Net::SSH connections are subject to plaintext recovery due to lack of CTR mode

Description

It is my understanding that due to the current Ruby OpenSSL bindings, only the following cipher modes are supported in Net::SSH:

Net::SSH supports the following ciphers:

aes128-cbc
3des-cbc
blowfish-cbc
cast128-cbc
aes192-cbc
eaes256-cbc
rijndael-...@lysator.liu.se
idea-cbc
none

I am not talking about the ciphers (aes, des, idea, etc.) here. A quick clarification for those who need it: AES, 3DES etc. are block ciphers, this means that they take a block of cleartext and a key and produce a block of ciphertext (and vice versa), but when you're dealing with streams of information, you have to figure out how to join these blocks together, and there are security tradeoffs in how you do it. So CBC is "cipher block chaining" mode, and CTR is "counter" mode. You will notice that the only block chaining modes supported are only CBC.

If you review the following: http://www.kb.cert.org/vuls/id/958563 you will see that this attack can potentially allow an attacker to recover up to 32 bits of plaintext from an arbitrary block of ciphertext from a connection secured using the SSH protocol in the standard configuration.

In order to mitigate this vulnerability SSH can be setup to use CTR mode rather CBC mode. According to CPNI Vulnerability Advisory SSH:

The most straightforward solution is to use CTR mode instead of CBC mode, since this renders SSH resistant to the attack. An RFC already exists to standardise counter mode for use in SSH (RFC 4344).

Due to the limited number of cipher modes available, any system wishing to do Net::SSH (eg. capistrano operations) that has picked specific ciphers for local policy reasons that do not include CBC ciphers will result in a mysterious problem due to lack of agreed cipher modes, the only solution is to downgrade the available ciphers presented to those of what Ruby has available. This has come up a number of times on the Capistrano list (e.g. http://www.mail-archive.com/capistrano@googlegroups.com/msg05641.html).

It is my understanding that the fix requires tweaking of Ruby's OpenSSL bindings to provide these newer cipher modes. In a sufficiently modern TLS implementation, I'd argue that it's simply going to be more and more incompatible with clients and servers as stricter requirements become standard.

History

#1 - 02/19/2011 10:50 AM - MartinBosslet (Martin Bosslet)

The Cipher class uses the OpenSSL EVP API, but if you look in the file evp.h in all available OpenSSL versions (including >= 1.0.0) you will find this:

```
#if 0
const EVP_CIPHER *EVP_aes_128_ctr(void);
#endif
```

As soon as this is supported by OpenSSL itself, it will also be available in Ruby's Cipher support.
Regards,
Martin

#2 - 06/26/2011 05:53 PM - naruse (Yui NARUSE)
- Status changed from Open to Assigned
- Assignee set to nahi (Hiroshi Nakamura)

#3 - 06/26/2011 06:40 PM - nahi (Hiroshi Nakamura)
- Target version set to 2.0.0

#4 - 06/10/2012 07:06 AM - MartinBosslet (Martin Bosslet)
I think we can close this? As of OpenSSL 1.0.1, OpenSSL::Cipher supports CTR modes.

#5 - 11/29/2012 10:28 PM - nahi (Hiroshi Nakamura)
- Category set to ext
- Status changed from Assigned to Third Party's Issue

Indeed. Closing this as TPI. Added CTR test at r37994 for making sure we can use CTR.