Prying open Pandora's box:
KCI attacks against TLS

Clemens Hlauschek, Markus Gruber, Florian Fankhauser, Christian Schanes

RISE – Research Industrial Systems Engineering GmbH

{clemens.hlauschek, markus.gruber, florian.fankhauser, christian.schanes}@rise-world.com

Abstract

Protection of Internet communication is becoming more common in many products, as the demand for privacy in an age of state-level adversaries and crime syndicates is steadily increasing. The industry standard for doing this is TLS. The TLS protocol supports a multitude of key agreement and authentication options which provide various different security guarantees. Recent attacks showed that this plethora of cryptographic options in TLS (including long forgotten government backdoors, which have been cunningly inserted via export restriction laws) is a Pandora's box, waiting to be pried open by heinous computer whizzes. Novel attacks lay hidden in plain sight. Parts of TLS are so old that their foul smell of rot cannot be easily distinguished from the flowery smell of 'strong' cryptography and water-tight security mechanisms. With an arcane (but well-known among some theoretical cryptographers) tool, we put new cracks into Pandora's box, achieving a full break of TLS security. This time, the tool of choice is KCI, or *Key Compromise Impersonation*.

The TLS protocol includes a class of key agreement and authentication methods that are vulnerable to KCI attacks: non-ephemeral Diffie-Hellman key exchange with fixed Diffie-Hellman client authentication – both on elliptic curve groups, as well as on classical integer groups modulo a prime. We show that TLS clients that support these weak handshakes pose serious security concerns in modern systems, opening the supposedly securely encrypted communication to full-blown Man-in-the-Middle (MitM) attacks.

This paper discusses and analyzes KCI attacks in regard to the TLS protocol. We present an evaluation of the TLS software landscape regarding this threat, including a successful MitM attack against the Safari Web Browser on Mac OS X. We conclude that the insecure TLS options that enable KCI attacks should be immediately disabled in TLS clients and removed from future versions and implementations of the protocol: their utility is extremely limited, their raison d'être is practically nil, and the existence of these insecure key agreement options only adds to the arsenal of attack vectors against cryptographically secured communication on the Internet.

1 Introduction

The TLS protocol [1, 2, 3] is probably the most widely used cryptographic protocol on the Internet. It is designed to secure the communication between client/server applications against eavesdropping, tampering, and message forgery, and it also provides additional, optional security properties such as client authentication. TLS is an historically grown giant: its predecessor, SSL [4, 5], was developed more than 20 years ago. It is designed for compatibility between all kinds of devices, and to support a large range of different cryptographic primitives to cater to different purposes, security, and authentication requirements, as well as to provide alternatives to switch to in case new attacks against specific primitives are discovered.

The TLS protocol consists of two principal layers: the TLS handshake protocol, and the TLS record protocol. During the TLS handshake protocol, client and server negotiate the cipher suite to use. A cipher suit is the set of algorithms (1) for establishing a shared secret (key exchange), (2) for authenticating the server to the client and, optionally, the client to the server, and (3) for encrypting and authenticating the communication data sent with the TLS Record Protocol. Some cipher suites provide distinctly different security guarantees than others. For example, cipher suites that contain the null cipher provide no encryption at all, and thus do not guarantee confidentiality in a trivial way. Another example are cipher suites that dispense with server authentication. They are trivially susceptible to Man-in-the-Middle (MitM) attacks in any reasonable threat model applicable in a network setting (i.e., a threat model that as-

sumes an active attacker, capable of intercepting, modifying, injecting, and blocking messages). Similarly, the RSA key exchange algorithm in TLS is less secure than an ephemeral (Elliptic Curve) Diffie-Hellmann key exchange algorithm[1]. The latter provides *Perfect Forward Secrecy* (PFS) [7], while the former does not. Perfect Forward Secrecy guarantees that previous sessions remain secure even after long-term secret key material has been compromised. All these examples are well-known among security researchers and practitioners.

However, what lacks widespread insight is the fact that some cipher suites are vulnerable to *Key Compromise Impersonation* (KCI) [8] attacks and that the failure to secure against KCI attacks opens a dangerous security hole, especially when considering the way client certificates are handled in actual systems. Cipher suites that are not resistant to KCI are routinely advertised by most TLS clients such as web browsers. This unnecessarily exposes the communication to an avoidable risk. Software vendors, not aware of the risk, often make it easy to install a client certificate maliciously in a victim's web browser, or get an app to use a maliciously installed client certificate – essential ingredients in successfully exploiting KCI vulnerabilities. To the best of our knowledge, this is the first work that investigates and analyses the threat of KCI attacks in the context of TLS. Clients that support TLS client authentication with one of the `ClientCertificateType` options listed in Table 1 are vulnerable.

Contributions. In this paper, we present an hitherto undiscussed, novel way to attack TLS. We analyze and evaluate the threat posed by TLS cipher suites and authentication options that are vulnerable to Key Compromise Impersonation (KCI) attacks. To the best of our knowledge, we are the first to do so. We show how to implement a successful MitM attack against affected systems such as Apple's Safari Web Browser on Mac OS X. More specifically:

- We explain KCI attacks against TLS in detail, sculpt out the necessary preconditions for a successful attack, and discuss how to meet these preconditions in practice.
- We present an analysis of the TLS software / library landscape regarding this threat.
- We cooperated with and assisted vendors that have critically affected software products, such as Apple's Safari web browser, during a responsible disclosure process.

[1] Supposedly, under the right circumstances, i.e. if it would have been implemented correctly in the TLS protocol. Cf. the recent Logjam attack on DH in TLS [6].

Furthermore, we present a convincing argument for the IETF TLS working group to drop KCI-vulnerable cipher suites from TLS 1.3, the next version of TLS which is currently in draft-status and actively developed.

2 Related Work

2.1 Transport Layer Security

During the evolution of the Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols, many new cipher suites and key agreement options have been added, such as the Elliptic Curve Cryptography cipher suites [9]; and industry best-practice advice is to disable protocol versions and cryptographic primitives that are considered to be insecure by now. This includes cipher suites with key size smaller than 128 bit, cipher suites that contain insecure hash functions such as MD5 [10, 11], the RC4 algorithm [12, 13], and the SSLv2 protocol [14, 15]. Tools such as `sslyze` [16] and `sslscan` [17] can be used to test TLS installations for such insecure configurations, as well as for faulty implementations. Recently, recommendations for secure use of TLS have been published by the IETF as RFC 7525 [18]. While these recommendations do not fail to discuss Perfect Forward Secrecy (PFS), Key Compromise Impersonation (KCI) is not even mentioned. Although prohibiting non-PFS cipher suites, as often recommended, would imply the prohibition of KCI-vulnerable cipher suites in TLS, security recommendations should always discuss KCI explicitly: system implementers sometimes have various reasons to forgo PFS guarantees in TLS (e.g., the performance advantage of the RSA handshake), but should never have a plausible reason to forgo KCI resistance.

Currently the IETF TLS working group is discussing the next overhaul of the protocol, TLS version 1.3 [19, 20]. Among other things, the TLS working group considers dropping less secure cryptographic options such as the RSA key agreement, dropping problematic features such as TLS renegotiation [21, 22], and considers including new cipher suites according to the state-of-the-art in the field. However, these improvements take years until industry installations will consider them and some legacy software will not upgrade to new standards at all. Therefore, it is not enough to consider the standard but also the many implementations of TLS libraries which are used without further consideration. Not only should the TLS working group drop the support for KCI-vulnerable cipher suites in TLS 1.3, but also should current TLS clients and libraries immediately disable the specific cipher suites discussed in this paper and stop offering them during handshake negotiation, because they pose an immanent risk ranging up to a full break of TLS security,

allowing a successful attacker to eavesdrop, tamper, and forge messages from purportedly secure communication.

This paper adds to the vast array of attacks against the SSL/TLS protocol [14, 23, 24, 25, 26, 27, 28, 29, 30, 21, 31, 32, 33, 34, 35, 36, 37, 38, 22, 39, 40, 6, 13]. For a good overview and summary of attacks and the history of SSL research consult Meyer's recent PhD thesis [41], or the shorter eprint version [42].

Similar to the recent Logjam attack [6], the attack we present in this paper is based on flaws in the TLS protocol, rather than on implementation errors, such as the related, also recently published, FREAK attack [40]. Similar to the Logjam and the FREAK attacks, our attack is possible because TLS carries for historical reasons the burden of outdated, insecure cryptographic algorithms and options that should have long been deprecated, and completely eradicated and removed from modern systems. However, different from Logjam and FREAK, our attack combines protocol flaws with the way client certificates are often handled in practice, i.e., in our attack scenario it is assumed that it is easy in practice for an attacker to get hold of the secret key corresponding to some client certificate installed at the victim's system. In this paper, we argue that this assumption can be easily made in many practical scenarios.

MitM attacks against TLS can also be orchestrated by subverting the certification authority (CA) infrastructure [43]. An adversary can either try to compromise an insufficiently secured and vulnerable CA (as was, for example, the case in the DigiNotar hack [44]); or the adversary has direct control over a CA (such as state-level adversaries that have legal power over CAs operating under their jurisdiction); and thus be able to issue fraudulent certificates for the purpose of active MitM interception attacks. For the attack presented in this paper, the server certificate does not need to be substituted with a fraudulent certificate. Thus, mitigation mechanisms against MitM attacks such as Perspectives [45], Convergence [46], TACK [47], DANE [48], or certificate pinning are not effectual against KCI-based MitM attacks.

Recently, it was discovered that Lenovo shipped a malware product called Superfish with its consumer laptops [49], which, after inserting its own CA certificate into the browser's trust store, conducted MitM attacks by generating fraudulent certificates on-the-fly in order to inject malicious code in web applications. A similar malware, using our KCI attack presented in this paper, would inject only client certificates into the certificate store, instead of CA certificates, and would not need to replace server certificates. Thus, such a malware would be much more difficult to detect, even by experts.

Moxie Marlinspike's *SSLStrip* attack [31,32] works by downgrading an HTTPS connection to an insecure HTTP connection, and assumes that the victim will not check user interface indicators that signal that the connection is not secured by TLS. In the attack presented in this paper, the respective interface indicator in a web browser remains unchanged, and it is thus much more difficult for security-aware users to determine whether a connection is being intercepted by an attacker. Similarly, mitigation mechanism against SSLStrip such as HSTS [50] are ineffectual regarding our attack. However, with KCI-based attacks, the user might get suspicious in some cases, because he might be asked – depending on the client software – to confirm the choice of a specific client certificate when he does not expect to do so. However, KCI attackers may be able to alleviate such suspicions easily with appropriate social engineering techniques (after a successful attack, the attacker has full control over the content of the communication, and could use this power to alleviate possible suspicions).

`rsa_fixed_dh`
`dss_fixed_dh`
`rsa_fixed_ecdh`
`ecdsa_fixed_ecdh`

Table 1: List of `ClientCertificateType` vulnerable to KCI attacks

Efforts have been made to work on formal security proofs for the TLS protocol [51, 52, 53]. However, a formal security proof for the complete TLS protocol, including all options and variations, seems to be elusive. Formal analysis are therefore confined to subsets of the protocol, and thus, attacks like ours remain difficult to detect by formal analysis methods alone.

2.2 Key Compromise Impersonation

Key Compromise Impersonation (KCI) was first identified as an attribute of key agreement protocols by Blake-Wilson, Johnson, and Menezes [8]. KCI-vulnerability [8] is a weakness of an authenticated key exchange protocol that allows an attacker who has compromised the secret client credentials of a victim (e.g., client certificate and corresponding secret key) to not just impersonate the compromised client to a server, which is trivial (since the attacker knows the secret client credentials), but also to impersonate any server to the compromised client.

An example of a key agreement protocol that has been proved secure against KCI attacks is Krawczyk's HMQV [54] protocol, a provable secure variant of the heavily analyzed Menezes-Qu-Vanstone (MQV) [55,56] protocol. Clearly, as we show presently, the TLS protocol is not resistant to KCI attacks. A formal definition of resistance against KCI attacks has been provided with

the security analysis of the HMQV [54, 57] protocol in the Canetti-Krawczyk model [58] of authenticated key agreement. For the purpose of this paper, an informal definition is satisfactory.

3 Overview of the attack

To illustrate the problem of KCI with reference to TLS, consider web browsers. TLS clients such as web browsers derive the trust for authenticating and thus securely connecting to servers from pre-installed certification authority (CA) root certificates. An attacker who manages maliciously to install his own CA root certificate in a victim browser's certificate store can trivially launch MitM attacks thereafter – he uses the private key corresponding to the maliciously installed root certificate to issue certificates for those endpoints to which he wants to intercept the communication. Now, web browsers usually allow the user the installation of new certificates – CA root certificates, as well as client certificates which can be used for client authentication. While the user usually receives a warning in one form or another indicating that the newly to be installed CA root certificate can be used to identify and authenticate servers, naturally, no such warnings are displayed when installing a new client certificate. And prima facie, there seems to be no sensible reason to do so: a client certificate is meant to authenticate a client to a server, whereas a CA root certificate's purpose is as indicated by the typical warning: a means to identify and authenticate servers. However, as we will show in detail, by installing client certificates in a victim's web browser – leveraging the KCI vulnerability of certain cipher suites in TLS – the owner of the private key that corresponds to the maliciously installed client certificate can *also authenticate servers* on the Internet to the client. Thus, an attacker who manipulates a client into using his prefabricated client certificate can authenticate his malicious server to the client while pretending to be someone else, and so launch a full-blown MitM attack against communication to possibly sensitive endpoints. The successful attacker would be able to eavesdrop and modify messages from the purportedly securely encrypted and authenticated communication channel.

4 Background: TLS and Fixed (EC) Diffie-Hellman

We first give a succinct overview of the TLS protocol, and then review non-ephemeral (Elliptic Curve) Diffie-Hellman key exchange with fixed (Elliptic Curve) Diffie-Hellman client authentication (henceforth called *fixed (EC)DH handshake* in this paper) more thoroughly. All TLS fixed (EC)DH handshakes are vulnerable to Key Compromise Impersonation (KCI) attacks. For a detailed description of the TLS protocol the reader is referred to the original specifications [1, 2, 3, 5], as well as to the additional cipher suites specifications [9, 59, 60, 61, 62], which extend the TLS protocol. For the purpose of this paper, we present an abstracted view of the TLS protocol, and go into the differences regarding different version of the protocol only where germane to KCI attacks.

4.1 The TLS Protocol

The TLS protocol consists of several subprotocols. All communication is between a client and a server.

- The *TLS record layer* is responsible for the transportation of data. This includes application data as well as the messages of the other subprotocols.
- During the *TLS handshake protocol* client and server negotiate the cipher suite, establish a shared secret, and verify each other's identities.
- At the end of the handshake protocol, the *TLS ChangeCipherSpec protocol* signals the switch to the newly established cryptographic parameters.
- The *TLS alert protocol* is employed to signal error conditions.

At the beginning of a TLS connection, the TLS record layer is initialized with the `TLS_NULL_WITH_NULL_NULL` cipher suite: encryption is the identity operation, and no message authentication code (MAC) is used. The first run of the TLS handshake protocol establishes a TLS session, which defines a set of cryptographic parameters, including the negotiated cipher suite and a shared secret, the *master secret*. TLS sessions are identified by a session identifier, and can be resumed on new connections, using an abbreviated version of the TLS handshake protocol. A run of the TLS handshake protocol that does not resume a previously established session is referred to as *full handshake* in the TLS specification. For the purpose of this exposition we only need to consider full handshakes, because our MitM adversary can always force a client to make a full handshake. During a successful run of the handshake protocol, the record layer is reconfigured to use the newly negotiated cipher suite and key material derived from the newly established master secret for encryption and authentication of the transported data.

4.2 Non-ephemeral (EC)DH vs Ephemeral (EC)DH

The non-ephemeral (EC)DH key exchange in TLS is a variant of the original Diffie-Hellman key exchange [63].

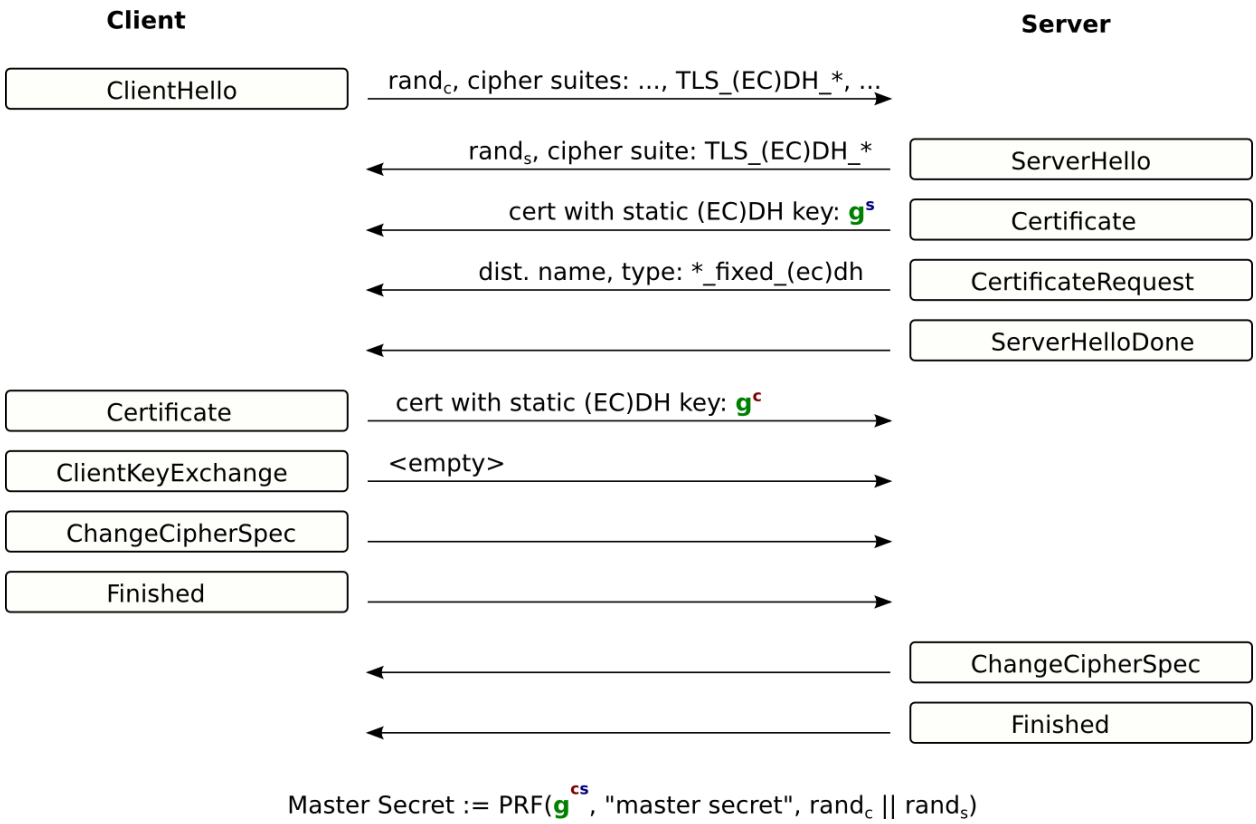

Figure 1: (EC)DH key exchange with fixed (EC)DH client authentication

In the non-ephemeral (EC)DH key exchange, the server uses a static (EC)DH key pair. The server's static (EC)DH public key is embedded in the server's certificate and signed by a certificate authority (CA). The client uses the (EC)DH public key from the server's certificate and combines it with its (EC)DH private key to obtain the shared secret. In contrast, during an ephemeral (EC)DH key exchange in TLS, a client uses the public key in the server's certificate for verification of a signature on an ephemeral (EC)DH public key. Non-ephemeral (EC)DH has less overhead: The server sends the certificate with the static (EC)DH public key, which involves no expensive public key operations, and after it receives the client's (EC)DH public key, it only needs to compute one exponentiation to derive the shared secret. In the ephemeral case, the server, in addition to the (EC)DH exponentiation, has to generate an ephemeral (EC)DH key pair (usually for each handshake), and compute a signature on this ephemeral public key. Whereas an ephemeral (EC)DH key exchange provides Perfect Forward Secrecy (PFS) [7], PFS does not hold for a non-ephemeral (EC)DH key exchanges and past communications can be decrypted if the adversary manages to get hold of the server's static private key. A non-ephemeral key exchange is used in TLS whenever client and server negotiate one of the cipher suites listed in Table 2 during the handshake.

In addition to the cipher suites listed in Table 2, also anonymous non-ephemeral cipher suites exist in TLS. Any anonymous cipher suite is trivially susceptible to a MitM attack, so we do not consider them here.

4.3 Fixed (EC)DH Client Authentication

When the non-ephemeral (EC)DH key exchange is combined with fixed (EC)DH client authentication, both the server and the client have static (EC)DH key pairs: The client's static (EC)DH public key is embedded in the client's certificate. The server authenticates the client by (1) verifying the client's certificate, and (2) by successfully completing the handshake and deriving the same master secret as the client, using the static (EC)DH public key from the client's certificate in the standard (EC)DH key derivation. The fact that the same master secret has been derived is verified by the exchange of the `Finished` messages at the end of TLS handshake, messages sent by default in every TLS handshake. Thus, computational overhead for fixed (EC)DH client authentication is added only by the verification of the client's certificate, which can involve at best just a single signa-

ture verification. A server requests fixed (EC)DH client authentication by demanding one of the certificate types listed in Table 1.

4.4 Fixed (EC)DH Handshake

Remember, in this paper, we call a non-ephemeral (Elliptic Curve) Diffie-Hellman key exchange with fixed (Elliptic Curve) Diffie-Hellman client authentication a *fixed (EC)DH handshake*.

A successful run of the TLS handshake protocol using a fixed (EC)DH handshake is depicted in Figure 1. Variations exist depending on whether a conventional Diffie-Hellman group is used or whether public key computations are done over an elliptic curve. Other variations are due to different signature, MAC, hash, and pseudo random function algorithms used during the handshake. The principles are the same regardless of these variations, and the diagram in Figure 1 and our presentation unifies them.

As illustrated in Figure 1, the client initiates the handshake by sending a `ClientHello` message. This message includes a randomly generated nonce $rand_c$, as well as a list of supported cipher suites. The client indicates its support for non-ephemeral (EC)DH key exchange algorithms by including in this list any non-empty subset of the set of cipher suites from Table 2, that is, any cipher suite that requires the `DH_DSS`, `DH_RSA`, `ECDH_ECDSA`, or `ECDH_RSA` key exchange algorithm, as they are called in the TLS specification.

Next, the server decides on a cipher suite that requires a non-ephemeral (EC)DH key exchange, choosing from the client's list of supported cipher suites. The server includes the chosen cipher suite together with a randomly generated server nonce $rand_s$ in the `ServerHello` message. If a conventional Diffie-Hellman key exchange (`DH_DSS`, or `DH_RSA`) has been chosen, the server includes in the `Certificate` message a certificate with a static Diffie-Hellman public key and the Diffie-Hellman parameters: a prime p, a prime q, and a generator g whose multiplicative order module p is q. Note that such a certificate is, in general, the same as a Digital Signature Standard (DSS) certificate, as both verifying a DSS signature and doing a Diffie-Hellman key exchange requires the same type of cryptographic key material. If an Elliptic Curve Diffie-Hellman key exchange (`ECDH_RSA` or `ECDH_ECDSA`) has been chosen, the server includes in the `Certificate` message a certificate with a static Elliptic Curve Diffie-Hellman public key and the elliptic curve parameters, which also specify, among others, again a generator g. Also in this case, such a certificate can be identical to an Elliptic Curve Digital Signature Standard (ECDSA) certificate. In both the conventional as well as the Elliptic Curve Diffie-Hellman case, the public key is the value of g^s (but computed in different groups), where s is the server's private key.

The server then indicates via the `CertificateRequest` message that it wishes the client to authenticate itself using a certificate that contains a static Diffie-Hellman key by listing `rsa_fixed_dh` or `dss_fixed_dh` (conventional Diffie-Hellman key exchange), or `rsa_fixed_ecdh` or `ecdsa_fixed_ecdh` (Elliptic Curve Diffie-Hellman). The server can request a client certificate signed by specific CAs by providing a list of distinguished names in the `CertificateRequest` message. This also allows the server to describe the desired authorization space. The server marks the end of the first round of communication by sending a `ServerHelloDone` message.

The client starts the second round of communication by sending the client certificate with a static (EC)DH public key in the `Certificate` message. The public key is the value g^c, where c denotes the client's secret key. In the fixed (EC)DH handshake, a `ClientKeyExchange` message with empty content is sent. The static (EC)DH public keys embedded in the certificates are used to calculate the master secret: the client computes $(g^s)^c$, the server computes $(g^c)^s$, and both obtain, because exponentiation in the respective group is commutative, the same master secret by computing $PRF(g^{cs}, \text{"}mastersecret\text{"}, rand_c||rand_s)$, where PRF is a pseudo-random function specified by the chosen cipher suite, and $||$ denotes string concatenation.

The `ChangeCipherSpec` messages indicate that henceforth, the record layer will use encryption and message authentication according to the negotiated cipher suite, using key material derived from the just established master secret. The handshake protocol is completed after both server and client have received and verified the (already encrypted) `Finished` message, which contains the result of $PRF(master\_secret, finished\_label, hash(handshake))$, combining a cryptographic hash of the previous handshake messages with the master secret and a server/client-dependent finished label using again the pseudo-random function specified by the chosen cipher suite.

5 KCI attacks against TLS

In an authenticated key agreement protocol, two parties C and S establish a shared secret and authenticate themselves to each other over an insecure channel. A TLS handshake consisting of a key exchange mechanism and client and server authentication mechanism is an example of an authenticated key agreement protocol. As has already been mentioned in Section 2.2, an authenticated key agreement protocol is resistant to KCI attacks if the

compromise of the private key material of C by mallory M does not allow M to impersonate other, uncorrupted parties to C.

Obviously, if the adversary, mallory M, gets hold of the private key material of C, he can always trivially impersonate C to other parties. KCI deals with the opposing situation, 'reverse impersonation'. Resistance to KCI attacks is a very desirable property of any key agreement protocol.

KCI allows for MitM attacks as follows. An adversary M who is in possession of the secret key material of C can impersonate any party S to whom C would legitimately authenticate itself. Thus, assume C is a client (e.g., a TLS capable web browser) which wants to connect securely to a server S. If an adversary M is in possession of a client certificate $cert_c$ and the corresponding secret key sk_c installed at C, then M would intercept and block the communication from C to S, pose as S, force C to use client authentication using $cert_c$, and launch a KCI-attack to successfully impersonate S to C. The adversary M would then connect to S (not necessarily using client authentication), and forward all traffic from C to S and vice versa, while being able to eavesdrop on and modify plaintext messages at will.

5.1 Prerequisites for KCI attack on TLS

Certain prerequisites need to be met so that an adversary M is able to launch a KCI-based MitM attack against a TLS client C that intends to securely connect to a TLS server S.

5.1.1 Prerequisite 1: Client Support

The client C must support a non-ephemeral (EC)DH cipher suite. C indicates this support by including any of the non-ephemeral (EC)DH cipher suites available in TLS, as listed in Table 2, during the `ClientHello` message. Additionally, the client implementation must support any of the fixed (EC)DH client authentication options implied by the client certificate types listed in Table 1. A client that fully implements any of the common SSL/TLS versions (SSLv3, TLS 1.0, TLS 1.1, TLS 1.2), but not necessarily any of the various TLS extensions, fulfills this prerequisite.

TLS libraries which fulfill this first prerequiste are, e.g., (1) the system TLS library on Mac OS X (Secure Transport), which is used by the Safari Web Browser, (2) the BouncyCastle library, or (3) recent versions of the OpenSSL library[2].

[2]Support for fixed DH client authentication has been very recently added to the OpenSSL 1.0.2 branch.

TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA256
TLS_DH_RSA_WITH_AES_128_CBC_SHA256
TLS_DH_DSS_WITH_AES_256_CBC_SHA256
TLS_DH_RSA_WITH_AES_256_CBC_SHA256
TLS_DH_DSS_WITH_CAMELLIA_128_CBC_SHA
TLS_DH_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_DH_DSS_WITH_CAMELLIA_256_CBC_SHA
TLS_DH_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DH_DSS_WITH_SEED_CBC_SHA
TLS_DH_RSA_WITH_SEED_CBC_SHA
TLS_DH_RSA_WITH_AES_128_GCM_SHA256
TLS_DH_RSA_WITH_AES_256_GCM_SHA384
TLS_DH_DSS_WITH_AES_128_GCM_SHA256
TLS_DH_DSS_WITH_AES_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_NULL_SHA
TLS_ECDH_ECDSA_WITH_RC4_128_SHA
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDH_RSA_WITH_NULL_SHA
TLS_ECDH_RSA_WITH_RC4_128_SHA
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384

Table 2: TLS cipher suites potentially vulnerable to KCI attacks

5.1.2 Prerequisite 2: Server Support

The adversary M must be in possession of a certificate $cert_s$ belonging to server S that contains static (EC)DH values. Such a certificate might be available because S supports a non-ephemeral (EC)DH key exchange. However, it is not necessary that S supports a non-ephemeral cipher suite (in practice, not many server do): a DSS certificate or an ECDSA certificate contains the same cryptographic parameters as a certificate that is used during a non-ephemeral (EC)DH key exchange.

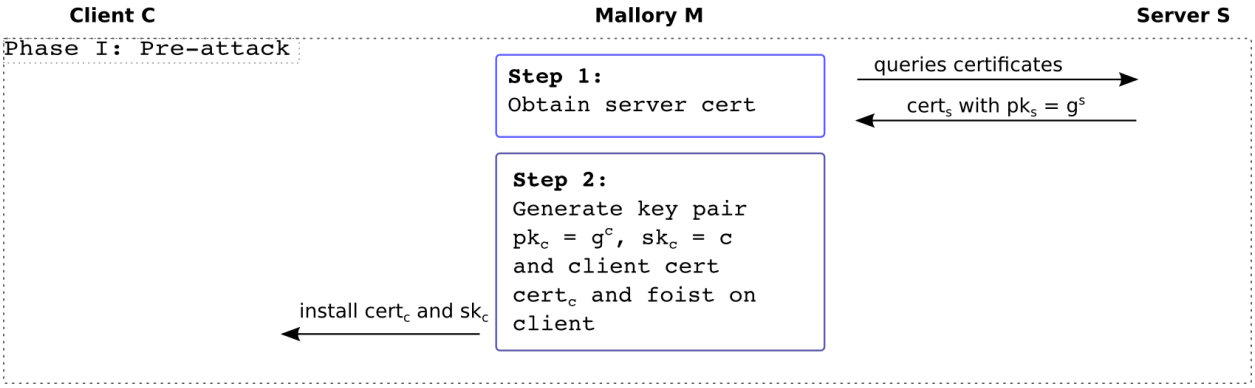

Figure 2: Establishing prerequisites to carry out a KCI attack

ECDSA certificates are structured identical to ECDH certificates, and can be used in the attack instead of ECDH certificates under certain circumstances: If the X509 Key Usage extension is used in the certificate, in principle, the `KeyAgreement` bit must be set so that these certificates can be used for the attack. The `KeyAgreement` bit is sometimes wrongly set in practice (e.g., it is set in the current ECDSA certificate for `facebook.com`). Also, many TLS implementations (such as GnuTLS, CyaSSL, MatrixSSL) do not honor X509 Key Usage Extensions [64]. embed TLS (former: PolarSSL) allows developers to disable the checking of X509 Key Usage extension without due warning. Also, if the X509 Key Usage extension is not used (it is not mandatory), the ECDSA certificate can be used in the attack.

In the case of DSS certificates, a static DH certificate is structured differently than a static DH certicate, and thus, different to the ECDSA case, DSS certificates can be used for the attack only if an implementation error in the client's TLS library allows accepting an DSS certificate when a static DH certificate is expected.

Thus, depending on the specific attack scenario, any server that supports DSS or ECDSA signatures in the handshake (e.g., S supports the `ECDHE_ECDSA` – ephemeral elliptic curve Diffie-Hellman with ECDSA signature – key exchange algorithm) might provide an eligible certificate. Note that if such a certificate exists for any of the reasons stated above, it can be easily queried and obtained by M.

5.1.3 Prerequisite 3: Compromised Client Cert

The adversary M must be in possession of a client certificate $cert_c$ and the corresponding secret key sk_c that must both be installed at the client C. The client certificate $cert_c$ must be 'compatible' with the server certificate $cert_s$ from prerequisite 2: That means, the public key must lie on the same elliptic curve, or use the same conventional Diffie-Hellman parameters (prime p, q, generator g). At least in the elliptic curve case, since most often standardized elliptic curve parameters are used in practice, a suitable client certificate $cert_c$ will be compatible to many other server certificates $cert_s$ for different servers S. Note that although secure mechanisms for generating and installing client certificates exist, in practice it is often quite easy for an adversary M to maliciously install a client certificate $cert_c$ at a client C in a way so that M remains in possession of the corresponding secret key sk_c, as we will discuss in more detail in the next subsection.

In the next two subsections, we describe how a KCI-based MitM attack on TLS might work in detail. Figure 2 and Figure 3 illustrate how an adversary M might proceed to exploit the fact that TLS does not guarantee resistance to KCI.

5.2 Pre-Attack Phase

In a pre-attack phase (see Figure 2) adversary M establishes prerequiste 3 as defined in the previous subsection. The pre-attack phase can be carried out well in advance of the actual attack. Its main purpose is to procure a client certificate $cert_c$ and its corresponding private key sk_c that is or will be installed at the client.

Successfully stealing a compatible pair $(cert_c, sk_c)$ from a client might be a possibility, but seems to be rather unlikely for most situations in practice. However, there are different ways for an adversary M to foist the required pair $(cert_c, sk_c)$ on a client, after M generated the certificate-key pair himself.

- A malicious software vendor, distributer, or an infiltrated malicious actor within a software company might prepare a software product so that it ships with client certificates pre-installed in order to cunningly hide a secret backdoor in the product. The product would contain no obvious flaws, could

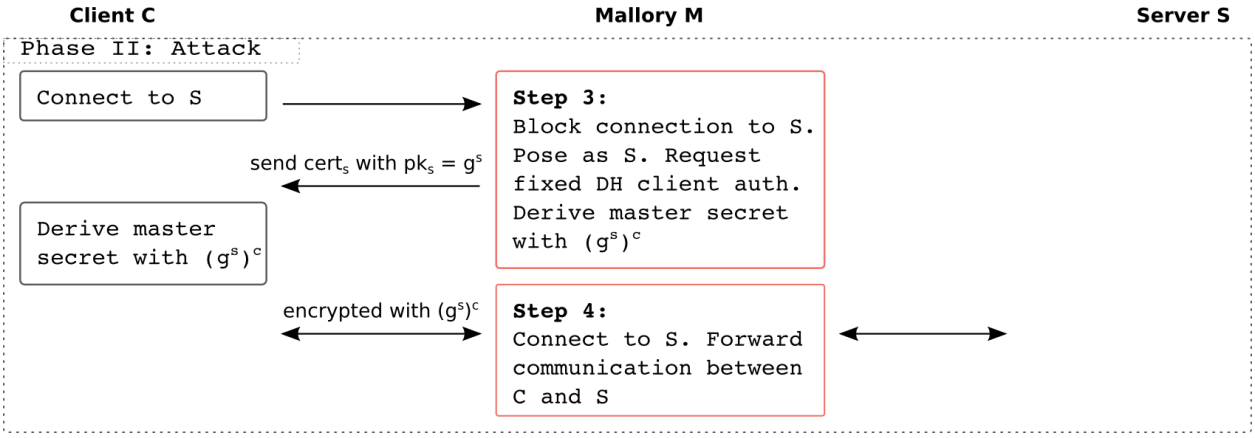

Figure 3: KCI attack used in a TLS MitM scenario

completely conform to the TLS specification, and nothing suspicious might hint at the backdoor, since there might exist completely legitimate reasons to include pre-installed client certificates in the product.

- Although modern browsers – with mechanisms such as the HTML5 `keygen` tag [65] – support the secure generation and installation of client certificates in a way so that the secret key never has to leave the client computer, it is nevertheless common practice among various institutions and companies (as numerous tutorials and how-to guides on the Internet illustrate) that network administrators distribute pre-generated pairs $(cert_c, sk_c)$ to clients. Some operating systems, such as Apple's iOS, have even integrated user interface functionality that allows for easy and smooth installation of pairs $(cert_c, sk_c)$ received via email. A malicious network administrator, or a random attacker using social engineering techniques, can leverage this situation and subvert the security of TLS clients by foisting legitimate access credentials for some arbitrary resource in form of client certificates on unsuspecting users.

- An innocuous looking but malicious Android app might install a pair $(cert_c, sk_c)$ to the system's trust store in a way so that it may get used to authenticate/attack the TLS connections occasioned by another, target application. We validated this scenario with a proof-of-concept implementation on Android 4.4.

This list of examples is most probably very far from exhaustive, and creative minds will be able to devise many more vectors that could enable an adversary to successfully foist pairs $(cert_c, sk_c)$ on clients. Awareness of the danger of KCI attacks seems to be very limited until now, and thus many systems are not designed to treat client certificate installation routines with the care adequate to the danger as propounded in this paper.

5.3 Attack Phase

Once preconditions are met or have been successfully established, an attacker can leverage the KCI weakness of TLS to launch MitM attacks against the connection from a client C to a targeted server S. The attack proceeds as sketched in Figure 3. We will now describe the details.

The client C initiates a TLS connection to server S. Adversary M blocks the traffic on its way to S and poses as S. When the client C starts the TLS handshake protocol by sending a `ClientHello` message, M responds as follows. In the `ServerHello` message, M chooses a cipher suite with a non-ephemeral (EC)DH key exchange (one from Table 2). In the `Certificate` message, M sends S's authentic server certificate $cert_s$, which contains S's public key $sk_s = g^s$. Via the `CertificateRequest` message, M requests fixed (EC)DH client authentication by requesting one of the certificate types listed in Table 1. M chooses the cipher suite and certificate type that match the type of the certificates of $cert_c$ and $cert_s$. By specifying the distinguished name of the CA that was used to sign $cert_c$, M can request that C uses the specific compromised pair $(cert_c, sk_c)$ to authenticate itself.

The client C now finishes the handshake as it would normally finish any fixed (EC)DH handshake. Since the certificate $cert_s$ is authentic, C believes that the connection to S has been correctly authenticated, and any user interface indicators that C might employ will communicate that to the user. Mallory, on the other hand, will slightly deviate from the normal fixed (EC)DH protocol. He does not know the server's secret key sk_s, only S's

public key $pk_s = g^{s'}$. However, since he knows C's secret key $sk_c = c$, he can derive the same master secret MS just by engaging in the exact same computation as C does:

$$MS := PRF((g^s)^c, \text{"master secret"}, rand_c||rand_s)$$

Apart from this slightly altered computation, M finishes the handshake with C like any normal fixed (EC)DH handshake. M has now successfully established a secure TLS connection, impersonating server S. M can now connect to server S and forward any communication from C to S and vice versa, while being able to secretly eavesdrop on and modify messages at will.

6 Evaluation and Impact

6.1 Clients and Client Libraries

We implemented a prototype MitM attack by patching the OpenSSL[3] library with a few lines of code, and using stunnel[4] to simulate a MitM attacker. With this prototype, we tested the most common web browsers (Firefox, Safari, Chrome, Internet Explorer, Opera) on Linux, MS Windows, Android, Mac OS X and iOS. Furthermore, we analyzed manually common TLS libraries, such as OpenSSL, Mozilla NSS, MS SChannel, BouncyCastle, GnuTLS, mbed TLS, MatrixSSL, wolfSSL.

We found that the TLS implementation in the BouncyCastle library fully implements the fixed (EC)DH handshake options. That means, any client program using the BouncyCastle TLS implementation might be vulnerable to our attack. We also found that the system's TLS library on Mac OS X (Secure Transport) to be similarly vulnerable. We then confirmed that our attack works for the Safari web browser on Mac OS X: Versions before Mac OS X 10.5.3 would allow the attack to succeed completely silently, after a client certificate has been maliciously installed in the certificate store. This behavior changed with Mac OS X version 10.5.3[5]. With version 10.5.3 and later, the browser asks the user to confirm the selection of a client certificate when our MitM proxy starts the attack. However, after selecting the malicious certificate for the first time for a particular domain, the attack would proceed completely silently on consecutive connections. We could not reproduce the attack with the most recent versions of Mac OS X (version 10.8., 10.9, 10.10). Further analysis of the source code made public by Apple[6] showed that the necessary code had been (possibly temporarily, according to the comments) disabled via preprocessor macros in these versions. We assisted Apple's security team in identifying, and analyzing vulnerable systems. The currently most commonly used branches of the OpenSSL library (branches 0.9.8, 1.0.0, and 1.0.1) do not support the necessary TLS options (so systems such as Google Android seem to be safe at this time). However, the source code of the OpenSSL library contains 'TODO's in the source code for implementing support for fixed ECDH handshakes. Not much code is missing for fixed ECDH support in OpenSSL (we added basic support with only a few lines of code for our MitM setup). After engaging the OpenSSL developer team during the responsible disclosure process, we found that the newest branch (branch 1.0.2) just recently added support for static DH, but not (yet) for fixed ECDH handshakes. That means, client that use the 1.0.2 branch of OpenSSL might as well be vulnerable to our attack.

Apart from BouncyCastle, the Mac OS X Secure Transport TLS library, and the newest branch of the OpenSSL library, we could not confirm that any other client TLS library that we looked at implements the necessary fixed (EC)DH handshake. However, since TLS use is widespread, and since TLS is not only used to secure web traffic, but all kinds of client-server communications, we estimate that many more systems might in fact be affected than the ones explicitly identified by us. Also, we have to warn that negative results should not necessarily be trusted, since, (1) the weakness is in the TLS protocol, (2) source code is in constant flux, and the missing features might be implemented by TLS library developers anytime in the future, as the example of adding support for fixed DH handshakes to the OpenSSL 1.0.2 library branch has shown.

6.2 TLS Servers

In order to get a feeling for how many servers are vulnerable to our attack, we leveraged Internet-wide scanning of TLS servers on the HTTPS/443 port, using a modified toolset based on the ZMap scanning framework [66, 67] and Paul Graham's masscan [68], as well as current datasets published by the ZMap team[7]. Our analysis results confirmed our initial suspicions (occasioned by the regular scan results conducted and published by Hubert Kario[8] using the `cipherscan` tool [69] on the TLS servers of the Alexa Top 1M most popular server list) that TLS servers implementing (EC)DH key agreement are very rare to find on the public Internet. However, at least around 9.25 % of all reachable HTTPS server in the Alexa Top 1M list offer ECDSA certificates, according to our analysis (Server offering DSS or ECDSA signature-based handshakes are prone to being attacked as well). Some of these ECDSA certificates,

[3] `https://www.openssl.org/`
[4] `https://www.stunnel.org/`
[5] `https://support.apple.com/en-us/ht1679`
[6] `http://opensource.apple.com`
[7] `https://scans.io`
[8] `https://securitypitfalls.wordpress.com/`

such as the ECDSA certificates for `facebook.com`, have the `KeyAgreement` bit set while some do not have the X509 Key Usage extensions at all, and are thus vulnerable to our attack even if client implementations checks for X509 Key Usage extensions are done correctly according to the TLS specification.

Currently, most public TLS servers use RSA signatures in the handshake for performance reason. However, the history of TLS attacks has shown that once a new attack against a specific TLS cipher suite is discovered, TLS server administrators immediately get forced to switch to a different cipher suite. The next such switch could very well be to ECDSA signatures. In addition popular TLS service providers such as Cloudflare [70] use and advocate the use of ECDSA certificates per default. For RSA certificates, unnecessarily setting the `KeyAgreement` bit is quite common by many CAs. Once more CAs are going to offer ECDSA certificates, it can be assumed that this practice will transfer to ECDSA certificate generation, and highly-ranked advise on the Internet supports such behavior [71].

7 Mitigation

In order to mitigate against KCI attacks, *server operators* of security or privacy critical TLS services must

1. Disable non-ephemeral (EC)DH handshakes,
2. Set appropriate X509 Key Usage extension for ECDSA and DSS certificates, and disable specifically the `KeyAgreement` flag.

TLS *client implementors* should immediately

1. Disable non-ephemeral (EC)DH handshake options,
2. Or disable at least support for fixed (EC)DH authentication.

TLS *library implementors* should immediately

1. Check whether they fully consider X509 Key Usage extensions,
2. Mark and properly document non-ephemeral (EC)DH handshakes as deprecated and dangerous.

Moreover, the common practice of insecure client certificate handling should end. Only mechanisms that ensure that a client's private key does not leave its machine, such as the HTML keygen tag, should be employed. System administrators at universities and industries should update their procedure to distribute client certificates accordingly.

TLS security best-practice recommendations such as the recent RFC 7525 [18] should be updated accordingly, and warn against KCI-vulnerable cipher suites. Moreover, the next version of TLS, v1.3, should definitely not include KCI-vulnerable cipher suites.

8 Conclusion

We have identified a novel way to attack TLS implementations. Although the attack has been known to at least some theoretical cryptographers who research in the field of authenticated key exchange protocols, knowledge has not yet spread to a wider audience of security researchers and practitioners. In this paper we explained KCI attacks against TLS and argued that they are practical in real-world scenarios. We identified and confirmed that widespread TLS clients, such as Apple's Safari Web browser on Mac OS X, are vulnerable. The immediate impact is not as serious as, for example, the one from the recent Logjam attack, because support for the necessary options in TLS clients and servers (both is necessary) is currently not as widespread as a malicious attacker would hope for. However, without adequate measures, this situation could change anytime in the future: Recently, OpenSSL developers have just added support for the vulnerable fixed DH handshake to the newest branch (1.0.2) of the library, and they seemed to be on track for also adding support for the fixed ECDH handshake option. Also ECDSA certificates will very probably become more popular in the future. We hope that our disclosure of the vulnerability prohibits any possible large-scale exploitation of the attack in the future, and stops malevolent actors form using it to cunningly hide backdoors in security or privacy critical systems.

Acknowledgment

We are highly indebted to Matthew Green for assisting us in responsibly disclosing the vulnerability, and providing valuable feedback. Furthermore, we would like to thank the OpenSSL developer team for their pre-public-disclosure discussion, feedback, and corrections. "Thank you"s also go to Lucas Telefont for helping us with our tests on Mac OS X; to the anonymous reviewers for their valuable comments and accurate reading of the paper; to the principal author's master thesis advisor Christopher Kruegel for general inspiration, and getting him started doing research and thinking about challenging and hard problems. Last but not least, we thank Thomas Grechenig for running an awesome company, and Franz Schönbauer for pushing cryptography within this company and providing the vision without this work might not have happened.

References

[1] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," RFC 2246 (Proposed Standard), Internet Engineering Task Force, Jan. 1999, obsoleted by

RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507. [Online]. Available: http://www.ietf.org/rfc/rfc2246.txt

[2] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," RFC 4346 (Proposed Standard), Internet Engineering Task Force, Apr. 2006, obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176, 7465, 7507. [Online]. Available: http://www.ietf.org/rfc/rfc4346.txt

[3] ——, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878, 6176, 7465, 7507. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[4] K. E. Hickman, "The SSL protocol," 1995. [Online]. Available: http://www-archive.mozilla.org/projects/security/pki/nss/ssl/draft02.html

[5] A. Freier, P. Karlton, and P. Kocher, "The Secure Sockets Layer (SSL) Protocol Version 3.0," RFC 6101 (Historic), Internet Engineering Task Force, Aug. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6101.txt

[6] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Beguelin, and P. Zimmermann, "Imperfect forward secrecy: How diffie-hellman fails in practice," https://weakdh.org/, 2015.

[7] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Des. Codes Cryptography*, vol. 2, no. 2, pp. 107–125, Jun. 1992.

[8] S. Blake-Wilson, D. Johnson, and A. Menezes, "Key agreement protocols and their security analysis," in *Proceedings of the 6th IMA International Conference on Cryptography and Coding*. London, UK, UK: Springer-Verlag, 1997, pp. 30–45.

[9] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)," RFC 4492 (Informational), Internet Engineering Task Force, May 2006, updated by RFCs 5246, 7027. [Online]. Available: http://www.ietf.org/rfc/rfc4492.txt

[10] A. K. Lenstra, X. Wang, and B. de Weger, "Colliding X.509 certificates." *IACR Cryptology ePrint Archive*, vol. 2005, p. 67, 2005.

[11] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. Weger, "Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate," in *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 55–69.

[12] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt, "On the security of RC4 in TLS," in *Proceedings of the 22Nd USENIX Conference on Security*, ser. SEC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 305–320.

[13] M. Vanhoef and F. Piessens, "All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/vanhoef

[14] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 protocol," in *Proceedings of the 2Nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, ser. WOEC'96. Berkeley, CA, USA: USENIX Association, 1996, pp. 4–4.

[15] S. Turner and T. Polk, "Prohibiting Secure Sockets Layer (SSL) Version 2.0," RFC 6176 (Proposed Standard), Internet Engineering Task Force, Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6176.txt

[16] A. Diquet and A. Grattafiori, "sslyze." [Online]. Available: https://github.com/iSECPartners/sslyze

[17] I. Ventura-Whiting and J. Applebaum, "sslscan." [Online]. Available: https://github.com/DinoTools/sslscan

[18] Y. Sheffer, R. Holz, and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," RFC 7525 (Best Current Practice), Internet Engineering Task Force, May 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7525.txt

[19] "IETF TLS WG." [Online]. Available: https://tools.ietf.org/wg/tls/

[20] E. Rescorla, "The transport layer security (TLS) protocol version 1.3." [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-05

[21] M. Ray, "Renegotiating TLS," 2009.

[22] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub, "Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 98–113.

[23] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs #1," in *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '98. London, UK, UK: Springer-Verlag, 1998, pp. 1–12.

[24] S. Vaudenay, "Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS ..." in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, ser. EUROCRYPT '02. London, UK, UK: Springer-Verlag, 2002, pp. 534–546.

[25] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux, "Password interception in a SSL/TLS channel," in *Advances in Cryptology - CRYPTO 2003*, ser. Lecture Notes in Computer Science, D. Boneh, Ed. Springer Berlin Heidelberg, 2003, vol. 2729, pp. 583–599.

[26] V. Klima, O. Pokorny, and T. Rosa, "Attacking rsa-based sessions in SSL/TLS," in *Cryptographic Hardware and Embedded Systems - CHES 2003*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, vol. 2779, pp. 426–440.

[27] D. Brumley and D. Boneh, "Remote timing attacks are practical," in *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, ser. SSYM'03. Berkeley, CA, USA: USENIX Association, 2003, pp. 1–1.

[28] G. V. Bard, "The vulnerability of SSL to chosen plaintext attack," *IACR Cryptology ePrint Archive*, vol. 2004, p. 111, 2004. [Online]. Available: http://eprint.iacr.org/2004/111

[29] B. Moeller, "Security of CBC ciphersuites in SSL/TLS. problems and countermeasures," https://www.openssl.org/~bodo/tls-cbc.txt, 2004.

[30] G. V. Bard, "A challenging but feasible blockwise-adaptive chosen-plaintext attack on SSL," in *Secrypt 2006, Proceedings of the International Conference on Security and Cryptography*. INSTICC Press, 2006, pp. 7–10.

[31] M. Marlinspike, "New tricks for defeating SSL in practice," Black Hat USA, 2009.

[32] ——, "More tricks for defeating SSL in practice," Black Hat USA, 2009.

[33] T. Duong and J. Rizzo, "Here come the $\oplus$ ninjas," 2011.

[34] K. Paterson, T. Ristenpart, and T. Shrimpton, "Tag size does matter: Attacks and proofs for the TLS record protocol," in *Advances in Cryptology – ASIACRYPT 2011*, ser. Lecture Notes in Computer Science, D. Lee and X. Wang, Eds. Springer Berlin Heidelberg, 2011, vol. 7073, pp. 372–389.

[35] B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in *Computer Security – ESORICS 2011*, ser. Lecture Notes in Computer Science, V. Atluri and C. Diaz, Eds. Springer Berlin Heidelberg, 2011, vol. 6879, pp. 355–371.

[36] T. Duong and J. Rizzo, "The crime attack," Presentation at Ekoparty Security Conference, 2012.

[37] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel, "A cross-protocol attack on the TLS protocol," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 62–72.

[38] K. G. Paterson and N. J. AlFardan, "Plaintext-recovery attacks against datagram TLS," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[39] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, "Revisiting SSL/TLS implementations: New bleichenbacher side channels and attacks," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 733–748.

[40] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, "A messy state of the union: Taming the composite state machines of TLS," in *IEEE Symposium on Security & Privacy 2015 (Oakland'15)*. IEEE, 2015.

[41] C. Meyer, "20 Years of SSL/TLS Research : An analysis of the Internet's Security Foundation," Ph.D. dissertation, Ruhr-University Bochum, February 2014. [Online]. Available: http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/MeyerChristopher/diss.pdf

[42] C. Meyer and J. Schwenk, "Lessons learned from previous SSL/TLS attacks - a brief chronology of attacks and weaknesses," Cryptology ePrint Archive, Report 2013/049, 2013, http://eprint.iacr.org/.

[43] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged SSL certificates in the wild," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 83–97.

[44] "Diginotar reports security incident," https://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_diginotar_reports_security_incident.aspx, 2011.

[45] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving ssh-style host authentication with multi-path probing," in *USENIX 2008 Annual Technical Conference*, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 321–334. [Online]. Available: http://dl.acm.org/citation.cfm?id=1404014.1404041

[46] M. Marlinspike, "Digital trust and the future of authenticity," Presentation at Blackhat conference, 2011.

[47] ——, "Trust assertions for certificate keys," http://tack.io/draft.html, 2013.

[48] P. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," RFC 6698 (Proposed Standard), Internet Engineering Task Force, Aug. 2012, updated by RFC 7218. [Online]. Available: http://www.ietf.org/rfc/rfc6698.txt

[49] B. Schneier, "Man-in-the-middle attacks on lenovo computers," https://www.schneier.com/blog/archives/2015/02/man-in-the-midd_7.html, 2015.

[50] J. Hodges, C. Jackson, and A. Barth, "HTTP Strict Transport Security (HSTS)," RFC 6797 (Proposed Standard), Internet Engineering Task Force, Nov. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6797.txt

[51] H. Krawczyk, K. Paterson, and H. Wee, "On the security of the TLS protocol: A systematic analysis," in *Advances in Cryptology – CRYPTO 2013*, ser. Lecture Notes in Computer Science, R. Canetti and J. Garay, Eds. Springer Berlin Heidelberg, 2013, vol. 8042, pp. 429–448.

[52] P. Morrissey, N. Smart, and B. Warinschi, "A modular security analysis of the TLS handshake protocol," in *Advances in Cryptology - ASIACRYPT 2008*, ser. Lecture Notes in Computer Science, J. Pieprzyk, Ed. Springer Berlin Heidelberg, 2008, vol. 5350, pp. 55–73.

[53] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *Advances in Cryptology – CRYPTO 2012*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds. Springer Berlin Heidelberg, 2012, vol. 7417, pp. 273–293. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32009-5_17

[54] H. Krawczyk, "HMQV: A high-performance secure diffie-hellman protocol," in *Advances in Cryptology – CRYPTO 2005*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Springer Berlin Heidelberg, 2005, pp. 546–566.

[55] A. Menezes, M. Qu, and S. Vanstone, "Some new key agreement protocols providing mutual implicit authentication," in *Selected Areas in Cryptography (SAC 95)*, 1995.

[56] L. Law, A. Menezes, M. Qu, J. Solinas, and S. +Vanstone, "An efficient protocol for authenticated key agreement," *Designs, Codes and Cryptography*, vol. 28, no. 2, pp. 119–134, 2003.

[57] H. Krawczyk, "HMQV: A high-performance secure diffie-hellman protocol," Cryptology ePrint Archive, Report 2005/176, 2005, http://eprint.iacr.org/.

[58] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, ser. EUROCRYPT '01. London, UK, UK: Springer-Verlag, 2001, pp. 453–474.

[59] E. Rescorla, "TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)," RFC 5289 (Informational), Internet Engineering Task Force, Aug. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5289.txt

[60] S. Moriai, A. Kato, and M. Kanda, "Addition of Camellia Cipher Suites to Transport Layer Security (TLS)," RFC 4132 (Proposed Standard), Internet Engineering Task Force, Jul. 2005, obsoleted by RFC 5932. [Online]. Available: http://www.ietf.org/rfc/rfc4132.txt

[61] H. Lee, J. Yoon, and J. Lee, "Addition of SEED Cipher Suites to Transport Layer Security (TLS)," RFC 4162 (Proposed Standard), Internet Engineering Task Force, Aug. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4162.txt

[62] J. Salowey, A. Choudhury, and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS," RFC 5288 (Proposed Standard), Internet Engineering Task Force, Aug. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5288.txt

[63] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, 1976.

[64] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 114–129. [Online]. Available: http://dx.doi.org/10.1109/SP.2014.15

[65] "HTML5. a vocabulary and associated APIs for HTML and XHTML," 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-html5-20141028/

[66] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast internet-wide scanning and its security applications," in *Proceedings of the 22Nd USENIX Conference on Security*, ser. SEC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 605–620. [Online]. Available: http://dl.acm.org/citation.cfm?id=2534766.2534818

[67] D. Adrian, Z. Durumeric, G. Singh, and J. A. Halderman, "Zippier zmap: Internet-wide scanning at 10 gbps," in *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. San Diego, CA: USENIX Association, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/woot14/workshop-program/presentation/adrian

[68] R. Graham, "Masscan: the entire internet in 3 minutes," http://blog.erratasec.com/2013/09/masscan-entire-internet-in-3-minutes.html, 2013.

[69] J. Vehent, H. Kario, and et. al., "cipherscan: A very simple way to find out which ssl ciphersuites are supported by a target." [Online]. Available: https://github.com/jvehent/cipherscan

[70] N. Sullivan, "ECDSA: The digital signature algorithm of a better internet," https://blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/, 2013.

[71] T. Porin, "Answer to: Which key usages are required by each key exchange method?" [Online]. Available: http://security.stackexchange.com/a/24107