Integrating an almost undiscoverable Trapdoor in a SSL/TLS Browser

Daniel Muster, 8048 CH-Zurich
July 2018, Version 2.2
(The first version appeared in 2014)

It's not the encryption
that's cryptography.
It's the random number generator!
Inspired by Tom Stoppard:
It's not the voting
that's democracy.
It's the counting!

Random number generators may be insufficient by mistake or by intention. That's why the quality of the key establishment protocol should not rely on the security of a single random number generator. It is demonstrated on SSL/TLS as a showcase how easily and efficiently a trapdoor may be implemented.

SSL/TLS has a great conceptual weakness. The quality of the random number generator in the client (browser) is crucial for confidentiality, integrity and authenticity of the SSL/TLS connection. This weakness makes SSL/TLS vulnerable to mistakes and to the integration of an almost undiscoverable trapdoor. "Almost undiscoverable" means that it could not be recognised from outside in a reasonable time. From outside is defined as without any code inspection or without any analysis of the code behaviour on the client operating system. (The browser is treated as a black box). In this paper an example is presented how such a trapdoor can be efficiently integrated.

A random number generator may be insufficient by mistake or by intention (trapdoor). However the reason may be for this insufficiency makes it obvious to ask for a new requirement in cryptographic protocols: "The security of a key agreement protocol should not rely on the quality of a single random number generator. "This requirement, that is not established in the security community and in literature, would have an impact to many implemented cryptographic protocols.

SSL/TLS an Overview

Here in brief and much simplified how the SSL/TLS handshake works (for additional information see the IETF standards):

1. Agreement on a secret key K.
2. From K the session keys for encryption and integrity checks are evaluated.
3. Exchange of control messages to verify if the server and the client have derived the same session keys.
4. Exchange of data between the server and the client.

For this purpose only phase 1 and 2 are of importance. The agreement can be done by a RSA encryption with the server public key from its certificate or with a Diffie-Hellman key

exchange and a server signature. For the key establishment it is completely irrelevant if the server will authenticate the client during phase 1 or not!

Principle of the Trapdoor

What is encrypted by RSA (M) or is the Diffie-Hellman private key (C) during the SSL/TLS handshake looks to everyone as generated “randomly”. But it is done in an almost predictable way for an external third man (named Carl) who listens to the SSL/TLS connection.

To realize that random information sent in plaintext during SSL/TLS phase 1 are taken as an input of a function F to generate the bits of M or C. The function F and its input parameters have to be kept secret between the SW developer of the browser and Carl.

RSA Encryption

The parameters for the key establishment in SSL/TLS are:

RA_C: 28 Byte randomly generated by the client and sent to the server.

RA_S: 28 Byte randomly generated by the server and sent to the client.

M: 46 Byte randomly generated by the client. M is encrypted with the server public key S of its certificate. => $E_S(M)$ is sent to the server.

From M, RA_C and RA_S the session keys are evaluated.

Integration of a Trapdoor in the RSA key exchange

M is not generated randomly but is the value of a function F. The function F and its input parameters have to be kept secret between the SW developer and the external third man (here named Carl) who wants to listen to the communication, to intercept it or to introduce his own data. Carl has only access to the communication wire where the SSL/TL communication takes place.

The input parameters of F are e.g.:

- The random numbers RA_C and RA_S (sent in plaintext)
- P, a randomly generated bit string of secret length.
- A secret string of defined length
- Session individual parameters as time (in the client-hello message), the server certificate, chosen ciphersuite, version number, session ID and so on. The whole value of these parameters or parts of it.

How it works

Except of P Carl knows each input of F mentioned before. He evaluates for each value P a candidate M’ by the function F and afterwards the session key candidates. Then he checks if the session key candidates and the SSL/TLS control messages are matched.

Diffie-Hellman and server signature

The parameters for the key establishment in SSL/TLS are:

RA_C: 28 Byte randomly generated by the client and sent to the server.

RA_S: 28 Byte randomly generated by the server and sent to the client.

S: Diffie-Hellman secret key randomly generated by the server. The server makes the Diffie-Hellman key operation => $y_S = g^S \bmod p$. This is sent to the client.

C: Diffie-Hellman secret key randomly generated by the client. The client makes the Diffie-Hellman key operation => $y_C = g^C \bmod p$. This is sent to the server.

From g^{CS} mod p, RA_C and RA_S the session keys are evaluated.

Integration of a Trapdoor in Diffie-Hellman exchange

C is not generated randomly but is the output of a function F. The function F and its input parameters have to be kept secret between the SW developer and Carl. The input parameters of F are e.g.:

- The random numbers $y_S = g^S$ mod p, RA_C and RA_S (sent in plaintext)
- P, a randomly generated bit string of secret length.
- A secret string of defined length
- The signature of the server (a random number sent in plaintext too!)
- Session individual parameters as time (in the client-hello message), the server certificate, chosen Ciphersuite, version number, session ID and so on.

How it works

Except of P Carl knows each input of F (mentioned above) to calculate C. For each value P the function F creates a candidate "Can" for C. Then he compares g^C mod p with g^{Can} mod p. If both are equal, he creates the SSL/TLS session keys with RA_C , RA_S and g^{CS} mod p . To be sure he also checks if the session keys and the SSL/TLS control messages are matched.

Further Explanations

As much session individual information as possible should be taken as input parameters for the function F to have as much entropy as possible to generate M or C. This helps to pretend that M and C are generated randomly and prevents collisions that could unmask the trapdoor. It would be optimal if the entropy of the function input were greater than M or C when generated randomly. (Entropy of input > than the bit length of M or C).

The longer P is chosen the longer it takes to get the right keys. But it diminishes the probability of a collision and therefore of unmasking the trapdoor.

Conditions for success

Important conditions for success are:

- Much entropy as possible to generate C or M to prevent any statistical collisions. The length of P is a trade-off between fast decryption and hiding the trapdoor. The best way to detect any anomalism is to make the server act in the same way (same random number and so on). Under these circumstances there will be a collision with probability greater 0.5 after about 2^T trials, $T \sim (|RA_C| + |P|)/2$.
  $|P|$ = the bit length of P, $|RA_C|$ = the bit length of RA_C
  This is true for RSA encryption, but with Diffie-Hellman key exchange there is even more entropy because of the server signature.
  After the detection of a statistically unexpected collision it is still a long way however to prove that there is a trapdoor in the browser!
  Remark: After Heartbleed was published almost every server has been reconfigured to the Diffie-Hellman key exchange. This has increased the entropy.
- F should preserve the entropy. The entropy of the input is almost equal the entropy of the output.

- The function F has to mask the (statistical) dependence of C and M from the SSL session input parameters. It may consist of hash functions and/or encryption operations. Even after B outputs for M or C one should not be able to recognize any difference from an output of B blocks for M or C by a complete random source. B should depend on the input entropy X. About ~ $2^{0.5X}$ trials must be made being able to detect a collision with probability 0.5.
- F has to be obfuscated in the source code, for obfuscation techniques and principles see [CoNa].
- Session individual parameters, that are handled by a not trapdoored SSL/TLS Browser anyway, should be taken as input parameters to evaluate C or M. So the discovery of the trapdoor by the appearance of any strange values or operations could be decreased.
- RA_C should be generated randomly with high quality to pretend that M or C generated the same way too.
- Even if the function F were discovered it would be another story to prove that there exist a shared secret (the function F and its input parameters) between the software developer and Carl.

Conclusions

SSL/TLS has an enormous weakness in design because the quality of the random generator in the browser is crucial for the SSL/TLS security. *That's why the security of the key establishment protocol must not depend on the quality of a single random generator.* (So a Diffie-Hellman key exchange with signature authentication might not meet this requirement.) This must be taken into account when selecting a key establishment protocol. Diffie-Hellmann key exchange with public key encryption authentication might be a good alternative, e.g. IKE chapter 5.2.

"Not being able to find an evidence for a trapdoor is not an evidence that there isn't any trapdoor." That's why the critical security parameters should be under control of the customer nowadays. Interfaces that allow the customer to analyze critical security parameters should be built in as default.

References

[BoMa] Colin Boyd, Anish Mathuria, Protocols for Authentication and Key Establishment, Springer, 2003

[CoNa] Christian Collberg, Jasvir Nagra, Surreptitious Software, Addison-Wesley, 2009

[Mus] Muster Daniel, Digitale Unterschriften und PKI, 3. Auflage, 2006

[Res] Eric Rescorla, SSL and TLS, Addison-Wesley, 2001

[Sch] Bruce Schneier, Applied Cryptography, Wiley, 1994

IKE The Internet Key Exchange (IKE), RFC 2409

SSL 3.0 The Secure Sockets Layer (SSL) Protocol Version 3.0, RFC 6101

TLS 1.0 The TLS Protocol, RFC 2246

TLS 1.1 The Transport Layer Security (TLS) Protocol Version 1.1, RFC 4346

TLS 1.2 The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246