

2009 Survey of SSL Deployment

Jason Mansfield, R&D, Anonymizer Inc

12/02/2009

Contents

1	Introduction	2
1.1	Executive Summary	2
1.2	Preface	2
2	Key Sizes	4
2.1	Symmetric	4
2.1.1	Client Support For Strong and Weak Key Sizes	5
2.2	Asymmetric	6
3	Algorithms	8
3.1	Symmetric	8
3.2	Asymmetric	8
3.3	Hashing	9
4	Certificates	10
4.1	Verification Algorithms	10
4.2	Self-Signed Certificates	11
4.3	Wildcard Certificates and Keypair Reuse	11
5	SSL Version 2	13
5.1	Downgrade Attack	13
5.2	Risk and Remediation	13
6	Future Research	15
A	Methodologies and Limitations	16
A.1	Test Environment	16
A.2	Host Discovery	16
A.3	Cipher Suite Enumeration	17
A.4	Certificate Collection	17
A.5	SSL Version 2 Detection	17
A.6	Geographic Attribution	18
B	Determining Client Requirements and Managing Server Settings In Apache 2	19

Chapter 1

Introduction

1.1 Executive Summary

This report presents the findings of an investigation of how SSL is deployed on the Internet. Information was gathered through random selection of nearly 15 million IP addresses, about 42,000 of which were found to run SSL. The following conclusions were drawn:

- Few if any of the collected statistics show an obvious regional bias.
- About 91% of hosts support SSLv2 which has known weaknesses.
- About 80% of hosts support cryptographic key sizes that can be considered weak.
- About 80% of hosts support both weak key sizes and SSLv2 permitting a specific, potentially severe attack.
- Site operators *may* be able to disable SSLv2 and weak key sizes without impacting customers.
- About 44% of SSL peer certificates discovered were self-signed, which some consider insecure.
- About 19% of SSL peer certificates discovered were expired, which some consider insecure.
- About 31% of hosts featured a public key in use on other systems, which some consider insecure.

The remainder of this report presents these findings in greater detail along with brief explanations of the relevant concepts.

1.2 Preface

The Secure Sockets Layer, SSL, is the most widely used mechanism for private communication on the Internet. Every day SSL is used countless times to secure Internet communications from eavesdroppers and to help verify the identity of websites. SSL and its successor Transport Layer Security (TLS) provide several important roles in Internet communication:

Privacy SSL uses encryption to ensure that the data being sent between two parties is not readable by eavesdroppers

Integrity SSL uses encryption to ensure that data being sent between two parties is not altered in transit

Authenticity SSL uses encryption and the certificate Public Key Infrastructure to verify the identities of the parties involved

This report discusses a number of different aspects of SSL beginning with key sizes. Keys are a critical part of encryption and for a given algorithm larger keys offer greater security. However keys can not be of arbitrary sizes and both parties communicating must support a given key size for it to be used. Encryption can be divided into two types: *symmetric* and *asymmetric*. These types serve different purposes and have different benefits and drawbacks. The requirements each type places on keys make them incompatible and a size appropriate for a symmetric key will usually not be appropriate for asymmetric encryption.

Next algorithms will be discussed. Encryption algorithms fill different roles or fill one or more roles in different ways. Like key sizes, an algorithm must be supported by both parties involved to be used. Each algorithm has its own strengths and weaknesses although some are almost always preferable to others (i.e. AES over DES).

The discussion then moves on to certificates which are used to verify the identity of one or both of the parties involved and are sometimes used in the process of establishing a private communications channel. Certificates are often an area of contention regarding what represents reasonable, secure practices.

After certificates, SSL version 2 is discussed. SSL version 2 has known weaknesses but is still widely supported on the Internet. Many sites support a configuration of SSL version 2 which could potentially greatly reduce the security of the private communications channel given a carefully positioned attacker.

Lastly, future research is suggested, along with explanation of the methods used in collecting and processing data and a short guide on determining what clients require to use SSL with a given site.

The data collection and analysis processes that produced this report grew organically from a few initial questions so it may be found lacking in structure and rigor. This is representative of the "personal side project" nature of the research and should not be taken as a reflection of Anonymizer Inc.

Chapter 2

Key Sizes

2.1 Symmetric

In SSL privacy is provided by symmetric encryption. With symmetric encryption a message is encrypted and decrypted using the same key. This is the significance of the term *symmetric*. Size for symmetric keys is expressed in bits and the number of bits determines the number of possible values for the key. Each bit doubles the number of possible values so a single bit key has two possible values ($2^1 = 2$), a two bit key has four possible values ($2^2 = 4$), a three bit key has eight ($2^3 = 8$) and a 32 bit key has over four billion possible values ($2^{32} = 4,294,967,296$).

Brute-force cracking an encrypted message involves attempting all possible keys. Adding a single bit to the length of a symmetric key doubles the possible number of keys therefore doubling the amount of work required for a brute-force attack. A given encryption algorithm, implementation of an algorithm, or incident of use may have weaknesses that reduce the possible values for the key which reduces the work required to crack the message. A sophisticated attacker would combine knowledge of weaknesses and high-performance hardware to break an encrypted message. This has been demonstrated against the widely deployed DES algorithm and the cost of the hardware to crack DES-protected messages is diminishing rapidly.

Perfect secrecy is the property of a cryptosystem where an encrypted message provides no clues in breaking that message. In other words the encrypted message provides no information to reduce the number of keys an attacker would have to search through to recover the original message leaving brute-force as the only cryptographic attack. There are other facts that, if known to an attacker, may reduce the number of keys that attacker might have to search through. Because computers produce consistent output for a given input knowing the state of a computer system before it generated a key may aid an attacker in reducing the cracking effort required.

Assuming no factors are in place to aid an attacker and brute-force cracking is the only option it is widely believed that 128 bit symmetric keys are completely infeasible to crack. If this were true for a cryptosystem, larger keys for that cryptosystem would be unnecessary. Occasionally a weakness is discovered in a cryptosystem that can not be easily changed or replaced. Methods for exploiting a weakness may seem purely theoretical at first but will improve over time. With this in

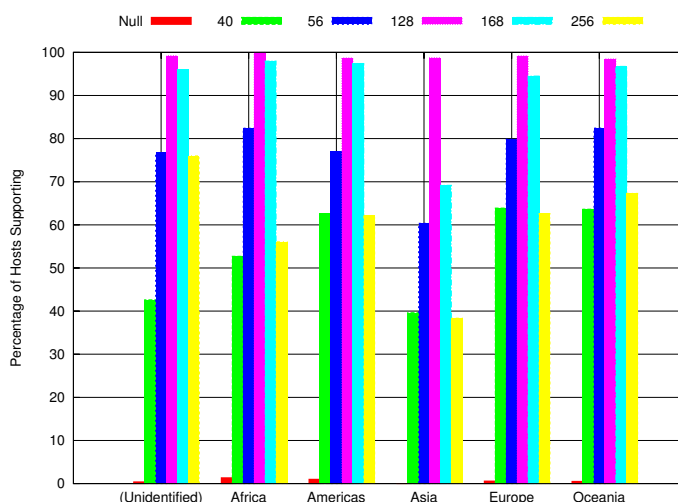


Figure 2.1: Host supporting various symmetric key sizes by region

mind it makes sense to use key sizes larger than 128 to reduce the potential impact of a weakness discovered for a given algorithm, even for algorithms such as RC4 and AES that have seen a great deal of scrutiny. For the purposes of this report symmetric key sizes used in SSL that are larger than 56 bits (128, 168, 256) are considered *strong* and those 56 or smaller (null, 40, 56) are considered *weak*.

Figure 2.1 shows the percentage of hosts in each geographic region which support a given key size. This chart shows that most hosts support at least some strong key size, specifically 128 bit keys. Hosts supporting 40 bit keys are in some regions a minority but support for 56 bit keys (DES) is quite common. The chart indicates a large amount of support for 168 bit keys (3DES) but this is somewhat misleading. The effective key size of 3DES is only 112 bit, the security of which is a matter of context and opinion. Of particular note is the lack of a clear correlation between region and supported key lengths. Of the 38,167 hosts for which cipher suites were enumerated, 30,703 (~80%) supported key sizes of 56 bit or below and 38,125 (~99.9%) supported strong key sizes.

2.1.1 Client Support For Strong and Weak Key Sizes

For the protection of both users and sites cipher suites with weak key sizes should be disabled. Unfortunately this may not be feasible if some clients do not support strong encryption. The question for the site operator concerned with the protection of his visitors is, "Do I have any visitors who don't support strong encryption?"

A generally applicable answer to this question is hard to come by. One approach is to consider what the client software in use supports and how prevalent each piece of client software is. What complicates this approach is the fact that each site will have a different crowd of visitors running different software. A security-oriented site is more likely to have more security-aware visitors running stronger software and settings. A site for computer novices might have more visitors running the default software and configuration for their system. Many sites exist to provide information about browser market share - what percentage of site visitors are using each web browser. The key term is *site visitors*. These statistics are based on data collected from specific sites and as mentioned above the content on a site creates a bias in its visitors. At the time of this writing one source said Internet Explorer has 79% of the market share and Firefox has 18%. Another site claimed that Internet Explorer has 51% of the market share and Firefox has 31%.

<i>Browser</i>	<i>Market Share</i>
Microsoft Internet Explorer	65.71%
Firefox	23.7%
Safari	4.24%
Chrome	3.17%
Opera	2.19%
Netscape	0.35%
Opera Mini	0.30%
Mozilla	0.11%
Konqueror	0.04%
ACCESS NetFront	0.04%
Playstation	0.03%
Danger Web Browser	0.01%
Obigo	0.01%
Microsoft Pocket Internet Explorer	0.01%
Blazer	> 0.01%
WebTV	> 0.01%
Lotus Notes	> 0.01%
BlackBerry	> 0.01%
iCab	> 0.01%
ANT Gallo	> 0.01%

Table 2.1: Browser Market Share Percentages per NET APPLICATIONS

In the absence of clearly accurate information, detailed information was chosen as an example. NET APPLICATIONS¹ provides browser percentages for the top 20 browsers they've detected with market shares including those dipping below .01% as shown in Table 2.1.

All major browsers in Table 2.1 support 128 bit encryption including Internet Explorer as far back as Internet Explorer 3². Opera Mini does not provide full end-to-end encryption but the Opera Mini servers support strong encryption. A cursory investigation did not locate evidence of strong

¹<http://www.netapplications.com/>

²<http://support.microsoft.com/kb/195833>

encryption support in the Blazer, WebTV, or iCab browsers. The TLS 1.0 and 1.1 standards mandate that a form of 3DES be available and TLS 1.2 standard mandates that a form of AES be available so any browser properly implementing TLS supports strong encryption. If these percentages are accurate less than 0.03% of browser market share lacks support for support strong encryption and SSLv3/TLS.

2.2 Asymmetric

There are two contexts for which asymmetric encryption is employed in SSL: key exchange and peer authentication. To use symmetric encryption for privacy both parties need to have the same key. In the case of SSL the two parties have rarely met in advance and must agree on a symmetric key on the fly. The inherent problem is that the two parties are *trying* to establish a secure communications channel so they don't already have one over which to exchange keys without eavesdroppers getting the keys as well. This problem can be solved either with asymmetric encryption (also referred to as *public key encryption*) or with key-exchange protocols.

Key-exchange protocols enable two parties to exchange randomly generated numbers and perform operations on them such that the two parties will reach the same result which is used for key material. An eavesdropper cannot discern the result based solely on the information exchanged. Key exchange will not be discussed further in this report.

Asymmetric encryption involves two different keys used for specific purposes, one for encryption and one for decryption. Because the encryption and decryption processes involve different keys it is referred to as *asymmetric*. In asymmetric encryption one party uses a special process to generate two keys that are tied to each other, referred to as a *keypair*. One key is made freely and publicly available and the other is kept secret. They are referred to as the public key and private key respectively.

Asymmetric encryption can be used for privacy. A piece of data can be encrypted using the public key rendering it unreadable. The public key cannot be used to reverse this process but the private key can. In this way anyone can acquire the means to securely encrypt a message to another party whom they have never met. Only the holder of the private key corresponding to a specific public key can read messages encrypted with that public key.

A somewhat counterintuitive process can employ asymmetric encryption to provide authenticity. The holder of a private key can encrypt a message using that private key. Anyone with access to the public key can decrypt the encrypted message. The significance of this is that only the corresponding public key can decrypt the message, not any other public key. If a given public key can decrypt a message, that message could only have been encrypted by someone holding the corresponding private key. Thus a message can be created for which anyone can verify the author even by parties who have never met the author. This process is referred to as *signing*.

The relation between the size of an asymmetric key and its strength is different than the relation for a symmetric key and its strength. With symmetric keys all possible values are functional keys. Asymmetric keys, on the other hand, require special mathematical properties. A 128 bit symmetric key has 2^{128} possible values because no special properties are required for a given value to be a key. By comparison only a relative handful of those values will have the properties required to function as an asymmetric key meaning many fewer values will have to be tried to break encryption that uses a key of that length. To achieve a number of possible keys equivalent to what's used in symmetric encryption, much larger keys are required. While symmetric key sizes like 128 bit and 256 bit are considered sufficiently strong, asymmetric keys (such as RSA or DSS) must be 1024 bit and larger to be considered strong.

The vehicle for verifying the identity of a host using SSL is the *certificate*. A certificate contains the public key of a host, information to identify them such as the website address, and other important information such as the expiration of the certificate and the algorithms in use to verify that host's identity. All of this information is cryptographically signed by an implicitly trusted organization called a *certificate authority* (CA). Each CA publishes a certificate with their public key and this public key can be used to verify the signature of a host's certificate. CAs are trusted by having

their certificate included with the operating systems, cryptographic libraries, and/or user software such as web browsers. Having an expiration in a certificate serves at least two purposes: it requires hosts to renew their certificates keeping CAs in business and it sets a limit on the length of time a stolen or cracked private key can be used to impersonate a host. If an attacker recovers a host's private key from a discarded backup tape for example, they can only impersonate the host in question until the certificate expires. In practice this utility of certificate expiration is rarely useful as few if any CAs enforce a policy of no keypair reuse. Because few if any CAs enforce such a policy and many site operators are unaware of the issue of key reuse, signing keys can be reused well beyond their recommended lifetimes. There are two solutions: always generate a new keypair or use larger keys. The latter may be infeasible if support is required for very old browsers that cannot support keys larger than 1024. It is worth noting that 1024 bit keys are still considered secure for the immediate future.

When obtaining a certificate, public key size is the choice of the site operator and using smaller or larger keys does not change the work required on the part of the CA. Strangely, many CAs will charge different prices based on the size of *symmetric* keys supported by the host. Luckily for the customer this setting can easily be changed temporarily for the purposes of compliance or to suit their own whim.

Table 2.2 shows the percentages of key sizes found in certificates collected from 38,664 hosts where 1024 bit keys clearly dominate. Changing the key size used in a certificate requires getting a new certificate. If that certificate must be signed then purchasing a new certificate is required. This expense means that changing key size will often not make good business sense for a given organization. Public key sizes do not change frequently so an immediate recollection of this data is likely to retain the same results. Future repetition of this effort might reveal trends indicating a relative shift from one key length to another.

<i>Key Size</i>	<i>Count</i>	<i>%</i>
512	761	2%
768	1171	3%
1024	32189	83%
1536	4	.01%
2048	4375	11%
4096	164	.4%

Table 2.2: Public Key Sizes for Peer Certificates

Chapter 3

Algorithms

3.1 Symmetric

Symmetric ciphers provide the privacy aspect of SSL and part of the purpose of SSL is to find a common ground in cipher support between two peers. Figure 3.1 shows the percentage of hosts supporting a given underlying cipher by region. In this case DES and 3DES are both simply considered as DES and incidents of a host supporting multiple key sizes for a given cipher are simply considered as supporting the cipher. Even if a host supports several key sizes for AES it is listed for AES only once: it either supports AES or it doesn't. Clearly support for DES and RC4 are nearly ubiquitous while support for RC2 is fairly uncommon. Similarly to Figure 2.1 there does not appear to be an obvious trend by region.

None of these ciphers (with the exception of NULL) have significant known weaknesses. However, RC2 and DES are block ciphers using 64 bit blocks. This represents a relatively limited number of plaintext and ciphertext blocks and as the amount of data transmitted increases the probability of a duplicate transmission increases. These duplicate blocks leak information about the plaintext and over time could reduce the security of the session. This probability doesn't become significant until (on average) about 32 GB of data have been transmitted. Most SSL sessions do not carry nearly that much data so it shouldn't be a concern but because it's an issue of probability on some rare occasions it could be a risk. With this in mind support for RC2 should be disabled if possible. This survey only observed RC2 being used with a 40 bit key so it should be disabled as a weak cipher if not for its block size. As stated in 2.1.1 on page 6 TLS 1.0 and 1.1 mandate that some form of 3DES be supported so only those site operators who can ensure that all clients support some other form of strong encryption can disable DES.

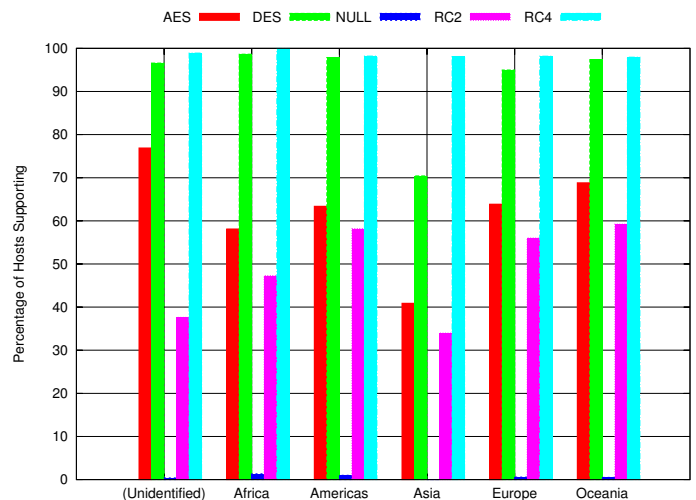


Figure 3.1: Percentage of Cipher Support By Region

3.2 Asymmetric

Asymmetric ciphers are used for the agreement of symmetric keys and for authentication of the host. As shown in Table 3.1 support for key agreement with RSA is nearly ubiquitous while support for

key agreement with Diffie-Hellman is somewhat rare. In SSLv2 key exchange is performed only using RSA.

SSLv2 supports a key exchange mechanism called *Clear Key* (CK) where some of the key material is encrypted using RSA and the rest is transmitted in the clear. This facilitates the use of ciphers with large keys but offers the security of smaller keys to comply with export regulations. For example, 128 bit RC4 could be used with 40 bits of the key transmitted RSA encrypted and the remaining 88 bits transmitted in the clear. Obviously Clear Key intentionally weakens the strength of the encryption and should be disabled if possible.

Peer authentication usually requires the use of a certificate. A certificate has a public key and that key is algorithm specific. Therefore when certificate-based authentication is taking place the algorithm in use was specified when the certificate was configured on the server. While the algorithm for authentication can be changed by generating a new keypair and new certificate it can't be decided on the fly. However, of the 38,167 hosts for which cipher suites were enumerated, three appeared to support both RSA and DSS. Upon inspection, those hosts would provide an RSA-based certificate when only presented with RSA cipher suite choices and a DSS-based certificate when only presented with DSS cipher suite choices. Those three hosts support RSA and DSS simultaneously and the appropriate certificate and keypair are selected based on client support and preference.

<i>Region</i>	<i>Diffie-Hellman</i>	<i>RSA</i>
<i>Unidentified</i>	937 (60%)	1558 (100%)
Africa	63 (43%)	148 (100%)
Americas	8785 (44%)	19855 (99.95%)
Asia	33 (21%)	159 (100%)
Europe	5200 (46%)	11205 (99.88%)
Oceania	2207 (42%)	5212 (99.87%)

Table 3.1: Key Agreement Support By Region

<i>Region</i>	<i>DSS</i>	<i>RSA</i>	<i>anon</i>
<i>Unidentified</i>	1 (.06%)	1558 (100%)	14 (.9%)
Africa	0 (0%)	148 (100%)	1 (.68%)
Americas	6 (.03%)	19859 (99.97%)	214 (1.1%)
Asia	0 (0%)	159 (100%)	1 (.63%)
Europe	13 (.12%)	11207 (99.89%)	176 (1.57%)
Oceania	5 (.1%)	5214 (99.9%)	57 (1.09%)

Table 3.2: Auth Cipher Support By Region

3.3 Hashing

Hashing is used in SSL as part of the certificate verification process and to verify that a message hasn't been tampered with in transit. For the latter case our tool only provided support for MD5 and SHA. Table 3.3 shows the percentage of hosts supporting each by region. Both are supported almost everywhere and neither is clearly preferred over the other.

MD5 and to a lesser degree SHA have received some bad press in recent years as having been "broken". While these claims do have merit under some circumstances MD5 and SHA are safe for this application. The known weaknesses of these algorithms allow an attacker to create a new message with the same authentication code using a significantly reduced amount of computing time. The authentication codes used in messages sent with SSL last for at most a few seconds, well below the amount of time required to forge a useful message even considering the weaknesses and a large amount of computing power for the foreseeable future.

<i>Region</i>	<i>MD5</i>	<i>SHA</i>
<i>Unidentified</i>	1510 (97%)	1528 (98%)
Africa	146 (97%)	148 (100%)
Americas	19271 (97%)	19636 (99%)
Asia	154 (97%)	116 (73%)
Europe	10904 (97%)	10852 (97%)
Oceania	4845 (93%)	5136 (98%)

Table 3.3: HMAC Hash Support By Region

Chapter 4

Certificates

Certificates serve the authentication function of SSL. A certificate presents information that should be sufficient to verify the identity of a peer. When a certificate is signed by a certificate authority the certificate also includes information that should be sufficient to verify the authenticity of the certificate itself. If any of the information presented doesn't properly validate, the user is alerted to a problem. In practice this process works well to alert the user that something may be wrong but can't completely assure the user that nothing malicious is happening. It's also worth noting that notifying the user that something is wrong with the certificate validation has been shown in at least one study to often fail to invoke a security-conscious response¹.

Overall 59,841 certificates were collected from 38,664 distinct hosts. 2.2 explained the use of expiration to limit the time a stolen/broken key can be used for site impersonation. 7403 (19%) of all peer certificates were expired, 2853 of which were signed by a third party and 4550 were self-signed. Strangely, eight certificates were discovered that had expiration dates prior to the date when they began validity, all of which were peer certificates. Six of those were self-signed certificates for network devices, one was a self-signed certificate for a website, the last was actually signed by a commercial certificate authority for a website. All of those eight certificates were expired.

A brief effort was made to show popularity of various certificate authorities by region. In the data collected many certificate authorities had multiple signing certificates associated with them. The time required to accurately attribute all of these signing certificates to the appropriate certificate authorities placed it outside the scope of this effort.

4.1 Verification Algorithms

For certificate verification two algorithms must be specified. One is a hashing algorithm to reduce the rather large certificate data down to a small fingerprint. The other is a signing algorithm used to encrypt the hash such that it can be verified with a certificate's public key. The majority of peer certificates collected use SHA1 with RSA (~72%) or MD5 with RSA (~28%) as shown in Table 4.1. GOST refers to a set of state standards from the former Soviet Union.

In the context described in 3.3 the weaknesses of MD5 are not a significant risk because the authenticated messages only have value for a few seconds at most. Site certificates are

<i>Algorithm</i>	<i>Count</i>	<i>~%</i>
SHA1withRSA	27,719	72%
MD5withRSA	10,911	28%
SHA1withDSA	26	>1%
SHA256withRSA	3	>1%
SHA384withRSA	1	>1%
SHA512withRSA	2	>1%
MD2withRSA	1	>1%
GOST R 34.11/34.10-2001 1	1	>1%

Table 4.1: Signing Algorithm by Peer Certificate

¹<http://lorrie.cranor.org/pubs/sslwarnings.pdf>

intended to last a year or more and certificate authority certificates have much longer expirations. The weaknesses in MD5 were demonstrated in 2008² to enable the creation of a rogue CA certificate. This certificate could be used by its creators to create and sign certificates which will allow the creators to impersonate arbitrary hosts in such a way that client software will provide no warnings.

Many hosts provided certificates beyond their own to establish a verification chain. Of these certificates 16,831 used SHA1, 3954 used MD5, 391 used MD2, and 1 used SHA512. This seems to indicate that use of MD5 in certificates is waning but it will be difficult to tell until similar data is collected in the future.

4.2 Self-Signed Certificates

Two important properties of certificates are the *subject* and the *issuer*. The subject notes who the certificate refers to. The issuer notes who signed that certificate and therefore who can verify it. The subject for a website certificate is usually the domain of the website which the browser compares to the URL provided. The certificate is authenticated by another certificate, that of the issuer. This in turn may be authenticated by another certificate and so on. Ideally this chain terminates at a certificate the browser or other client software trusts implicitly. Such an implicitly trusted certificate is called *Root Certificate*. Because there is no further certificate to authenticate a root certificate against its subject and issuer are the same. When the subject and issuer are the same this is called a *self-signed certificate*. Root certificates are trusted not because of who signed them but because they came installed with the operating system and/or client software.

Anyone can create a self-signed certificate for use on their own systems. Such a certificate doesn't have a chain of trust that terminates at an implicitly trusted certificate. For this reason self-signed certificates will trigger client warnings. The risk of self-signed certificates is that the browser can't authenticate the certificate properly which places that burden on the user. Users are very poor at remembering the cryptographic information necessary to thoroughly recognize that the certificate is what they should be expecting.

Of the peer certificates collected 16,986 (44%) were self-signed with only 8,863 distinct subjects among them. 13,004 distinct public keys were identified among self-signed certificates and 12,114 of those public keys were unique to one host.

4.3 Wildcard Certificates and Keypair Reuse

A wildcard certificate is a single certificate that applies to multiple domains. A wildcard certificate is recognized by the presence of an asterisk for the hostname portion of the domain name (*.example.com instead of www.example.com). Wildcard certificates are more costly than standard certificates but can be applied to multiple sites in the same domain so they can potentially be more cost-effective than getting standard certificates for each site. Wildcard certificates can also be easier to manage than standard certificates since the site operator doesn't need to manage a certificate for each site.

A certificate maps a public key to a site and, in the case of wildcard certificates, multiple sites. When wildcard certificates are used each site must use the same keypair. The more sites using a given key the more systems must be kept up to date and secure to protect the private key. If any single host is sufficiently breached to leak the private key all sites using that certificate can be impersonated by attackers. The greater the number of hosts with a given keypair the greater the chance one of those hosts will allow a private key to be stolen.

The certs collected presented 24,220 distinct subjects, 1,801 (~7%) subjects were distinct wildcard subjects, 1,701 (~7%) were not self-signed, and 172 (~.7%) were not yet expired. Wildcard

²<http://www.win.tue.nl/hashclash/rogue-ca/>

deployment accounted for 5,056 (~13%) of the 38,664 peers. The five largest wildcard deployments found were 564, 341, 304, 190, and 158 hosts.

Keypairs can be reused in completely different certs as well. The five largest incidents of keypair reuse not for wildcard certs were 534, 224, 218, 171, and 146 hosts. The first, third, and fourth of those were snake oil or localhost keypairs and certificates: certificates installed or generated when the SSL software is installed. In these cases the keypairs and certs came with the package and anyone who can retrieve the package can potentially impersonate the sites using those keys or possibly decrypt their traffic. One of the snake oil keypairs was used by a host to generate a certificate signing request and was subsequently signed by a commercial CA. The second keypair count listed belonged to single hostname from a large content delivery network.

To put these host counts in perspective, the IPs tested were randomly generated and represent around one half of one percent of allocated IP space. If these numbers are representative then there are two hundred times as many hosts with those certificates and keypairs as the number discovered. Overall 11,867 (~31%) hosts were using the same keypair as another host and 28,472 distinct public keys were found.

Chapter 5

SSL Version 2

Despite having known weaknesses in the protocol (not in the encryption) SSL version 2 is still widely supported on the Internet. Those weaknesses include the potential ability for an attacker to add data to the end of a communication, weak mechanisms for message authentication, and the potential ability for an attacker to force the connection to use crackable encryption.

5.1 Downgrade Attack

The ability for an attacker to force weak encryption is referred to as a *downgrade attack*. This attack occurs when a client sends its list of supported cipher suites to the server. If this list is intercepted by an attacker the attacker can remove all the strong ciphers from the list and forward it on to the server. When the SSL connection is established encryption will be in place and the attacker cannot eavesdrop. However, if the encryption is weak enough and/or the attacker has sufficient computing resources the attacker can crack the encryption and read the data. Given a high level of computing power and a long-lived, high-activity connection an attacker may be able to crack the encryption sufficiently quickly to inject their own validly encrypted data into the conversation.

Region	Hosts	~%
<i>Unidentified</i>	1235	79%
Africa	123	83%
Americas	15618	79%
- North America	15070	79%
Asia	90	57%
Europe	9108	81%
Oceania	4340	83%

Table 5.1: Hosts Susceptible To Downgrade Attack By Region

With SSLv3 and TLS the end of negotiation includes an exchange of checksums for the negotiation process so tampering is detected. Therefore SSLv3 and TLS connections fail when the downgrade attack is attempted. With SSLv2 the downgrade attack is less of an issue for privacy if no weak ciphers are enabled. If an attacker tampers with the list of supported ciphers sent by the client adding an unsupported cipher suite then the negotiation will complete using encryption the client can't understand. If a client sends a list of cipher suites only containing strong encryption an intermediate attacker can only choose from that list and in turn can only force the selection of a specific suite that uses strong encryption which the attacker cannot break.

Of the 38,167 hosts probed for cipher suites, 30,703 (~80.4%) supported weak ciphers and 30,514 (~79.9%) support both SSLv2 and weak ciphers permitting the downgrade attack. This is presented in 5.1.

5.2 Risk and Remediation

Of the 42,626 hosts with which SSL was negotiated, 38,742 (~90%) of them supported SSLv2. All of the browsers noted in section 2.1.1 to support strong encryption also support SSLv3 or TLS 1.0 or

higher. Again, if the browser market share data from Net Applications is correct less than 0.03% of the market *might not* support SSLv3 or TLS. For most site operators disabling support for SSLv2 can improve security for their visitors with very little risk of incompatibility for visitors. Site operators should take steps to determine if their visitors require SSLv2 support and disabling it if possible. Guidelines for making this determination can be found in Appendix B.

Chapter 6

Future Research

This research effort captured the ways in which each item of interest varied by region and sub-region. However, the data collected only represents the state of various hosts at one point in time. Therefore it cannot give any indication of trends over time. Repeating this effort in the future will present the opportunity to potentially answer the following questions:

- What regions are experiencing the largest growth in SSL deployment?
- How are the relative strengths of symmetric and asymmetric key lengths changing?
- As SSL certificates expire, are site operators generating new keypairs?
- How popular are the various Certificate Authorities?
- Are browsers/libraries inspecting all the relevant certificate fields (e.g. *not before*)?
- Are there any trends that could explain support for NULL ciphers on some hosts?

Appendix A

Methodologies and Limitations

A.1 Test Environment

All tests were performed from Anonymizer's dedicated R&D network. This network is connected to the Internet via a fairly modest business-class broadband connection. Most work was done from a Debian 5.0 (Lenny) virtual machine. Some processor-intensive work was done on a more powerful Ubuntu 9.04 (Jaunty) workstation system.

All network tests were performed from the same system using a consistent IP. Many of these tests required repeatedly connecting to the same system. While efforts were made to spread these connections out over time some hosts or intermediate network devices may have perceived this connections as a threat and blocked the source IP.

Some rounds of testing were done days or weeks apart. This may have caused variations in the number of hosts tested due to failures of remote and intermediate systems.

A.2 Host Discovery

Host discovery consisted of randomly generating IP addresses within IP space that may be in use and trying to establish an SSL connection to that host on TCP port 443. Establishing an SSL socket with a given host made that host a candidate for further inspection.

The tool for discovery was written in perl using the `IO::Socket::SSL` module. This tool would make four `int(rand(255))` calls and join them with dots to form an IP address. If the first octet was not in IANA's list of allocated IP space¹ that address was ignored. If the IP address was in RFC1918 space that address was ignored. The tool would then note the generated address and attempt an SSL connection to it on port 443 with a timeout of ten seconds. If the returned `IO::Socket::SSL` object evaluated to true the address and negotiated cipher suite were noted. In the above context *noting* a piece of information consisted of printing it to STDOUT. All output was captured with the `tee` command so it could be recorded and displayed in real-time.

Because most IPs on the Internet do not feature an SSL service running on port 443 the majority of attempts failed due to the ten second timeout. For the sake of performance the tool was made to run many parallel discovery attempts. This was accomplished by having the tool `fork()` of a number of children as determined by a variable. Each child process performed attempts independently and given the size of the search space no attempts were made to prevent repeat tests of a given IP address. The few instances of repeat tests against a given IP address were factored out before subsequent tests.

Despite the parallelism of the tool, the biggest limitation of this method was speed. The bulk of discovery took place over a period of about two weeks and only about 0.5% of allocated IP

¹<http://www.iana.org/assignments/ipv4-address-space/>

addresses was covered. A larger pool of hosts could have been discovered using a more aggressive port-scanning tool, however one concern was being seen as a potential threat by intrusion detection systems and blocked.

A.3 Cipher Suite Enumeration

Cipher suite enumeration involved presenting an SSL host with a large list of cipher suites our client could support. When the host selected one suite the connection would be torn down and the selected suite would be removed from the list of suites presented. This process would be repeated for each host until that host no longer found a suite it supported among the list of suites presented to it.

Java was chosen for the initial version of the suite enumeration tool due to the relative ease and safety in applying parallelism to the testing workload. It was later determined that the cryptography provider in use on our test platform did not provide a comprehensive list of supported cipher suites. No attempt was made to use different cryptography providers.

A second tool for suite enumeration was created in perl, again using the `IO::Socket::SSL` module. For this tool the list of cipher suites was populated from the suites listed in the *ciphers*² man page excluding those listed as “not implemented”. This tool functioned as described above with one exception.

To achieve parallelism this tool would *fork()* children but those children had to work out of a central queue of tests so coordination was required. The queue was created by noting that a test needed to be performed for each suite against each host. Given 42,626 hosts and 51 cipher suites, 2,173,926 connections were needed to complete all tests. Each test consisted of determining if the SSL connection could be created when the host was presented with only the desired cipher suite. The tool would read the output from previous runs and eliminate tests already performed from the queue. This allowed the tool to be stopped and restarted without it duplicating tests. To ensure that tests against a single host or network were spread out over time the queue was shuffled before being processed. Using this method each test was independent of all other tests. This simplified leveraging parallelism against the test load but caused unnecessary tests because the optimization mentioned in the first paragraph could not be applied.

A.4 Certificate Collection

The original tool for cipher suite enumeration written in Java would also collect the certificate chain presented by the remote host. An `SSL_SOCKET` object can provide an `SSL_SESSION` object which has a *getPeerCertificateChain()* method. This method returns an array of `X509CERTIFICATE` objects in order, starting with the direct peer’s certificate. This array was written to disk using Java’s serialization mechanism. A separate Java tool would transform the chain into a series of simple text files.

A.5 SSL Version 2 Detection

To determine which hosts supported SSL version 2 a simple perl script was written using the `IO::Socket::SSL` module. An SSL connection was created for each IP explicitly configured to SSLv2. If this connection succeeded the host was assumed to support SSL version 2. If the connection failed it was noted for retest. Retests were performed at least several hours apart to reduce the impact of temporary outages. No IP was tested more than three times. Hosts that failed all three times were considered unknown.

Because the number of tests to be performed was relatively small no parallelism was used.

²From the `openssl` package

A.6 Geographic Attribution

Attributing hosts to a geographic location was performed using a licensed copy of a commercial IP geolocation database providing details at the national and city levels. Countries were grouped according to the United Nations classification for regions and subregions³.

The primary difficulty in geographic attribution was the accuracy of the databases being used. In some instances the city database would attribute an IP to a city such as Tokyo or Bucharest but the nation-level database could not discern the country for the IP. Instances of this were resolved using the results of *whois* queries and web searches. The same methods were used to attribute IP addresses listed as "Anonymous Proxy", "Satellite Provider", or similar labels.

Many IP addresses in question couldn't not be attributed to a country by the databases being used. In some cases large groups of these IPs shared the same netblock. These were attributed based on *whois* query results. Those unattributed IPs not part of larger blocks were left unattributed and are represented on the charts provided as (?) or (UNIDENTIFIED).

Geographic attribution could have been greatly improved with the use of multiple IP geolocation sources and the use of more attribution "by hand" using *whois* queries and other open source intelligence sources (OSINT). The use of multiple IP geolocation sources would introduce conflicting information for some IP addresses. The use of OSINT collection would likely be highly labor intensive.

³<http://millenniumindicators.un.org/unsd/methods/m49/m49regin.htm>

Appendix B

Determining Client Requirements and Managing Server Settings In Apache 2

In the interest of flexibility most popular web servers feature support for various protocol versions and encryption components, even though some are potentially insecure. If possible, site operators should disable less secure options like SSLv2 and weak key/small block ciphers. Some sites have visitors that require support for SSLv2 or weak ciphers and disabling those would make the site unusable for those visitors. Before disabling potentially insecure features site operators should collect ample statistics on the SSL parameters used by their visitors to see if support for those features is required.

It is a fairly simple procedure to configure Apache 2 and `mod_ssl` to log SSL negotiation in sufficient detail to determine what clients are using. The log settings can generate dozens of log lines per request. Leaving high levels of logging on busy web servers can increase the load on those servers and potentially fill log storage. If a server is run with this level of logging the server operator should consider running daily log rotation that discards the excess log data.

In the configuration section for SSL (indicated by *SSLEngine On*) the *LogLevel* parameter should be set to *debug* and apache should be restarted/reloaded. To reiterate, this will generate an enormous amount of log data and may overwhelm the website if the server is very busy. With debug level logging the *ErrorLog* file will have entries for each SSL connection. Among the SSL debugging information will be lines like these:

```
[Tue Nov 17 09:51:19 2009] [info] Connection: Client IP: 192.168.1.33, Protocol:
TLSv1, Cipher: DHE-RSA-AES256-SHA (256/256 bits)
```

```
[Tue Nov 17 09:54:53 2009] [info] Connection: Client IP: 192.168.1.33, Protocol:
SSLv2, Cipher: DES-CBC3-MD5 (168/168 bits)
```

In the above example we see one connection using TLSv1 and one using SSLv2. In this example there are SSLv2 clients so disabling SSLv2 may be problematic. We also see one client using AES and another using 3DES. The *ciphers* man page with the `openssl` package contains a list that maps `openssl` names such as DHE-RSA-AES256-SHA to the more informative suite name `TLS_DHE_RSA_WITH_AES_256_CBC_SHA`.

Support for SSLv2 is controlled by the *SSLProtocol* configuration directive. To disable SSLv2 add the `-SSLv2` argument. The end product may look like

```
SSLProtocol all -SSLv2
```

Cipher suite support is controlled by the *SSLCipherSuite* configuration directive. This can be configured on a per-directory basis but that practice is discouraged. The meaning of the arguments is explained in the *ciphers* man page of the `openssl` package. A good setting is *HIGH:MEDIUM* but

this is not practical of some clients require support for weaker encryption. The `openssl ciphers -v` command can display the list of ciphers a given argument indicates:

```
$ openssl ciphers -v HIGH:MEDIUM:-3DES:-RC2
ADH-AES256-SHA          SSLv3 Kx=DH          Au=None Enc=AES(256) Mac=SHA1
DHE-RSA-AES256-SHA     SSLv3 Kx=DH          Au=RSA  Enc=AES(256) Mac=SHA1
DHE-DSS-AES256-SHA    SSLv3 Kx=DH          Au=DSS  Enc=AES(256) Mac=SHA1
AES256-SHA             SSLv3 Kx=RSA         Au=RSA  Enc=AES(256) Mac=SHA1
ADH-AES128-SHA         SSLv3 Kx=DH          Au=None Enc=AES(128) Mac=SHA1
DHE-RSA-AES128-SHA    SSLv3 Kx=DH          Au=RSA  Enc=AES(128) Mac=SHA1
DHE-DSS-AES128-SHA    SSLv3 Kx=DH          Au=DSS  Enc=AES(128) Mac=SHA1
AES128-SHA             SSLv3 Kx=RSA         Au=RSA  Enc=AES(128) Mac=SHA1
ADH-RC4-MD5            SSLv3 Kx=DH          Au=None Enc=RC4(128) Mac=MD5
RC4-SHA                SSLv3 Kx=RSA         Au=RSA  Enc=RC4(128) Mac=SHA1
RC4-MD5                SSLv3 Kx=RSA         Au=RSA  Enc=RC4(128) Mac=MD5
RC4-MD5                SSLv2 Kx=RSA         Au=RSA  Enc=RC4(128) Mac=MD5
```

Based on this output site operators can see exactl which cipher suites are enabled for a given `SSLCipherSuite` value.