Cryptography and Security - Assignment 1

Mateusz Jachniak

22.11.2020

Contents

1 Assignment 1 - malicious devices

1.1 Content

We have discussed examples on how to create a trapdoor in the output of a device that allegedly follows some secure cryptographic protocol. Your task is to examine a communication protocol of your choice – the only conditions is that it must be used in practice. You have to show how to create such a trapdoor in the following situation:

- you have the full access to the protocol implementation on the attack device,
- the modification should be undetectable when the messages of the protocol are inspected by an observer having no access to attacker's secrets,
- the attacker (and nobody else) should be able to recover the plaintext messages sent with this protocol. In particular, a person who has a read access to the code contained in the device should not be able to do it.

1.2 Task

Provide a report containing:

- a concise and clear description of the attacked protocol,
- a description of the attack focusing on the crucial details.

2 Solution - Kleptographic attack on SSH

2.1 SSH - Secure Shell

The Secure Shell (SSH)[1] Protocol is a protocol for secure remote login and other secure network services over an insecure network. The SSH protocol consists of three major components:

- **The Transport Layer Protocol** provides server authentication, confidentiality, and integrity with perfect forward secrecy. This protocol is mainly responsible for algorithm negotiation, session key exchange, and authenticating the server to the client.
- **The User Authentication Protocol** authenticates the client to the server. All client-driven authentications are made with this protocol.
- **The Connection Protocol** multiplexes the encrypted tunnel into several logical channels. The details of these protocols are described in separate documents. This protocol multiplexes multiple logical communications channels over a single SSH connection.

SSH provides a secure channel over an unsecured network by using a client–server architecture[2], connecting an SSH client application with an SSH server. SSH is generally used to access Unix-like operating systems, but it can also be used on Microsoft Windows. Windows 10 uses OpenSSH as its default SSH client and SSH server.

2.2 Transport Layer Protocol

Setting up encryption and identity keys are done in the transport layer protocol so the messages should be briefly explained exchanged in this protocol There are three steps in the transport layer protocol:

- Identification string exchange - in this step, the client and the server send each other a string, usually containing SSH version and software version.
- Algorithm negotiation - here each side sends a message containing lists of supported algorithms in the order of preference,
- Key exchange - In this step, the client and the server execute Diffie- Hellman with parameters p,g and q, to establish a session key/shared secret. This is done/achieved in the following steps:
  1. The client generates $x \leftarrow Z_q^*$, computes e=g^x and sends e.
  2. The server generates $y \leftarrow Z_q^*$, computes M and sends it to the client. M = (Public server's key, $f = g^y$(mod p), server-signed hash H (described below)). Its value is computed in the following way:
     - $f = g^y$(mod p)
     - $K = e^y$(mod p)
     - H = hashFunction(K,f,e,client's message, server message, server's public key, server's and client's list of supported algorithms (described at Algorithm Negotiation))
  3. Client computes $K = f^x$ mod p and H and client can verify signature s.

Both sides share a secret key K and derive encryption and identity keys in a deterministic way from this key.

2.3 Kleptographic attack

In the SSH protocol random numbers that are private to the client are not sent on the network. The only example of this type is the private key used in the Diffie-Hellman key exchange.
To simplify the scheme, let's concentrate only on Diffie-Hellman key exchange. Let's choose y_1 randomly (first time) and the crypto-system outputs $f_1 = g_1^y$ and stores y_1 in memory (for the next protocol execution). In the next iteration let $y_2 = hashFuntion(e^{y_1})$ and $f_2 = g^{y_2}$. Now, no observer can compute the session key, without knowing the value of y, but the attacker can easily do it, by computing $hashFuntion(y_1^x)$. And it works for every next step of protocol.
What did we want to achieve? If the attacker put his public key in the device could get easily session key K. Also user should no be aware of it, because hashing function should provide fair distribution of private keys (so it could also go wit h public keys). Also, messages are just normal messages. The attacker is the only non-participating party able to successfully eavesdrop on the encrypted communication.

3 Conclusions

Due to the scope of kleptographic attacks in many cryptographic algorithms, we cannot fully trust them. An attacker could publish an open-source library or system which from the first look is innocent, but could be

susceptible to such attacks. But my example is not the only way of performing the attack. There are many different kleptographic attacks on protocols such as SSL and SSH. We just have to hope that is that some additional precautions can help prevent kleptographic attacks.

References

[1] Secure shell (ssh) authentication protocol architecture. https://tools.ietf.org/html/rfc4251.

[2] Secure shell (ssh) authentication protocol. https://tools.ietf.org/html/rfc4252.