Prying Open Pandora’s Box: KCI Attacks against TLS

Clemens Hlauschek,
Markus Gruber,
Florian Fankhauser,
Christian Schanes

USENIX WOOT 2015
Outline of this Talk

- Authenticated Key Agreement and KCI
- TLS is vulnerable to KCI
- KCI and TLS in practice
- Conclusion and Mitigation
Key Compromise Impersonation (KCI)

Weakness of Authenticated Key Agreement protocol
Key Compromise Impersonation (KCI)

Weakness of Authenticated Key Agreement protocol

Authenticated Key Agreement
- 2 parties exchange messages
- Over an adversarial network
- To derive a shared secret (session key)
Weakness of **Authenticated Key Agreement** protocol

- Compromise of long-term secret allows to trivially impersonate the compromised party
- KCI – reverse situation: Impersonate an uncompromised party to the compromised party
- KCI allows for MitM attacks
Key Compromise Impersonation (KCI)

Weakness of Authenticated Key Agreement protocol

- Compromise of long-term secret allows to trivially impersonate the compromised party
- KCI – reverse situation: Impersonate an uncompromised party to the compromised party
- KCI allows for MitM attacks
Key Compromise Impersonation (KCI)

Weakness of Authenticated Key Agreement protocol

- Compromise of long-term secret allows to trivially impersonate the compromised party
- KCI – reverse situation: Impersonate an uncompromised party to the compromised party
- KCI allows for MitM attacks
Non-ephemeral Diffie-Hellman key exchange with fixed Diffie-Hellman client authentication

- $\mathbb{Z}_p$ as well as EC
- In all TLS versions
- Client indicates support in ClientHello message
- Server requests fixed_(ec)dh authentication
- Session key is derived from static DH values:

  client: $PRF((g^s)^c, rand_c||rand_s)$
  server: $PRF((g^c)^s, rand_c||rand_s)$
Non-ephemeral Diffie-Hellman key exchange with fixed Diffie-Hellman client authentication

- \( \mathbb{Z}_p \) as well as EC
- In all TLS versions
- Client indicates support in ClientHello message
- Server requests fixed_(ec)dh authentication
- Session key is derived from static DH values:
  
  - client:  \( PRF((g^s)^c, rand_c || rand_s) \)
  - server:  \( PRF((g^c)^s, rand_c || rand_s) \)
Non-ephemeral Diffie-Hellman key exchange with fixed Diffie-Hellman client authentication

- $\mathbb{Z}_p$ as well as EC
- In all TLS versions
- Client indicates support in ClientHello message
- Server requests fixed_(ec)dh authentication
- Session key is derived from static DH values:
  - client: $PRF((g^s)^c, rand_c || rand_s)$
  - server: $PRF((g^c)^s, rand_c || rand_s)$
Non-ephemeral Diffie-Hellman key exchange with fixed Diffie-Hellman client authentication

- $\mathbb{Z}_p$ as well as EC
- In all TLS versions
- Client indicates support in ClientHello message
- Server requests fixed_(ec)dh authentication
- Session key is derived from static DH values:
  - client: $PRF((g^s)^c, rand_c || rand_s)$
  - server: $PRF((g^c)^s, rand_c || rand_s)$
Non-ephemeral Diffie-Hellman key exchange with fixed Diffie-Hellman client authentication

- $\mathbb{Z}_p$ as well as EC
- In all TLS versions
- Client indicates support in ClientHello message
- Server requests fixed_(ec)dh authentication
- Session key is derived from static DH values:
  - client: $PRF((g^s)^c, rand_c||rand_s)$
  - server: $PRF((g^c)^s, rand_c||rand_s)$
TLS protocol is vulnerable to KCI

Man-in-the-Middle attack against TLS using KCI

- Block connection to server
- Send server cert
- Request fixed (EC)DH
- Request compromised cert via Distinguished Name in CertRequest
- Both attacker and client do the same session key computation:
  \[ PRF((g^s)^c, rand_c||rand_s) \]
- Connect to server
TLS protocol is vulnerable to KCI

Man-in-the-Middle attack against TLS using KCI

- Block connection to server
- Send server cert
- Request fixed (EC)DH
- Request compromised cert via Distinguished Name in CertRequest
- Both attacker and client do the same session key computation: \[ PRF((g^s)^c, rand_c || rand_s) \]
- Connect to server
TLS protocol is vulnerable to KCI

Man-in-the-Middle attack against TLS using KCI

- Block connection to server
- Send server cert
- Request fixed (EC)DH
- Request compromised cert via Distinguished Name in CertRequest
- Both attacker and client do the same session key computation: $PRF((g^s)^c, rand_c||rand_s)$
- Connect to server
TLS protocol is vulnerable to KCI

Man-in-the-Middle attack against TLS using KCI

- Block connection to server
- Send server cert
- Request fixed (EC)DH
- Request compromised cert via Distinguished Name in CertRequest
- Both attacker and client do the same session key computation:
  \[ PRF((g^s)^c, \text{rand}_c||\text{rand}_s) \]
- Connect to server
TLS protocol is vulnerable to KCI

Man-in-the-Middle attack against TLS using KCI

- Block connection to server
- Send server cert
- Request fixed (EC)DH
- Request compromised cert via Distinguished Name in CertRequest
- Both attacker and client do the same session key computation: $PRF((g^s)_c, rand_c||rand_s)$
- Connect to server
Prerequisites KCI attacks against TLS

1. **Victim client support**: must implement non-ephemeral Diffie Hellman with fixed client authentication handshake
   - rsa_fixed_dh
   - dss_fixed_dh
   - rsa_fixed_ecdh
   - ecdsa_fixed_ecdh

2. **Victim server support**: must have matching certificate

3. **Compromised client certificate’s secret**:
   - Stolen private key
   - Client cert foisted on victim (various vectors)
Prying Open Pandora’s Box: KCI Attacks against TLS

Foisting client cert on victim: Social engineering

- Secure ways for generating client certs exist
- Common practice: send pre-generated client certs with secret key to user
- Insecure OS mechanisms to install client certs
- Attacker / malicious admin coax victim to install client certificate for network X, then use it to exploit connections to all vulnerable servers

### HTML `<keygen>` Tag

**Example**

A form with a keygen field:

```
<form action="demo_keygen.asp" method="get">
  Username: <input type="text" name="usr_name">
  Encryption: <keygen name="security">
  <input type="submit">
</form>
```

**Definition and Usage**

The `<keygen>` tag specifies a key-pair generator field used for forms. When the form is submitted, the private key is stored locally, and the public key is sent to the server.
Foisting client cert on victim: Social engineering

- Secure ways for generating client certs exist
- Common practice: send pre-generated client certs with secret key to user
- Insecure OS mechanisms to install client certs
- Attacker / malicious admin coax victim to install client certificate for network X, then use it to exploit connections to all vulnerable servers
Foisting client cert on victim: Social engineering

- Secure ways for generating client certs exist
- Common practice: send pre-generated client certs with secret key to user
- Insecure OS mechanisms to install client certs
- Attacker / malicious admin coax victim to install client certificate for network X, then use it to exploit connections to all vulnerable servers
Foisting client cert on victim: Social engineering

- Secure ways for generating client certs exist
- Common practice: send pre-generated client certs with secret key to user
- Insecure OS mechanisms to install client certs
- Attacker / malicious admin coax victim to install client certificate for network X, then use it to exploit connections to all vulnerable servers
Secure ways for generating client certs exist

Common practice: send pre-generated client certs with secret key to user

Insecure OS mechanisms to install client certs

Attacker / malicious admin coax victim to install client certificate for network X, then use it to exploit connections to all vulnerable servers
Foisting client cert on victim: Social engineering

- Secure ways for generating client certs exist
- Common practice: send pre-generated client certs with secret key to user
- Insecure OS mechanisms to install client certs
- Attacker / malicious admin coax victim to install client certificate for network X, then use it to exploit connections to all vulnerable servers
For example (hypothetically): Abusing the trustStore on Android devices

- A user installs a malicious, but benign looking app
- Malicious app installs client certificate in system trustStore
- Targeted app makes TLS connection
- MitM forces targeted app to use client authentication, using the previously installed cert
- User confirms client authentication
A malicious vendor or distributor might install a backdoor in form of a client certificate

- **Superfish-MitM**: Inject own CA certificate
- **KCI-Backdoor**:
  - Implementation fully spec-conform
  - Server certs do not change
Victim server support: Matching Certificate

Server must either

- Support a non-ephemeral (EC)DH handshake
- Have an ECDSA certificate ( < 10% )
  - ECDH and ECDSA cert same structure
  - If X509 KeyUsage extension is used
    - KeyAgreement Bit must be set
    - But client may not check KeyUsage extension
- KeyUsage extension not mandatory
Victim client support

Vulnerable client software

- Programs using BouncyCastle might be vulnerable
- Apple SecureTransport on older versions of Mac OS X (Safari)
- OpenSSL
  - Recently added support (1.0.2 branch) for fixed DH ($\mathbb{Z}_p$) client authentication
  - TODOs in the source code for fixed ECDH client authentication
- RSA Bsafe(?): support for non-ephemeral ECDH (according to API documentation)
Conclusion and Mitigation

- Clients should disable KCI-vulnerable cipher suites
- ECDSA server certificates should not set KeyAgreement bit in X509 KeyUsage extension
- Industry best-practice guides (e.g., RFC 7572) should warn against KCI-vulnerable cipher suites
- Secure generation of client certificates (private key does not leave user’s computer) should become common practice

Although we managed to attack prestigious targets (Safari – Facebook), both client and server support are rather rare, currently. Hopefully, this work prevents the issue from ever becoming more widespread:

- OpenSSL only very recently added support for fixed DH client authentication
- ECDSA certificates are probably becoming more widespread in the future