the GNU General Public License (GPL). Both PGP and GPG are software applications that provide cryptographic services based on the OpenPGP standard. There are a number of other applications that conform to the OpenPGP standards, including email clients and file encryption tools.

PGP/GPG can be used to provide confidentiality, integrity and authenticity to any static data blocks (as opposed to data streams) but is most commonly used with emails and email attachments.

4.4.1 Trust Model

The OpenPGP trust model relies on users' manual verification of a certificate to perform authentication. OpenPGP certificates, also known as Public Key Packets, differ from trusted third party certificates by potentially having signatures from more than one person or entity. These certificates may be signed by many people who, in signing, endorse the binding of identity information and the public key presented in the certificate.

OpenPGP uses Web of Trust (WoT), a distributed trust model. Once a user establishes trust in another's certificate, it will be added to their keyring. The keyring is a set of trusted certificates. Each certificate many be signed by many others. Although a certificate may not be signed directly by a trusted certificate on a keyring, it may be signed by an entity endorsed by one or more of your keyring certificates. This principle can extend to form trust chains of any length. If a user can build a path through from their keyring, through some number of certificates to their target, they can then make an informed trust decision.

In addition to a user having some number of trusted entities on their keyring, they can also associate different trust values with each entity. There are typically four values: untrusted, partial, complete, ultimate. The ultimate level of trust is usually reserved for a user's own signing key. Typically at least two endorsements by partially trusted entities are required to achieve the same confidence as a single endorsement from a completely trusted entity, but this can be customized.

There are many public key servers available to make OpenPGP public keys accessible and facilitate building trust chains.

OpenPGP systems, as with the other security systems we have discussed, provide confidentiality and data integrity through the use of encryption and/or digital signatures. The strength of these attributes is dependent on the strength of cryptography

used.

4.4.2 Usability with PGP

OpenPGP systems present significant usability issues. In 1999 Witten and Tygar published *Why Johnny can't encrypt: A usability evaluation of PGP 5.0* [54]. This paper was one of the first works on security usability and identified it as a significant problem. Only one third of participants were able to successfully sign and encrypt an email using PGP. They attributed much of the difficulty to user interface design and concluded interface design for security software should meet different criteria. Seven years later in 2006, similar results were shown when this study was replicated with current versions of PGP and Outlook Express [68]. Despite years of improvements to PGP and the email client, most users were still unable to sign and encrypt email.

We feel this is due to the complex nature of OpenPGP systems, especially in certificate verification. As has been shown with SSL certificates, users have trouble understanding certificates. This hypothesis is further supported by Garfinkel and Millar in 2005, *Johnny 2: a user test of key continuity management with S/MIME and Outlook Express* [69]. When using S/MIME, a simpler tool to secure email based on CA certificates, significantly more users were able to encrypt and sign email.

While developing an algebra to express trust in certification chains, it was also shown that trust in OpenPGP trust chains can easily be misinterpreted due to hidden dependencies [70].

4.5 IPsec

Internet Protocol Security or IPsec is a suite of protocols for encrypting IP packet payloads. It provides an obvious solution to provide end-to-end security, encrypting the entire IP payload. It was originally designed by the US Naval Research Center

and published in RFC 1825 in 1995 [71], updated again with RFC 2401 in 1998 [72] and most recently, RFC 4301 in 2005 [73].

IPsec encrypts IP packets based on a shared symmetric key called a session key. IPsec requires the client and server to mutually authenticate and then establish a session key. There are a number of protocols supported to perform the authentication and key sharing in IPsec. IPsec is a versatile mechanism that can be used integrated with many different authentication mechanisms; we will provide a brief overview the two principal ones, Kerberos [74] and Internet Key Exchange (IKE) [75, 76].

A Kerberos server acts as a trusted third party to mutually authenticate client and server and establish a shared secret between them. A Kerberos server holds a shared secret with every entity on the network, clients and servers. A client who wishes to connect to a server first connects to the Kerberos server and authenticates through proof of a shared secret, (typically a username-password combination), or a public-private key pair. Once the authentication has been complete the client receives an authentication token (or ticket) from the Kerberos server, the token will grant access to a specific resource or service on the network. When the client connects to the service provided by a server, it presents its authentication token to the server who verifies that token with the Kerberos server. If the Kerberos server verifies the token, access is granted and secure communication will begin. Kerberos is particularly useful in large enterprise networks because a single authentication token can grant access to multiple network resources.

The second technique is IKE. The IKE exchange contains two phases. The first phase sets up the security association or SA and the second performs authentication. In the first phase the the client and server exchange protocol version numbers, supported cryptographic algorithms, Diffie Hellman parameters and nonces. From that information the Diffie Hellman key is computed and a masterkey called SKEYSEED is derived. This master key is used to generate subsequent keys used to encrypt and

provide integrity checks for the second phase. In the second phase the client and server exchange messages that are encrypted and integrity protected. The messages contain authentication information. This may be a pre-shared key, username-password combination, a public key certificate or a combination of secrets. The secrets are securely transmitted using the key negotiated in phase one and the authentication information is verified. Once both the client and server are authenticated, a shared key is derived which will be used to encrypt IP payloads [76].

4.5.1 Trust Model

Despite being a versatile suite of protocols that supports many authentication mechanisms, IPsec generally relies on the same underlying trust model of manual verification. One of the primary differences between IPsec and other manual verification systems we have discussed (such as SSH), is that IPsec relies on establishing these secrets before communication is made.

To successfully use IKE, the public key certificate of the server must be installed in the client before beginning; and in the reverse direction, the username-password combination or public key of the client, must be installed on the server. Alternatively if Kerberos is being used, the Kerberos server must be set up in advance and a secret must be shared with the potential Kerberos client. This set-up constitutes manual verification and binding of secret material to host or user names.

One notable case occurs when a IPsec is deployed with Kerberos. Kerberos servers are generally used within large enterprise networks where the Kerberos server is a third party to the client and server. However, the Kerberos server and the network server are controlled by the same entity, generally an enterprise IT department. For the purposes of our trust model evaluation, this is not considered a third party because it is actually the same entity. It is unlikely, although possible, that a Kerberos server would be operated by a distinct third party.

IPsec arguably provides the most versatile security of any of the technologies we have discussed. The encryption provides strong confidentiality and integrity. The confidence in authentication is also very high because it is performed before communication, by sharing some secret likely through a secure out of band mechanism.

This system however, also provides one of the least usable security solutions. The tasks we have discussed, installing keys or shared secrets and binding them to user or hostnames, requires thorough understanding of the technologies at use. This is not knowledge very many users possess and should only be done by expert users.

4.6 Trust Management For Humans

Trust Management For Humans is a proposed system by T. Close [77], that uses the HTTPSY [78] mechanism to provide resilience to phishing attacks. The technical mechanism behind the HTTPSY extension is to identify a host by the hash of a certificate's public key (used to verify the certificate's signature). This means a link not only contains the standard URI destination, but also a hash of the public key used to verify the site's SSL certificate. Of course, for an SSL certificate verification to be useful, all implementing sites must support SSL.

HTTPSY is part of a proposal to map the hash of public key to a human "petname" as explained in Trust Management For Humans [77]. The petname is a user chosen string that identifies a given site, for example "my money" may be the chosen name for a bank's site. The fingerprint of the certificate is mapped to the petname within a browser's local cache, and the petname is displayed in the browser chrome. If a user was redirected to a phishing site, the proper petname would not be seen.

The same author, T. Close also created a system call Web-Keys, however this primarily addresses a different problem, that of Cross Site Request Forgery

4.6.1 Trust Model

The Trust Management for Humans system provides an extension to SSL and modification to the web browser. As with the Perspectives system, much of the security is dependent on the underlying system, SSL. SSL provides strong cryptographic mechanisms. Data integrity and confidentiality are as strong as the underlying chosen ciphers. Authentication is also strong, once a user has verified that a public key belongs to the intended target.

Trust Management for Humans provides a unique approach by delegating the verification to the user, through the use of petnames. This removes the problem of trusting a third party verifier and the issue becomes one of a user manually verifying. A user must question whether they trust the source of the information, the hash of the public key. This is also the same person providing the link, so if the link is malicious, the hash likely will match. However the verification mechanism is based on previously viewing a public key and associating it with a petname, so an attacker would have to generate a public key (and associated private key), that have a hash collision with the previously viewed public key; a very difficult task depending on the hash function.

This system also introduces a trust on first use (TOFU) element. When a user visits a site for which they haven't chosen a petname, they must decide if they trust this site and if so, choose an appropriate petname. As with other TOFU systems, if an attacker can compromise the first use, all subsequent uses will be compromised. Further, with this system there may be multiple "first use" scenarios, if a user does not set the petname, then each subsequent visit until one is set will expose them to the same first use risks.

Trust Management for Humans provides a user chosen browser text cue, mapped the hash of the public key of a site. As we have previously seen (Section 3.2.4) users

do not always verify browser security cues. It may improve this because it is a user chosen cue; however, this has not been studied.

4.7 ITrustPage

ITrustPage is a browser plugin that validates web page forms to prevent phishing [79]. ITrustPage works by caching a user's web forms; whenever a user reaches a form, the cache will be checked to see if the user has previously validated the form. If a form is unknown, the user is prompted to enter details about the form. These details are used to perform a Google search, if url of the form is found in the top 10 results the form will be validated. If the form is not found, the top ten results are presented to the user who performs a visual comparison. If the user finds a visual match, the original is determined to be a phishing attack and the user will be redirected to the Google found result.

4.7.1 Trust Model

ITrustPage as with other systems we have viewed, provides an extension to aide in the identity verification of underlying systems. This extension does not provide confidentiality or integrity. It does help a user to provide identity verification, although without the use of any cryptographic means, unlike other systems we have discussed.

ITrustPage uses an external information repository to validate a user is at the intended webpage or web form. ITrustPage chose the Google search engine as the information repository, although surely others would be possible. Ronda et al. observed there are two characteristics that make the Google search a desirable choice. First, the site is popular and many other sites link to it. These facts are derived by PageRank algorithm and the site being ranked in the top ten. Secondly, many phishing sites are short lived which makes the result less likely to have been crawled

by Google, let alone achieved a top ten rank [79].

Any verification mechanism relies upon the trustworthiness of the verification provider. In the case of ITrustPage, Google search engine is used as the third party information repository, so the trust of a verification is based on trust for Google. It should also be noted that due to Google's PageRank algorithm, their ranking of a site is not based on identification of the site, it is based on how many other sites link to it. Although strictly speaking, this is a authoratative trusted third party model, if you trust Google as an aggregator the trust is derived from many untrusted third parties, providing a distributed untrusted third party model.

Chapter 5

Trust Models

In this chapter we first define trust. We have discussed many existing systems and have seen that many systems build trust in similar ways. We group the systems together based on how trust is established. From these groupings we see that similar trust models have similar weaknesses. Finally, we discuss the common over-arching problem of successfully identifying remote hosts.

5.1 Finding Trust

We start our discussion on trust with a definition. Most people have an intuitive notion of trust, but a concrete definition is more elusive. We will use the definition provided by Diego Gambetta in his work `Can we Trust Trust?` [80]:

> "Trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action."

Abdul-Rahman et al. observed that there are three primary points made by this definition [81].

1. Trust is subjective.

2. Trust is affected by those actions which we cannot monitor.

3. The level of trust depends on how our own actions are in turn affected by the agent's actions.

Trust is used daily in all aspects of our lives. While our goal is to understand trust models in computer systems, we must first consider the way we formulate trust in all areas, not only technology. We now introduce four trust models from a human experience perspective. We assert that all the systems we have discussed fit into one or more of these trust models.

5.2 Manual Verification

A person can gain trust in an agent through their own actions and observations. A person may also be able to build trust based on their previous transactions with an agent. If an agent has previously performed as expected, this could also build trust in future actions.

We have seen manual verification with SSH, IPsec and SSL with self signed certificates. Manual verification provides strong security but also require sophisticated knowledge.

These systems require a user to manually acquire, verify and install the appropriate certificates for hosts they wish to communicate with. Ideally all certificates will be installed prior to attempting to connect to the remote host. In this case verification will already be performed and, assuming verification was not faulty, clients can authenticate and communicate securely using the installed certificates. When attempting to connect to a new host, for which a certificate or key has not been established, these technologies differ. SSH and SSL with self signed certificates provide

a trust on first use (TOFU) model that allows a user to make a trust decision about the binding of a public key to a host when the first connection is made, as shown in Figure 8.

IPsec uses a number of techniques for acquiring keys. The first, especially popular in enterprise level Virtual Private Network deployments, is to run a certificate or authentication server. This is a dedicated server whose purpose is the distribution of certificates or other authentication tokens. A user would securely connect to this sever using a pre-installed certificate or username/password combination and would then be issued the appropriate certificate or other token to authenticate to their desired host. This method can at a high level be thought of as a certificate authority that is not run by a third party but by your own organization. This is a kind of manual verification technique because the verification is performed manually when the certificate is installed and not as part of a communications protocol. The second method we discuss is similar to SSH in that the certificate or public key is presented for a trust on first use verification. As with SSH, a user will be presented with a public key or certificate and must manually verify the key, a step which many users omit.

The manual verification of the key or certificate, whether in SSH, SSL or IPsec, is not a trivial task and requires an understanding of public keys and cryptography. There are three primary steps to complete verification. First, a user must ensure any metadata with the key is correct and valid. The metadata can include information about the organization such as a name, location and hostname, the key validity period, and algorithm used to generate the key. Second, the user must verify that the key has not been corrupted in transit, usually by comparing hash values. Finally, a user must verify that the key is authentic and trusted. This is usually done by obtaining the key or its hashed value through a separate, trusted path.

There is an alternative option that we feel is very common and is supported by

the unmotivated user principle. Users do not perform verification and assume the key is correct. This strategy leaves a user vulnerable to compromise in all future communications.

5.2.1 Usability

The manual verification trust model does not appear to be suitable for average users. One of the underlying themes in the usability research we have discussed, is that users do not understand the concepts of public key cryptography and certificates [30,54,57]. Some users are also not diligent in verifying the security indicators presented.

The verification of a public key for a remote host requires users to understand and potentially generate the fingerprint for a public key. They must also understand that the acceptance of a key binds the hostname and IP address to that key permanently. We feel these concepts are too complicated for an average user.

IPsec, when deployed with properly configured key management tools is a very usable solution. We have anecdotally seen it used by many users without security backgrounds in the enterprise setting while not providing usability challenges. This, however, is thanks to the team of support personnel also in an enterprise setting. They maintain a vast infrastructure and assure necessary keys are preinstalled on every machine. They also provide instructions and support should anything go wrong.

The maintenance of enterprise VPN using IPsec is certainly more than can be expected from an average user, but even simple IPsec deployments present significant challenges. Users must generate certificates in advance and distribute them to all the necessary clients, as well as installing both server and client software. Given users' difficulties understanding certificates for SSL, this also appears to be too complex for an average user.

Even the most sophisticated of users still struggle with the basic problem of

"bootstrapping" security. When establishing the first secure communication channel, regardless of the technology, the public key or certificate must be accepted as trustworthy. Best practice recommends verifying the credential via an out of band, secure mechanism. However, if this is the first secure communication channel, no such mechanism exists. At some point trust must be assumed or may be derived from a third party; this is a motivating factor for other trust models.

5.3 Authoratative Trusted Third Party

A person can formulate trust in an agent based on the endorsement of a single trusted third party. This is herin considered an authoratative trust model because trust is derived from an endorsement of a single authoratative entity. The level of trust achieved from this situation is also dependent on the trust in the third party. This model can also be extended to form trust chains. The authoratative trusted third party (or trust anchor) can delegate the trust endorsement to another party. This process can be repeated forming trust chains of arbitrary length.

This trust model, when combined with asymmetric cryptography, is the basis for Public Key Infrastructure (PKI) systems. These systems can build long, complex trust chains and include many trust anchors (or certificate authorities). These systems can be thought of as arbitrary digraphs, where each node is an entity and directed edge is a trust assertion. They have further been classified based on their structure; one common structure is hierarchical (as used by DNSSec and DNSCurve). A thorough overview of PKI trust models can be found in works by Perlman [82] and Menezes et al. [22].

The most widespread and popular trust model of the systems we have discussed is the authortative trusted third party. This is primarily due to SSL which is the de facto standard for securing HTTP traffic on the Internet. There is also a significant

movement towards deploying DNSSec; the root zone has finally been signed in July 2010.

The trusted third parties of the technologies we have discussed are certificate authorities for SSL and domain registrars for DNSSec. Certificate authorities verify the identity of an applicant and then digitally sign a certificate, binding the identity information to a public key. The signature on a certificate is verified with the public key of the certificate authority which are distributed with web browsers. The trusted third party for DNSSec and DNSCurve are, by design, the domain registrars. They provide or verify the public keys with which a domain's signatures can be verified.

ITrustPage also uses an authoratative, non-hierarchical third party, the single information source to verify web forms.

5.3.1 Identity Verification

When agreeing to sign a customer's certificate, a CA or domain registrar must verify that customer's identity. This identification process has been problematic in the past. For the identity verification process to be effective, it must consistently be performed to the same standard and achieve sufficient verification. We do not have a clear metric for sufficient verification, but we have many examples where verification is clearly insufficient. Extended Validation certificates have made progress towards a definition for sufficient verification in the context of SSL; no such standard exists for DNSSec or DNSCurve.

These authoratative trusted third party models share the same structure in which a relatively small number of trust anchors delegate trust to a large number of entities. For example, if DNSSec is fully deployed, one single TLD, `.com` will be responsible for delegating trust to over 88 million active domains (as of July 2010). We believe this presents a problem of scale: to perform consistent and sufficient identity verification on each of those entities seems a nearly impossible task. SSL shares a similar problem,

although in a smaller scale.

5.3.2 Trust in a Third Party

To determine if we trust a third party, we must first understand how that party became trusted. SSL certificate authorities have their root certificates installed in browsers by the browser manufacturers such as Microsoft, Mozilla and Apple. The initial decision to trust a CA is made by the browser manufacturers. Users are free to change what CAs are trusted, although many never do. To issue certificates for DNSSec, you must be a domain name registrar. Top level domain name registrars are authorized by ICANN [83].

When deciding to trust a third party we must also question their motivations. One clear motivation is financial; many registrars and CAs provide their revenue through the sale of these services. Note that if this was their sole motivation, a user could also purchase false certificates or DNS services; fortunately there are many other constraints under which CAs and registrars operate. There are laws in many jurisdictions which provide strong motivations or guidance to how a CA or registrar operates. Entities are also motivated by their personal set of morals and may be guided by what they feel is right or wrong. They may also be motivated by their reputation and so avoid actions which may cause it harm. A final motivation may be political: many root CA certificates belong to governments. There is also at least one root CA certificate belonging to an entity that allegedly fosters malware creation and distribution, censorship, and is under control of a government [41, 42]. Perceived motivations can bolster or undermine trust in a third party.

Organization	Product Observed	Number of Root CA	Date Observed
Mozilla	Firefox 6.0.2	208	Sept 2011
Microsoft	Windows 7	321	Sept 2011
Google	Chrome 13.0.782.220	165	Sept 2011

Table 2: The Number of Root CA Certificates

5.3.3 Multiple Roots

There are many CA root certificates installed in each browser; a table of numbers for three popular browsers is available in Table 2. As of July 2010, there were also 960 domain registrars accredited by ICANN. In the case of SSL, users must review a certificate metadata to determine which CA has signed it; with DNS technologies, the registrar may be found through the WHOIS service. Given the large number of trust anchors and the fact that many users lack the knowledge to perform these operations, a user is likely to trust all anchors without any scrutiny.

5.4 Distributed Trusted Third Party

The trusted third party model can also be distributed. A person can gather trust endorsements for a single agent from many different third parties. The person will have a trust relationship with all of the third parties; however each may have varying levels of trust. The aggregate of these endorsements is evaluated to establish trust in the agent. This provides a robust, distributed model because any combination of third parties may be able to help establish trust for an agent.

This is a natural trust model for people to use in social context: a user is able to easily make this evaluation and decision when evaluating endorsements from trustworthy and not-so-trustworthy peers. These trust models have also been adapted to computer systems. If trust can be established for an agent, that agent in turn

may provide trust recommendations for other agents thus allowing the formation of trust chains. The most widely used approach to automatically evaluating trust in a distributed trusted third party model is OpenPGP, as we discussed in Section 4.4; another proposal for a distributed trust models has also been made by Abdul-Rahman et al. [81]. Despite the ease with which this trust model is used in a social context, technological systems based on a distributed trusted third party model have had significant usability issues [54, 68]. One source of usability may be with the way trust is represented. In the OpenPGP system trust is quantized into four values of untrusted, partially trusted, fully trusted and ultimate. In the all the other systems we have seen, trust is represented in binary.

The Perspectives system provides a useful addition to the task of verifying a TOFU key or certificate. A user is given the certificate from different network locations as well as a history of how the certificate may have changed. The Perspectives system includes client software that performs the verification, no longer making it a manual process. Perspectives uses multiple notaries to make an informed trust decision; each notary is equally trusted, and the collective endorsements establish trust in an otherwise untrusted element.

5.5 Distributed Untrusted Third Party

The final source of trust that we discuss, is subtly different from the distributed trusted third party we just discussed: it is a distributed untrusted third party. This can be thought of as *trust in the masses.* Trust for an agent can be built by the endorsements of third parties, the important difference being that the user has no prior trust relationship with the third party: they are untrusted. In this situation, each endorsement is equal in weight and the total number of endorsements determines the strength of the formulated trust. An important consideration when making this

type of trust decision is also population size; for this reason percentage is a common metric. For example, 10 endorsements out of 10 users (100 percent) is much stronger than 10 endorsements out of 100 users (10 percent).

This type of trust model, also known as a crowd-sourcing system, has seen extensive use on the Internet. EBay was one of the first major sites to deploy this system to promote trust in its reputable sellers. It has been studied and shown that is a reasonable source of trust for some systems [84].

5.5.1 Hybrid Trust Models

Finally, we return to the challenge of securely identifying and connecting to remote resources. We observe that the distributed untrusted third party model is frequently used to rank and order web content to a user; this closely aligns with the first step of identifying the desired service of a remote host. Many users use search engines as the first step to discover services offered by remote hosts.

PageRank is an algorithm to rate the relative importance of an object within a set. It is used by Google as the basis of their search engine. PageRank measures the number of links pointing to a page to determine its importance. PageRank's basic notion is the same as that of the distributed untrusted third party trust model, the best or most trusted content will have the most endorsements [85].

This provides an interesting hybrid of trust models. A single trusted third party, such as a search engine, collects trust endorsements from the masses. This leverages the power of a distributed untrusted model, but is collected and aggregated by a single trusted source.

This is the approach used by ITrustPage.

Technology	Trust Architecture	Trust Mechanism	Confidentiality	Integrity	Authenticity
DNSSec	Hierarchical, Multiple Roots, Authortative Trusted Third Party	Asymmetric cryptography, digital signatures	None	Cryptographically signed	Provided by domain registrars, who have previously failed at sufficient identity verification
SSL	Hierarchical, Multiple Roots, Authortative Trusted Third Party	Public key certificates binding domain name to key, digitally signed by a CA	Strong Encryption	Message Digest	Signed Certificates, trust in signer (CA or self) varies, identity verification inconsistent, null certificate attacks can circumvent certificates, EV certificates provide stronger identity verification
SSH	Client-Server, Trust on First Use	Username-password or asymmetric key pair	Encryption	Message Digest	Manually binding public keys to hostnames
DNSCurve	Hierarchical, Multiple Roots, Authortative Trusted Third Party	Digitally signed and encrypted DNS Records, using asymmetric cryptography	Encryption, however source domains can be inferred from IP header	Encryption, no strong integrity mechanism	Provided by domain registrars, who have previously failed at sufficient identity verification
OpenPGP	Web of Trust, arbitrary digraph, Distributed Trusted Third Party	Asymmetric cryptography	Encryption	Message Digest	Build trust path through trusted entities with differing trust levels, trusted path cannot always be found
IPsec	Generally Client-Server	Various, Kerberos and IKE most popular	Encryption	Message Digest	Strong authentication once appropriate keys are distributed

Table 3: Overview of the trust models of the technology discussed

5.6 Limitations

Each of the trust models have their own weaknesses, as we have discussed. However any system that connects with a remote host shares the common problem of identifying the correct remote host.

The problem of identifying remote resources and hosts is challenging and involves several steps that most users do not regularly consider. The first step in communicating with a remote host is determining the correct service. If a user wishes to do online banking, the bank's website is the correct service. However this answer is not always so clear; if a user wishes to download a song through BitTorrent, there are many potential BitTorrent services they may wish to consider. Thanks to the Internet, there are online service providers all over the earth that are easily accessible; given this variety how does a user determine the correct one? One of the most common tools is search engines. They not only provide links to the services available based on keyword searches but also attempt to rank them by relevance.

Once a service has been chosen, how does a user identify it? For network communication, a network or IP address is necessary. However, we do not use IP addresses for identification, we use hostnames. Identifying services based on hostnames is not an easy task and one that many attackers exploit in phishing attacks. If a user wishes to connect to their bank, say Bank of America, finding the correct hostname can be difficult. The problem is further exacerbated by the domain hierarchy; potential hostnames could include `bankofamerica.com`, `secure.bankofamerica.com`, `bankofamerica.us`, `bank-of-america.com`, and many others. Identifying a previously unvisited service solely based on hostname can be very challenging, including for the most sophisticated user. Even if a service is frequently visited, a user may not notice a different, potentially malicious hostname. To aide in this, users again frequently use search engines which provide links to the service based on hostname.

When a hostname has been determined, it must then be translated to a machine routable IP address through DNS. We note that hostname to IP address mapping is not one to one; a single host can translate to many IP address and many hostnames can map to the same IP address. DNS security (DNSSec and DNSCurve) attempts to secure this step in the process. As we have discussed, this does not provide secure communication with the remote host.

After a user has the IP address, communication to that host can occur. To secure the communication we need end-to-end security which includes authentication: a verification that the host we have identified is indeed the host we are communicating with. Our idea of the remote service's identity is described by the hostname; therefore, systems bind that identity information to cryptographic information, a public key. With proper use of asymmetric cryptography it can be guaranteed that the host with which we are communicating possesses the secret key and is the desired host.

Modern systems rely on this binding to build trust; however, it is not always as strong as assumed. The first step we described of identifying the correct service is often taken for granted. Users can falsely assume that the service provided by a named host is the intended service. Further, if the intended service has been determined, the hostname used to identify it may not be correct. On the other side of the binding, it is also assumed that the public key is or can be securely acquired, which is not always the case. A user is not always performing the verification they intend.

Chapter 6

Looking Forward To A New Trust Model

In this chapter we describe, based on our trust model analysis, we suggest what a new trust model should look like. We describe an alternative authentication strategy for SSL that follows this trust model. We also describe how such a system would work against several known attack vectors.

6.1 The Characteristics Of A New Trust Model

We have analyzed security technologies, focusing on their trust models, and have identified common problems with all of them. Based on our trust model analysis, we now summarize these problems and describe the four characteristics of a new trust model.

Systems using authoratative trusted third party trust models such as SSL, DNSSec, and DNSCurve present problems with the trusted third party anchor. A user must have complete trust in each third party trust anchor, a condition that is not always true. Third party trust anchors may have a variety of motivations that may not include the security of a user's transaction. Futher, in real world deployment, the high number of trust anchors makes it very difficult for a user to make an informed trust decision for each anchor. However, these systems, in particular SSL,

also provide many positive characteristics such as strong integrity and confidentiality and a wide deployment.

1. A new trust model should have a trust anchor that has an interest in or is a part of the transaction. Additionally, a new trust model should maintain the positive characteristics of strong integrity and confidentiality but should anchor trust without relying on an authoratative trusted third party.

Another common trust model is manual verification. This is commonly employed by self signed certificates used by SSL and possibly IPsec; also SSH uses this trust model. A user is presented with a certificate, public key, or key fingerprint and the user must manually inspect it, ideally using an out of band mechanism, and make a trust assessment. The weakness with this model is that most users do not have the knowledge to make such an assessment and many users are also unmotivated to do so.

2. A new trust model should not rely on users to manually verify credentials.

An alternative trust model is the distributed trusted third party or web of trust approach. This model has benefits in that the degree of trust is highly customizable and each user decides on their own trust anchors. Current OpenPGP applications, though, have shown that average users can have difficulties managing keys and performing cryptographic operations [54].

3. A new trust model should not rely on users to manage keys.

Finally, we also observe that most of the trust models we have seen are based on hostnames used to identify a remote resource. Users do not always understand or pay

attention to hostnames; this provides a vector which can be exploited such as with phishing attacks.

4. A new trust model should use something other than a hostname to identify a remote resource.

To find a trust model that meets the characteristics described above, we first consider the characteristic of using a trust anchor that is involved in the transaction. We believe this points towards a **trusted referrer** trust model.

6.2 Trusted Referrer

Fundamentally, a trusted referrer is still a trusted party; however the party is directly involved in the transaction. A referral is a recommendation towards another entity; the referrer wants the user to arrive at the destination. This provides a potentially strong source of trust that can form the basis of trust relationships.

If we consider this trusted referrer in the case of the Internet, we see that links on the Internet are trust endorsements, and by following a link a user is implicitly declaring their trust for the link's poster. This is a trusted referral.

Trusted referrer is a natural concept that people understand; it is used in every day life. It also follows closely with the navigation model of the Internet. The Internet is full of web links, and each link is a referral.

The trust in a referrer, as any trusted party, will depend on the particular party; a decision that is is implicitly made by a user's choices of what referrals (links) they follow. There are two user studies which provide strong support for this assumption. Hargittai et al. [86] conducted a user study of 102 young adults and found that users rely heavily on search engines to guide them to credible material and the trust they have in that material is derived directly from the path taken to get to it. The

study does not examine if this trust is well placed (i.e. if the link is trustworthy), although it does note that many participants chose this method based on previous success. Also, Pan et al. [87] conducted a user study that revealed that college student users have substantial trust in Google's ability to rank results relevant to their query. Specifically, users would click on higher ranked Google results instead of lower ranked results more relevant to their query. This supports two of the points of such a system, first that users start at familiar sites such as search engines while browsing the Internet. Secondly, that users have significant trust in these familiar sites, search engines in this case, so they would be strong trust anchors.

A user's trust in a trusted referral is only in the context of the link. For instance, if a user trusts Google to provide them with the most relevant search result, the trusted referral will ensure that the link a user follows from Google in the destination Google intended.

The second criteria we have used when looking at the confidence in authentication is the identity verification process. Our proposed system design does not provide any identity guarantees, it only assures you are going to an intended site. When considered in the case of a search engine, the verification process is akin to returning the best search result. Returning the best search result is what all search engines strive for; if they cannot do so, they will undoubtedly lose customers. There is a significant industry in search engine optimization, the optimization of web pages to achieve higher ranked search results. This is an industry that search engines fight against to avoid manipulation of their results. It is clear that major search engines invest enormous resources into returning the result that best satisfies a user's query, far more than domain registrars or CAs invest in identity verification.

While some research clearly points towards familiar sites such as search engines being strong trust anchors, we believe this will also extend to other familiar sites such as news aggregators, blogs and social networking sites also being strong trust

anchors. The basic principle is the same, these sites rely on producing the best content or results to attract users. However, further research is needed to verify that this holds.

Users utilize and in fact rely heavily on search engines to lead them to the best web content; in many cases they trust a search engine's ranking over their own assessment. Search engines in turn invest heavily and strive to return the best results; a failure to do so results in loss of users' trust and browsing. Users' strong trust in them makes them excellent trust anchors.

6.3 SSL Based Trusted Referrals

We describe the design of a system that conforms our new trust model. We design such a system with the goal of preventing network based Man in the Middle attacks such as DNS cache poisoning and compromising SSL connections with fraudulent certificates.

First we consider the strengths of SSL, it is widely deployed and thoroughly tested system. The modern versions of the SSL protocol and cryptography have shown few weaknesses. These characteristics are desirable and make SSL a good starting point for our system design.

SSL has shown weaknesses in two primary areas, the authentication provided by certificate authorities and usability. As previously discussed, the usability of SSL particularly within a web browser has been problematic. Since usability is out of the scope of this thesis, we accept these weakness and acknowledge they will continue to be a part of our SSL based system design. We do note that user studies have shown that careful design and testing of the user interface can provide some usability improvements [58, 60].

The authentication of SSL can fail when an attacker is able to obtain a fraudulent

certificate or a certificate authority uses their authority in a way which the end user may not be aware of or agree with. The authortative trust model of SSL increases the impact of such compromise because a certificate authority can issue a certificate for any site on the internet; a single CA failure could compromise any SSL secured communication.

HTTPSY has provides a valuable mechanism for validating the server of SSL connections by including the hash of the public key with the URL. HTTPSY has been used in the Trust Management for Humans system to address phishing. We can leverage the idea to help our goal of preventing network base MITM attacks. HTTPSY is not completely backwards with the existing HTTP and HTML Internet infrastructure. Any machine that does not understand HTTPSY will not be able to handle the URL as standard HTTP. It also provides human compatibiliy issues, further complicating a URL making it more difficult for human manipulation.

We use the idea to include public key certificates hashes with web links. The public keys can then be used to create secure SSL connections to the link's target. The verification of the certificate is no longer provided by a third party certificate authority. The certificate presented with the link is used as the identity of the site, instead of a hostname, and can be compared to the destination certificate of the destination site to verify it's identity. The identifier of the site, the certificate, is verified by the link's poster to match its intended destination. We propose a mechanism other than HTTPSY that provides compatibility with existing Internet infrastructure.

Such a system meets the criteria we describe for a new trust model. This system uses SSL to provide strong confidentiality and integrity, as well as leveraging the widespread and established SSL infrastructure. The identity verification is provided by the public key certificates included with web links, the certificate serves as the identity of the remote resource, not a hostname. Since a web link is processed and the link destination is loaded by a web browser, the verification of a certificate can

be performed by the web browser. The web browser may automatically maintain a cache of public keys for a user's preferred sites; this would not require a user to manage public key certificates. Finally, the source of the link who includes a public key certificate, will verify that the certificate and link point the the intended resource, not an arbitrary third party.

If a trusted referral is compromised, the scope of the compromise is limited. Only the single referral will be compromised, a user man find a referral from another trusted source or continue to communicate with any other host who's referral is not compromised.

6.3.1 HTML Tag Attributes

Our system design proposes including public key certificates with web links. To accomplish this, we must modify the attributes of the HTML anchor or `<a>` element to include the appropriate certificate information. We will include a new attribute with links called an `auth` attribute, short for authentication. The `auth` attribute will contain the fingerprint of the link target's certificate.

We have chosen to include the fingerprint, a secure hash, of a target's certificate in the anchor elements. The primary advantage is the size: a fingerprint is a small number of bytes relative to the size of an entire certificate. A full certificate contains a key that is likely between 1024-4096 bits, as well as many bytes of identifying information. A fingerprint varies in size depending on the hashing algorithm used, but is likely 128-256 bits, or 16-32 bytes in length.

We recognize that multiple secure hash functions are commonly used to fingerprint certificates, and we wish to support this diversity as well as future secure hash functions. To accommodate this, we also include an `authtype` attribute in anchor elements. This attribute specifies the algorithm with which the fingerprint was computed.

Without auth and authtype	With auth and authtype
 Carleton University 	 Carleton University

Figure 9: Example of HTML anchor elements with and without authentication attributes

The hash function provides a one way function that can be used to securely verify a given certificate matches the intended certificate, an important part of authentication and the security of the system as a whole. As such, the use of a secure hash function is critical. The choice of hashing algorithm varies and MD5 and SHA1 are the two most common. MD5 has been shown to not be collision resistant [88] and progress has also been made towards achieving collision attacks against SHA1 [89]. As such we would recommend the use of SHA256. We leave the full analysis and final choice of hash function as a choice to the implementor.

Website developers, as they create content, must include the `auth` and `authtype` elements for each link to a different domain. In creating a link, they must acquire the certificate fingerprint securely from the target site and include it in the `auth` attribute. This should require minimal extra work; as developers test any cross domain links they wish to include, the certificate of that site will be acquired and simple script could calculate the fingerprint. Such a system does not verify the hostname of a site to which a link points; the developer only needs to confirm this is the destination they intended. Developers should also record of the validity period of the certificate for a target site. The `auth` and `authtype` attributes of the link will have to be updated when the validity period of the certificate expires. This process could be automated to allow for simple posting of secure links.

The certificate of an intended target can be acquired by a basic web request. The developer can verify that the certificate belongs to the intended target though various potential checks including: a visual inspection of the content, knowledge of unique private data on the site, verifying the certificate if signed by a certificate authority or public domain registration information from the WHOIS utility. This step may be vulnerable to phishing and other spoofing attacks.

One other minor change to HTML content that should be noted is that links with the `auth` attribute should point to `https` sites, not `http`. This change is compatible with both current versions of HTML and the future HTML 5.

6.3.2 Server Side

The server side implementation of such a system, apart from the HTML content modification we have discussed, would be very simple. An HTTP web server needs only to support SSL connections.

A server administrator must enable SSL connections and create an SSL certificate. Any client connecting to the server would follow the usual handshake and connection process for SSL.

One very important characteristic of such a system is that it is independent of the type of SSL certificate installed on a server; it provides certificate verification based on the link to a site. A site may use any of the three levels of certificates we have discussed, self-signed, standard CA signed or extended validation. Regardless of the type of certificate used, such a system would provide verification that the certificate at a destination matches the `auth` attribute of a link.

6.3.3 Client Side

Such a system would also require modification to a web browser's handling of links and certificates. There are two primary modifications: the handling of extra attributes in anchor objects, as well as the certificate fingerprint verification.

The first step occurs when the HTML source is being parsed. The browser must interpret the `auth` and `authtype` attributes for links. These will be stored in the Document Object Model (DOM) with other attributes of the anchor element such as name, ID, href (link target). This fingerprint should also be cached for future use, with cached fingerprints being stored until newer fingerprint is available. If a browser does not understand the `auth` and `authtype` attribute, they will simply be ignored. This provides backward compatibility.

The second step is performed once a user has clicked a link with `auth` attribute. The link will be followed normally, a connection will be made to the remote host and the SSL handshake will begin. While the client is performing the certificate verification (i.e. validating the signature of the CA on the certificate), it must also perform the extra verification for our system. This verification requires the browser to hash the certificate presented, with the algorithm specified by the `authtype` field, and compare the hash value with the fingerprint stored in the `auth` attribute of the anchor element of the DOM. When a user navigates to a site without following a link (e.g. by typing in the address bar), the browser will also perform the same verification based on any cached copy of a fingerprint associated with the domain. A previously unvisited site uses the TOFU model and no verification will be made.

Users are mobile and may uses a variety of different machines, some of which may be untrusted. While we envision that initial implementations would maintain a single cache of fingerprints within a browser, it may also be possible to synchronize fingerprint caches across machines and potentially even between users. This would

be useful for the situation of occasionally use of a temporary workstation or users migrating to a new machine.

An implementation must also consider revocation of trust. If a user accidentally reaches a site that they do not trust, there must be some simple facility for the user to indicate so. Specifically, a user interface element to remove that certificate from the cache so trust chains are not built upon it.

6.3.4 Trust Model

We have described a system design that uses a new trust model, trusted referrals, that leverages a user's browsing habits to build trust chains anchored in familiar sites. This is similar to an untrusted third party trust model, with the third parties being any website that posts a link.

The trust anchors or completely trusted entities will be the familiar sites a user visits, such as their preferred search engine, news sites, article aggregators and blogs. There is not a strict definition of an anchor; they are simply sites that a user visits and trusts the content that they provide. From each of those familiar trusted sites users will follow links to new sites. These new sites are partially trusted entities and users may choose to trust them and follow links. If a user does not trust the partially trusted entity, they may return to a trusted search engine and search for the desired material.

Such a system would provide confidentiality, integrity and authenticity. Confidentiality and integrity are provided through the SSL communications channels, and as with SSL the strength of these characteristics depends on the choice of ciphers used.

In terms of authentication, this system provides a subtle variation on the type of authentication it provides. The other systems we have discussed provide a binding public key to identity information, especially hostname. Our system design does not provide that; instead, it provides a verification that the link you are following connects

you to the host the link creator *intended* you to arrive at. It does not provide any guarantees about who the host is. Such a system uses the certificate of the destination host as an identity, not the hostname.

6.4 Security Analysis

6.4.1 Threat Model

The primary threat model our system design defends against is an intruder in the middle. Intruder in the middle attacks are attacks where an attacker can intercept, modify, read or inject traffic into an IP stream for which it was not the intended recipient. Our threat model includes DNS cache poisoning as intruder in the middle because through cache poisoning an attacker can intercept, through redirection, IP traffic.

Under this threat model an attacker would have three objectives.

Redirect Traffic An attacker may wish to redirect traffic to malicious site for the purposes of phishing, malware distribution or to artificially receive more traffic.

Read Contents An attacker may wish to read the contents of the IP traffic to obtain sensitive or private information. Examples may include passwords, credit card numbers, authenticated session cookies or private personal information.

Modify Contents An attacker may wish to modify the contents of IP traffic for two possible objectives

- Alter the information received by the client
- Alter the information submitted by the user to the server.

To perform an intruder in the middle attack, an attacker must gain access to IP traffic. This is typically done through controlling a machine on the physical route to a destination or through exploiting weaknesses in the network protocols.

Intercepting traffic on the physical route may be done by legitimately controlling a machine through which the traffic passes. This could be a typical server for instance, acting as a firewall, or it could be a switch or router. An attacker may also illegitimately gain access to such a machine through a variety of attacks such as password bruteforce or exploiting software vulnerabilities. An opportunistic attacker may be able to receive traffic through a misconfigured switch or router. The physical route may not only be a wired connection, an attacker may also gain access through unsecured wireless transmission.

DNS cache poisoning has already provided one example of how network protocols may allow intruder in the middle attacks, there are also two other well known examples that directly attack IP routing. The first is Address Resolution Protocol (ARP) poisoning. ARP poisoning allows an attacker on the same switch to declare their physical (Ethernet) address as the destination for targeted IP addresses, frequently the gateway (to the Internet) address [27]. Border Gateway Protocol is one of the core routing protocols on the Internet. BGP announcements, or the declaration of routes to specific IP ranges, are not authenticated. If an attacker can get a BGP announcement to a core router, they can potentially hijack the route to specific IP addresses [28, 29].

6.4.2 Defending

We look at how SSL-based Trusted Referrals will defend against several specific intruder in the middle attacks identified in our threat model. When this system is fully deployed and a user browses in the assumed model, it will prevent all known intruder in the middle attacks. Most intruder in the middle attacks, though, would still result

in a denial of service for the client.

Such a system would also substantially improves a user's security against many other attacks including phishing and malware distribution. Many other forms of attacks rely on intruder in the middle to redirect users to the attack site. An attacker may use intruder in the middle attacks to redirect users to a malicious site that performs phishing attacks, drive-by-downloads or other malware distribution.

DNS Cache Poisoning

The first scenario we consider is a DNS cache poisoning attack. An attacker has poisoned a DNS cache for `www.mybank.com` to point to their replica of the `mybank` website. The real bank website uses SSL signed by a certificate authority, and the attacker's replica uses SSL with a self-signed certificate. The user accesses the bank website by querying for "mybank" in their favorite search engine, and clicking on the top result.

When this attack occurs without our potential system in place, the user will receive the poisoned DNS entry while following the link from the search engine. The user will arrive at the attacker's site replicating "mybank" and will receive an SSL self signed certificate warning (see Figure 7). Some users will heed the SSL warning and terminate the connection; however, many users will also disregard the warning and continue to the attacker's site and divulge their banking credentials.

With our system design in place, the user will click on the trusted referral link from the search engine and their browser will recognize the additional `auth` and `auth-type` properties of the HTML. The browser will update the cached record for `www.mybank.com`, if present, with the fingerprint from the link. The bank continues to use a certificated signed by a traditional CA, so the fingerprint will be a hash of the CA signed certificate. While following the link the user will receive the poisoned DNS entry and attempt to connect to the attacker's site. When presented with the

attacker's self signed certificate, the browser's SSL client will observe that the fingerprint does not match the search engine referral's fingerprint and react appropriately, ideally terminating the connection.

Compromised Router and Untrusted Certificate Authority

The second scenario we look at will be a sophisticated attack with two points of deception. An attacker has compromised a user's home router by logging in with the default username and password, which the user neglected to change. The attacker scripted a simple rule to intercept traffic for `99.99.99.99`, the user's frequently visited online store, and change the IP to `66.66.66.66`. The legitimate online store site uses SSL signed by a CA. The attacker owns the IP `66.66.66.66` and has created a replica of the online store normally found at `99.99.99.99` and also created scripts to install malware on the user's system when clicking a link for an item on sale. The attacker has also, through a social engineering, acquired a certificate signed by a CA recognized by all browsers.

Without a system such as ours in place, the user would navigate to the store's site and be redirected to the attacker's site. The attacker's site appears visually identical. The user also verifies the `https` in the url bar and lock icon in the chrome because they are concerned about their account information including address and credit card numbers; both indicate a secure SSL connection. The user follows the link for the item on sale and unknowingly has malware infect their machine.

With our system design in place, when a user enters the URL for the site, the browser would also notice that this is a frequently visited site for which the certificate fingerprint was previously cached. The user's traffic would be forwarded to the attacker's site. However when verifying the CA signed certificate, the browser would notice that the certificate fingerprint does not match the cached copy, and would respond appropriately, ideally terminating the connection.

ARP Poisoning Little Brother

The final scenario involves a technologically savvy little brother. A teenage girl is updating her personal profile on her favorite social networking site. This site uses unsecured HTTP for all traffic. The teenage girl's little brother has discovered the tool ettercap [90] used to poison ARP tables and hijack TCP sessions. He is on the same local network and has propensity for practical jokes.

Without a system such as ours in place the brother could use his tool to hijack his sister's TCP session and modify her HTTP requests to change her public status and profile.

With our proposed system design, an attacker could still hijack a TCP session; however, the HTTP requests would be encrypted in an SSL session so they would not be able to modify the request.

6.4.3 Comparison to Related Systems

We have discussed three system that use similar technologies to address the overall issue of identifying remote resources, ITrustPage in section 4.7, Trust Management For Humans in section 4.6 and in section 4.2, Perspectives. In Table 4 we present a comparison of theses technologies and SSL-based Trusted Referrals versus the attacks scenarios listed in our Threat Model (section 6.4.1). We look at the protection the our proposed system design provides given normal circumstances to show that a system using our new trust model provides additional protections

We do not include Web-Keys in our comparison because it solves an entirely different problem, that of client authentication to the server. As such it would not provide defense against any of the attacks we describe.

	Cache Poisoning	Compromised Router and Untrusted CA	ARP Poisoning
Perspectives	**Safe** When the user gets redirected to the fake bank site with a self signed SSL certificate, Perspectives would query multiple notaries to verify the certificate. Due to the short life of cache poisoning attacks, and multiple network vantage points provided by Perspectives, the discrepancy in self signed certificates would be noticed and the user alerted	**No Protection** Perspectives only validates self signed or Trust on First Use (TOFU) credentials. The certificate presented by the site has been signed by a Certificate Authority Trusted by the user's browser.	**Not Applicable**
ITrustPage	**Little Protection** When the user enters their bank credential in the login form, ITrustPage would attempt to verify the from belongs to `www.mybank.com`. Since the DNS cache was poisoned, the browser and ITrustPage plugin believe the user is at `www.mybank.com`. If the attacker's site replicated the HTML form from the legitimate bank site, ITrustPage would validate the phishing site as legitimate.	**Little Protection** The user has not been presented with any forms so ITrustPage will not be used. ITrustPage may provide some security if the user is required to log in before shopping.	**No Protection** This is not the situation ITrustPage attempts to solve, hence it does not provide any protection
Trust Management for Humans	**Possible Protection** Trust Management for Humans requires the user to perform a verification of browser cues. In this scenario the user is already presented with at lease one browser cues, for the self-signed certificate, which was not noticed. If that user does not heed the browser cues, then Trust Management for Humans will not provide any additional protection. However the user chosen petname intuitively seems to be a simpler browser cue, so it may be noticed.	**Likely Protection** Trust Management for Humans requires the user to perform a verification of browser cues. In this scenario the user has already verified the SSL browser cues so Trust Management for Humans will likely also be noticed. If that user does not heed the browser cues, then it will not provide any additional protection. Even thought the attacker has a CA signed certificate, they will not be able to generate the same key pair, so the certificate fingerprint will not match.	**Safe** Trust Management for Humans requires SSL which provides confidentiality to HTTP sessions
SSL-based Trusted Referrals	**Safe** The attacker's site will not have a fingerprint matching the that in the link.	**Safe** The cached fingerprint of the certificate will not match the attacker's CA signed certificate.	**Safe** Such a system requires SSL which provides confidentiality to HTTP sessions

Table 4: Comparison between the protections offered by related technologies

6.4.4 Weaknesses Of SSL-based Trusted Referrals

While we have explained scenarios in which our system design provides strong defense against intruder in the middle attacks, there are also attacks against such a system. Our system design, like many others relies on trust chains and each link a user follows is a trusted referral. To compromise such a system an attacker must break the trust chain.

If an user reaches an untrusted site, links are no longer trusted referrals and the trust chain will be terminated. They can return to a trusted site and start a new path to their destination. They must also indicate that the site is untrusted, likely though a button in the browser chrome. By doing so they will remove the untrusted site's certificate fingerprint from the cache so referrals will not be trusted.

SSL-based Trusted Referrals uses web links as trusted referrals; if an attacker can modify the content of a web page they can change the target of the links or `auth` attributes and break the trust.

There are many ways an attacker can modify the content of a page to break the trusted referrals. We assume the system as been implemented and deployed as specified, therefore an attacker cannot modify the content in transit, unless they can compromise SSL. There are many situations where users are allowed to post arbitrary content, including links, on web pages such as discussion boards, comments sections or social networks. A user must be careful to note the source of user posted content. However, if a user is aware of the author of user submitted content, our trust model holds; they can choose to follow a link based on the the reputation of the author. There will still undoubtedly be cases where the author misrepresents themselves, or is not available. In theses circumstances the trust chain will be broken.

One of the most common attacks to illegitimately modify web content is Cross Site Scripting (XSS). This occurs when an attacker can inject scripting code into a

web page. This script code can execute any number of actions, including modifying the content of a web page, including links or potentially link attributes such as the `auth` attribute.

Any web server that does not use SSL and does not have a certificate that can be fingerprinted could not be referred to. A site's administrator may elect not to use SSL to reduce server load, because they are ignorant to many threats or because they simply don't want to. If they do not, other sites will not be able to provide trusted referrals to them.

The system design we have proposed uses TOFU model when browsing to a site via the url bar. When a user enters a URL, if the site does not have a cached fingerprint, they will be vulnerable to all the intruder in the middle attacks our system normally defeats. It is also worth noting that this will be the case the very first time a user goes to a site after enabling this system. Once they have visited a site for the first time the fingerprint will be cached and trust chains can be built securing the future communication. If an attacker did compromise the initial connection, all future communications could also not be trusted.

A similar TOFU situation can also occur when a user visits a site directly (i.e. not through a trusted referral), after a certificate has been updated. This would create a false positive for an attack situation. This situation can be resolved by navigating to the site though a search engine or other familiar trusted site.

Of course the security of a system is only as strong as the tools used to build it, in our case this is secure hash functions and SSL. Our system design allows the link poster the choice of secure hash function. As we have discussed several common secure hash functions are already "broken"; collisions for MD5 can be achieved in minutes on modern hardware [88]. An attacker that can find collisions within a secure hash function could use those to compromise the entire system. This system uses SSL to communicate with remote hosts so any SSL vulnerabilies due to protocol, software,

cryptography or user interface will also be present in an implementation of SSL-based Trusted Referrals.

SSL-based Trusted Referrals also relies upon the browser to store and verify fingerprints. If a browser, or entire operating system, is compromised such a system could be defeated.

Chapter 7

Discussion

In this chapter we provide a discussion of the trusted referral model and its relation to the established technologies. We discuss limitations and challenges that will be faced by this trust model any system implementing it, which leads into future work that could address some of these limitations. We discuss the implications of our analysis of trust models and trusted referrals. Finally, we conclude.

7.1 Limitations of the Proposed Trust Model

We have proposed a new trust model which provides security based on trusted referrals. Some users have become accustomed to the security provided by current systems especially SSL, the identification of a website based on hostname. The trust model we propose provides a different kind of security, trusted referrals, and users will take time to understand and adjust.

7.1.1 User Interface

We feel the single greatest challenge in implementing this system will be providing a clear user interface that average users can understand. Every one of the security technologies we have discussed have difficulties with user interface; this is not uncommon,

it is an incredibly difficult problem.

An unclear or ambiguous user interface can ruin much of the security provided by any system. If users cannot understand the security status of their connection, how can we expect them to make an intelligent decision. Further, many users simply do not care about security or assume it; even with the proper information they may not bother to act securely.

We feel trusted referrals may provide a potential improvement in usability. It builds trust on a user's normal actions and they do not need to perform additional steps to ensure security.

This trust model may also provide improvements in this area as it does not rely on users to understand and verify a hostname, a task that has proven to be difficult for average users. User interface has been a significant issue with SSL and an implementation of this trust model on the Internet would face the same challenges.

We have also been encouraged by recent work which has shown that careful design of SSL user interfaces can provide better security [58]. However, the usability and user interface remains a very difficult problem for all secure systems. Although we believe our new trust model would provide improvements in this area, it has not been studied and it is clear that much future work is still needed in this area.

7.1.2 Deployment Of Our Proposed System Design

Moving forward with our proposed system design, one of the clear challenges would be wide scale deployment. The ideal initial source of server deployment would be very common sites, especially search engines. If a few sites began supplying the `auth` and `auth-type` properties in their links, we feel this could immediately provide significant improvement as users could start to use the trusted referrals. From there, we hope it would spread to all sites across the Internet and users would be able to trusted referrals for most links. Without wide deployment the trust chains a user can build

are short, increasing the difficulty to find a trusted path to a destination.

There are three primary components which would need to be created. First, new properties would need to be added to the anchor types of the HTML Document Object Model (DOM). This change would not be difficult as the DOM is an extensible structure and already sees different tags and properties used for proprietary content. The second component would be a cache for fingerprints. Most browsers already have caching facilities that could be leveraged to include a separate cache for fingerprints. The third component would be an additional linkage to allow the cache of fingerprints to serve as verifiers for SSL.

We believe the best route for an implementation would initially be as browser plugin that a user could choose to deploy when desired. As the model gains adoption, we hope it would become core functionality of all web browsers.

A limitation to this system and its deployment will be the lack of support for non-secured pages. A majority of the Internet does not use SSL and our system design does not currently have facilities to provide trusted referrals to these unsecured pages. There may also be some resistance to deploying SSL with self-signed certificates because browsers provide large warnings against visiting self-signed SSL sites. These warnings may cause a loss in traffic to some sites.

7.2 Future Work

We feel there are several areas for future work primarily in implementing and testing a system to use the trusted referral model

The biggest challenge we identified, a meaningful and comprehendible user interface, is a clear area for future work. This will include both the design of the interface, but then at least one user study to verify it has met its goals. Once the user interface has been designed, a user study should be completed to evaluate the usability of the

full system.

We would also like to see a method to integrate non-SSL-secured sites with our proposed system design.

As previously mentioned, a future area of work would be portability of a users' personal trusted referrals between browsers or other clients. A mechanism in which a user can migrate their trust referrals to a new machine or synchronize them between multiple machines would be useful features. There is also potential for a user to store their personal cache of trusted referrals on a service provider on the Internet. In such a scenario, a user could access their own sources of trust from any Internet connected machine.

The final clear area for future work in our proposed system design, is defining the appropriate mechanisms to handle key rollover and revocation. These are very challenging problems that apply to all PKI systems. Certificates have expiry dates, which would would implicitly add expiry dates to links, they must be appropriately updated. Also, if a key pair is compromised an administrator will be forced to discontinue using that key pair and generate a new one. This would invalidate links that had fingerprints to the former public key. The appropriate mechanism to deal with these situations securely and ideally transparently to the end user must be developed.

7.3 Implications

Trusted referrals provides security based on a different trust model and we feel this security model is more in line with the current state of the Internet and we believe also with its future.

All the previous systems we have discussed rely on hostnames as the basis for identity, and all of these systems have vulnerabilities because of that. The new trust model tries to secure communications by verifying a user reaches the referred

destination. This is done through trusted referrals, sites we visit provide links or referrals to the resources we wish to see. We, as users, trust these sources; if we believe the link is indeed to a resource we wish to see, follow the link.

The widespread growth of social networks across all types of web content coalesces with this trust model. Social networks allow users to share or endorse web content they find interesting, useful or amusing and also potentially provide comments on it; they can also provide comments rejecting the content. As social networks' popularity grows and all web content begins to contain endorsements from a user's social network, users will be given even more information to make the trust decision.

Social networks' integration with trusted referrals leads to a more natural web of trust model. A user will evaluate the endorsements from any number of social peers, for each peer they may have varying degrees of confidence. With the assessment of a user's collective social group they are empowered to make more informed trust decisions.

We feel this direction provides a potentially bright future in which users can make informed trust decisions not based on complex cryptography and hostnames, but based on what humans evaluate naturally: referrals from peers. This trust decision is applied implicitly by following a link and all the while strong cryptography is working invisibly to provide secure communications to remote hosts.

7.4 Conclusion

In this thesis we have provided a thorough overview of existing technologies used to secure communications to remote hosts. Through careful analysis of these technology's trust models we have summarized deficiencies in all of them. Trusted third parties used by DNSSec, SSL, and DNSCurve may not be trusted at all. We also found that all systems have difficulty in terms of usability, especially those of manual

verification trust models, SSH and IPsec.

To address these some of these issues we suggest a new trust model based on trusted referrals. This trust model leverages users' natural behaviour of following referrals (including web links) for which they trust. We also describe SSL-based Trusted Referrals to show how this trust model could be implemented. This system design includes the fingerprint of public key certificates with all links. Sites are then connected to securely through SSL, with the certificates being verified by the link's fingerprint. The security analysis of our potential system design shows when fully deployed it could protect against many known intruder in the middle attacks, which may be used to instrument phishing attacks (we do not direcly deal with phishing).

We believe that this trust model is more aligned with the current state of the Internet including user's navigation, how they seek content and a increase in social content. It provides trust based on simple user actions, and does not require a user to understand the workings of the secure system. It also disassociates from the hostname based identity which has been problematic to other systems. Finally with the future integration of social networks into all web content, we feel this system will provide a natural way for users to easily communicate securely with remote hosts.

List of References

[1] J. Abbate. "From ARPANET to INTERNET: A history of ARPA-sponsored computer networks, 1966-1988." *Disseration, University of Pennsylvania* http://repository.upenn.edu/dissertations/AAI9503730/ (1994).

[2] P. Mockapetris. "Domain names: Concepts and facilities." RFC 882. http://www.ietf.org/rfc/rfc882.txt (1983).

[3] P. Mockapetris. "Domain names: Implementation specification." RFC 883. http://www.ietf.org/rfc/rfc883.txt (1983).

[4] P. Mockapetris. "Domain names - concepts and facilities." RFC 1034 (Standard). http://www.ietf.org/rfc/rfc1034.txt (1987).

[5] P. Mockapetris. "Domain names - implementation and specification." RFC 1035 (Standard). http://www.ietf.org/rfc/rfc1035.txt (1987).

[6] Internet Systems Consortium. "ISC BIND." https://www.isc.org/software/bind.

[7] D. J. Bernstein. "dnjdns." http://cr.yp.to/djbdns.html.

[8] S. Bellovin. "Using the domain name system for system break-ins." In "The 5th Usenix UNIX Security Symposium," Usenix (1995).

[9] S. Friedl. "An Illustrated Guide to the Kaminsky DNS Vulnerability." http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html (2008).

[10] IBM Internet Security Systems. "BIND 8.2 and 8.2.1 remote buffer overflow in the processing of NXT records." http://www.iss.net/security$_c$*enter*/*reference*/*vuln*/*bind* − *nxt* − *bo.htm*.

[11] CERT. "CERT Advisory CA-2002-19 Buffer Overflows in Multiple DNS Resolver Libraries." http://www.cert.org/advisories/CA-2002-19.html.

[12] D. Moore, C. Shannon, D. Brown, G. Voelker, and S. Savage. "Inferring Internet denial-of-service activity." *ACM Transactions on Computer Systems (TOCS)* **24**(2), 139 (2006).

[13] N. Brownlee, K. Claffy, and E. Nemeth. "DNS measurements at a root server." In "Global Telecommunications Conference, 2001. GLOBECOM'01," volume 3, pages 1672–1676. IEEE. ISBN 0780372069 (2002).

[14] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh. "Protecting browsers from DNS rebinding attacks." *ACM Transactions on the Web (TWEB)* **3**(1), 2 (2009).

[15] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee. "Increased DNS forgery resistance through 0x20-bit encoding: security via leet queries." In "Proceedings of the 15th ACM conference on Computer and Communications Security," pages 211–222. ACM (2008).

[16] D. Eastlake 3rd and C. Kaufman. "Domain Name System Security Extensions." RFC 2065 (Proposed Standard). http://www.ietf.org/rfc/rfc2065.txt (1997).

[17] D. Eastlake 3rd. "Domain Name System Security Extensions." RFC 2535 (Proposed Standard). http://www.ietf.org/rfc/rfc2535.txt (1999).

[18] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. "DNS Security Introduction and Requirements." RFC 4033 (Proposed Standard). http://www.ietf.org/rfc/rfc4033.txt (2005).

[19] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. "Resource Records for the DNS Security Extensions." RFC 4034 (Proposed Standard). http://www.ietf.org/rfc/rfc4034.txt (2005).

[20] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. "Protocol Modifications for the DNS Security Extensions." RFC 4035 (Proposed Standard). http://www.ietf.org/rfc/rfc4035.txt (2005).

[21] B. Laurie, G. Sisson, R. Arends, and D. Blacka. "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence." RFC 5155 (Proposed Standard). http://www.ietf.org/rfc/rfc5155.txt (2008).

[22] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC (1997).

[23] D. Piscitello. "Domain Hijacking: Incidents, Threats, Risks and Remedial Actions." http://www.icann.org/en/announcements/hijacking-report-12jul05.pdf (2005).

[24] D. Danchev. "ICANN and IANA's domains hijacked by Turkish hacking group." http://www.zdnet.com/blog/security/icann-and-ianas-domains-hijacked-by-turkish-hacking-group/1356 (2008).

[25] L. Constantin. "Gross Negligence Surfaces in Baidu Domain Hijacking Incident." http://news.softpedia.com/news/Gross-Negligence-Surfaces-in-Baidu-Domain-Hijacking-Incident-136099.shtml (2010).

[26] J. Leyden. "DNS attack hijacks Twitter." http://www.theregister.co.uk/2009/12/18/dns$_t$*witter*$_h$*ijack*/(2009).

[27] R. Wagner. "Address resolution protocol spoofing and man-in-the-middle attacks." *The SANS Institute* (2001).

[28] A. Kapela and A. Pilosov. "Stealing the internet - a routed, wide-area, man-in-the-middle attack." In "Defcon 16, Las Vegas," (2008).

[29] R. Kuhn, K. Sriram, and D. Montgomery. "Border gateway protocol security." *Recommendations of the National Institute of Standards and Technology, Special Publication 800-54* (2007).

[30] R. Dhamija, J. Tygar, and M. Hearst. "Why phishing works." In "Proceedings of the SIGCHI conference on Human Factors in computing systems," page 590. ACM (2006).

[31] K. Hickman. "SSL 0.2 Protocol Specification." http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt (1994).

[32] D. Wagner and B. Schneier. "Analysis of the SSL 3.0 protocol." In "Proceedings of the 2nd USENIX Workshop on Electronic Commerce," pages 29–40 (1996).

[33] P. K. Alan Freier, Philip Karlton. "The SSL Protocol Version 3.0." http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt (1996).

[34] T. Dierks and C. Allen. "The TLS Protocol Version 1.0." RFC 2246 (Proposed Standard). http://www.ietf.org/rfc/rfc2246.txt (1999).

[35] T. Dierks and E. Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.1." RFC 4346 (Proposed Standard). http://www.ietf.org/rfc/rfc4346.txt (2006).

[36] T. Dierks and E. Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.2." RFC 5246 (Proposed Standard). http://www.ietf.org/rfc/rfc5246.txt (2008).

[37] W. Stallings. *Handbook of computer-communications standards; Vol. 1: the open systems interconnection (OSI) model and OSI-related standards* (1987).

[38] R. Housley, W. Polk, W. Ford, and D. Solo. "Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile." RFC 3280 (Proposed Standard). http://www.ietf.org/rfc/rfc3280.txt (2002).

[39] "CA/Browser Forum." http://www.cabforum.org.

[40] I. Ristic. "Internet SSL Survey 2010." In "Black Hat USA 2010," (2010).

[41] E. Felten. "Mozilla debates whether to trust chinese CA." http://www.freedom-to-tinker.com/blog/felten/mozilla-debates-whether-trust-chinese-ca (2010).

[42] J. Corbet. "China internet network information center accepted as a mozilla root CA." *Linux Weekly News* http://lwn.net/Articles/372264/ (2010).

[43] M. Marlinspike. "SSL And The Future Of Authenticity." http://blog.thoughtcrime.org/ssl-and-the-future-of-authenticity (2011).

[44] B. Krebs. "The New Face of Phishing." http://blog.washingtonpost.com/securityfix/2006/02/the$_n ew_f ace_o f_p hishing_1$.*html*.

[45] E. Nigg. "Untrusted Certificates." https://blog.startcom.org/?p=145.

[46] Microsoft Corporation. "Microsoft Security Bulletin MS01-017." http://www.microsoft.com/technet/security/bulletin/ms01-017.mspx (2001).

[47] P. Hallam-Baker. "The Recent RA Compromise." http://blogs.comodo.com/it-security/data-security/the-recent-ra-compromise/ (2011).

[48] "Verisign." http://www.verisign.com/.

[49] "Godaddy." http://www.godaddy.com/.

[50] J. Bednar. "Whom do you trust? Issues of Several CA's authentication mechanisms." http://jooray.soup.io/post/10105517/State-of-art-certificates-Whom-do-you (2003).

[51] H. Lee, T. Malkin, and E. Nahum. "Cryptographic strength of SSL/TLS servers: current and recent practices." In "Proceedings of the 7th ACM SIGCOMM conference on Internet measurement," page 92. ACM (2007).

[52] M. Marlinspike. "Null Prefix Attacks Against SSL/TLS Certificates." http://thoughtcrime.org/papers/null-prefix-attacks.pdf (2009).

[53] M. Marlinspike. "Defeating OCSP With The Character '3'." http://www.thoughtcrime.org/papers/ocsp-attack.pdf (2009).

[54] A. Whitten and J. Tygar. "Why Johnny can't encrypt: A usability evaluation of PGP 5.0." In "Proceedings of the 8th USENIX Security Symposium," pages 169–184 (1999).

[55] S. Schechter, R. Dhamija, A. Ozment, and I. Fischer. "The Emperor's New Security Indicators." In "IEEE Symposium on Security and Privacy," IEEE Computer Society (2007).

[56] R. Dhamija and J. Tygar. "The battle against phishing: Dynamic security skins." In "Proceedings of the 2005 symposium on Usable Privacy and Security," page 88. ACM (2005).

[57] C. Jackson, D. Simon, D. Tan, and A. Barth. "An evaluation of extended validation and picture-in-picture phishing attacks." *Financial Cryptography and Data Security* pages 281–293 (2007).

[58] J. Sobey, R. Biddle, P. C. van Oorschot, and A. S. Patrick. "Exploring user reactions to new browser cues for extended validation certificates." In "Proceedings of the 13th European Symposium on Research in Computer Security," pages 411–427. Springer (2008).

[59] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, , and L. Cranor. "Crying Wolf: An Empirical Study of SSL Warning Effectiveness." In "Proceedings of the 18th USENIX Security Symposium," (2009).

[60] R. Biddle, P. Van Oorschot, A. Patrick, J. Sobey, and T. Whalen. "Browser interfaces and extended validation SSL certificates: an empirical study." In "Proceedings of the 2009 ACM workshop on Cloud Computing Security," pages 19–30. ACM (2009).

[61] T. Ylonen and C. Lonvick. "The Secure Shell (SSH) Protocol Architecture." RFC 4251 (Proposed Standard). http://www.ietf.org/rfc/rfc4251.txt (2006).

[62] D. Wendlandt, D. Andersen, and A. Perrig. "Perspectives: improving ssh-style host authentication with multi-path probing." In "USENIX 2008 Annual Technical Conference on Annual Technical Conference," pages 321–334. USENIX Association (2008).

[63] D. J. Bernstein. "DNSCurve: Usable security for DNS." "http://dsncurve.org".

[64] A. Cowperthwaite and A. Somayaji. "The Futility of DNSSec." In "5th Annual Symposium on Information Assurance (ASIA'10)," pages 2–8. Albany, NY, USA (2010).

[65] D. Bernstein. "Curve25519: new Diffie-Hellman speed records." *Public Key Cryptography-PKC 2006* pages 207–228 (2006).

[66] Zimmermann and Associates LLC. "Philip Zimmermann Creator of PGP." http://www.philzimmermann.com/.

[67] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. "RFC 4880-OpenPGP Message Format." (4880). http://www.ietf.org/rfc/rfc4880.txt.

[68] S. Sheng, L. Broderick, J. Hyland, and C. Koranda. "Why Johnny Still Can't Encrypt: Evaluating the Usability of Email Encryption Software." In "Symposium On Usable Privacy and Security, Poster Session," (2006).

[69] S. Garfinkel and R. Miller. "Johnny 2: a user test of key continuity management with S/MIME and Outlook Express." In "Proceedings of the 2005 symposium on Usable Privacy and Security," page 24. ACM (2005).

[70] A. Jøsang. "An algebra for assessing trust in certification chains." In "Proceedings of the Network and Distributed Systems Security Symposium (NDSS'99). The Internet Society," (1999).

[71] R. Atkinson. "Security Architecture for the Internet Protocol." RFC 1825 (Proposed Standard). "http://www.ietf.org/rfc/rfc1825.txt" (1995).

[72] S. Kent and R. Atkinson. "Security Architecture for the Internet Protocol." RFC 2401 (Proposed Standard). "http://www.ietf.org/rfc/rfc2401.txt" (1998).

[73] S. Kent and K. Seo. "Security Architecture for the Internet Protocol." RFC 4301 (Proposed Standard). "http://www.ietf.org/rfc/rfc4301.txt" (2005).

[74] S. Sakane, K. Kamada, and M. Thomas. "Kerberized Internet Negotiation of Keys (KINK)." "http://www.ietf.org/rfc/rfc4430.txt" (2006).

[75] D. Harkins and D. Carrel. "The internet key exchange (IKE)." RFC 2409. "http://www.ietf.org/rfc/rfc2409.txt" (1998).

[76] C. Kaufman. "Internet Key Exchange (IKEv2) Protocol." RFC 4306. "http://www.ietf.org/rfc/rfc4306.txt" (2005).

[77] T. Close. "Trust Management For Humans." http://www.waterken.com/dev/YURL/Name/ (2004).

[78] T. Close. "Authentication of TLS Upgrade Within HTTP/1.1." http://www.waterken.com/dev/YURL/httpsy/ (2004).

[79] T. Ronda, S. Saroiu, and A. Wolman. "Itrustpage: a user-assisted anti-phishing tool." In "Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008," pages 261–272. ACM (2008).

[80] D. Gambetta. "Can we trust trust." *Trust: Making and Breaking Cooperative Relations, electronic edition, Department of Sociology, University of Oxford* pages 213–237 (2000).

[81] A. Abdul-Rahman and S. Hailes. "A distributed trust model." In "Proceedings of the 1997 New Security Paradigms Workshop," page 60. ACM (1998).

[82] R. Perlman. "An overview of PKI trust models." *IEEE Network* **13**(6), 38–43 (1999).

[83] ICANN. "The new gTLD program factsheet." http://www.icann.org/en/topics/new-gtlds/factsheet-new-gtld-program-14apr11-en.pdf (2011).

[84] P. Resnick and R. Zeckhauser. "Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system." *Advances in Applied Microeconomics: A Research Annual* **11**, 127–157 (2002).

[85] L. Page, S. Brin, R. Motwani, and T. Winograd. "The pagerank citation ranking: Bringing order to the web." *Technical report, Stanford Digital Library Technologies Project, 1998* (1998).

[86] E. Hargittai, L. Fullerton, E. Menchen-Trevino, and K. Y. Thomas. "Trust Online: Young Adults' Evaluation of Web Content." *International Journal of Communication* **4**. http://ijoc.org/ojs/index.php/ijoc/article/view/636 (2010).

[87] B. Pan, H. Hembrooke, T. Joachims, L. Lorigo, G. Gay, and L. Granka. "In Google we trust: Users' decisions on rank, position, and relevance." *Journal of Computer-Mediated Communication* **12**(3), 801–823 (2007).

[88] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. Osvik, and B. de Weger. "MD5 considered harmful today." In "Announced at the 25th Chaos Communication Congress.", URL: http://www. win. tue. nl/hashclash/rogue-ca (2008).

[89] X. Wang, Y. Yin, and H. Yu. "Finding collisions in the full SHA-1." In "Advances in Cryptology–CRYPTO 2005," pages 17–36. Springer (2005).

[90] A. Ornaghi and M. Valleri. "Ettercap." http://ettercap.sourceforge.net/.