SSH and Debian Weak Keys

The purpose of this activity is to gain hands-on experience with the SSH protocol and to explore how the Debian weak keys bug could be exploited to gain access to a system.

Examining SSH Traffic

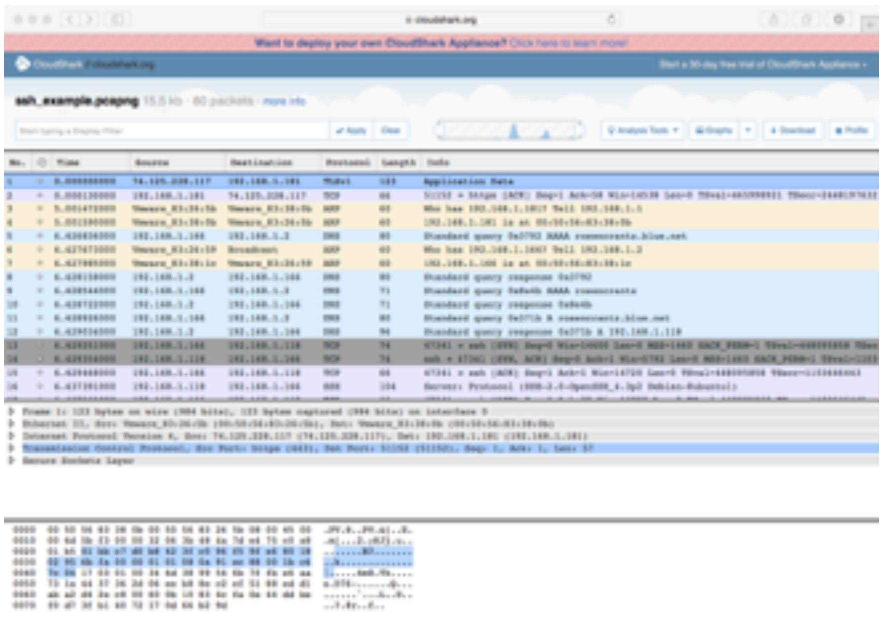
Perhaps we can find a user's public key by sniffing their network traffic. An ssh session for a user on `rosencrantz.blue.net` was captured and saved. We will use CloudShark to look at the traffic and see if we can find the user's public key.

a. Download the packet capture file:

   http://www.csee.umbc.edu/~cmarron/pub/ssh_example.pcapng

b. Browse to the CloudShark website www.cloudshark.org and click the Upload button.

c. Select the file `ssh_example.pcapng` and upload it to CloudShark. You should see the following list of packets, which includes packets sent between **`rosencrantz.blue.net`** (192.168.1.118) and the Ubuntu workstation the user was working on (192.168.1.166):



```
No. Time         Source           Destination      Protocol Length Info
1   0.000000000  74.125.228.117   192.168.1.181    TLSv1    123    Application Data
2   0.000130000  192.168.1.181    74.125.228.117   TCP      66     51152 → https [ACK] Seq=1 Ack=58 Win=[illegible]
3   5.001472000  Vmware_83:26:5b  Vmware_83:38:5b  ARP      60     Who has 192.168.1.181? Tell 192.168.1.1
4   5.001590000  Vmware_83:38:5b  Vmware_83:26:5b  ARP      60     192.168.1.181 is at 00:50:56:83:38:5b
5   6.426836000  192.168.1.166    192.168.1.2      DNS      80     Standard query 0x0792 AAAA rosencrantz.blue.net
6   6.427473000  Vmware_83:26:19  Broadcast        ARP      60     Who has 192.168.1.166? Tell 192.168.1.2
7   6.427985000  Vmware_83:38:1c  Vmware_83:26:19  ARP      60     192.168.1.166 is at 00:50:56:83:38:1c
8   6.428158000  192.168.1.2      192.168.1.166    DNS      80     Standard query response 0x0792
9   6.428544000  192.168.1.166    192.168.1.2      DNS      71     Standard query 0x8e4b AAAA rosencrantz
10  6.428722000  192.168.1.2      192.168.1.166    DNS      71     Standard query response 0x8e4b
11  6.428926000  192.168.1.166    192.168.1.2      DNS      80     Standard query 0x271b A rosencrantz.blue.net
12  6.429034000  192.168.1.2      192.168.1.166    DNS      96     Standard query response 0x271b A 192.168.1.118
13  6.429251000  192.168.1.166    192.168.1.118    TCP      74     47341 → ssh [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=[illegible]
14  6.429356000  192.168.1.118    192.168.1.166    TCP      74     ssh → 47341 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=[illegible]
15  6.429448000  192.168.1.166    192.168.1.118    TCP      66     47341 → ssh [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=[illegible] TSecr=[illegible]
16  6.437391000  192.168.1.118    192.168.1.166    SSH      104    Server: Protocol (SSH-2.0-OpenSSH_4.3p2 Debian-8ubuntu1)

▷ Frame 1: 123 bytes on wire (984 bits), 123 bytes captured (984 bits) on interface 0
▷ Ethernet II, Src: Vmware_83:26:5b (00:50:56:83:26:5b), Dst: Vmware_83:38:5b (00:50:56:83:38:5b)
▷ Internet Protocol Version 4, Src: 74.125.228.117 (74.125.228.117), Dst: 192.168.1.181 (192.168.1.181)
▷ Transmission Control Protocol, Src Port: https (443), Dst Port: 51152 (51152), Seq: 1, Ack: 1, Len: 57
▷ Secure Sockets Layer

[illegible]
```

d. Scroll down until you see the beginning of the SSH protocol.

e. You can examine the contents of a packet by clicking on it in the upper window. A list of the protocol layers will appear in the lower window; click on the "▷" to display details of a particular protocol level.

1. Find the TCP packet that initiates the SSH connection.

   What is the packet number (first column)?

2. Find the SSH Protocol messages.

   What version of OpenSSH is the server running?

   What version of OpenSSH is the client running?

3. Find the SSH key exchange init packets.

   What is the client's preferred key exchange algorithm (first in the list)?

   What is the client's preferred encryption algorithm?

   What is the server's preferred key exchange algorithm?

   What is the server's preferred encryption algorithm?

4. Recall, the client and server must agree on key exchange and encryption algorithms. They will use the first entry in the *client's* list that is also in the server's list.

   What key exchange algorithm will the server and client agree to use?

   What encryption algorithm will the server and client agree to use?

5. Packets 23 – 27 complete the Transport Protocol key exchange.

   What cryptographic algorithm is implemented by these packets?

   Examine the contents of these packets. You may also need to review the "SSH Background" notes to answer the following questions.

   At any point in this exchange, does the client transmit its public key?

6. Examine the SSH Protocol contents one of the subsequent SSH packets, e.g. packet #29.

   Can you read the contents of the packet? Why or why not?

   Recall that the SSH Transport Protocol sets-up a connection *before* the Authentication Protocol is invoked.

   Should be expect to find a user's public key by sniffing the SSH session? Explain.

What good is a user's key?

A key is *weakly generated* if it is computationally feasible to derive the private key from the public key. In the context of SSH, what good is the private key to an attacker?

1. Suppose an attacker is in possession of Alice's private SSH key for the server `big.secret.net`. The private key is in the file `id_rsa_alice` and Alice's user name is `alice`. Review the man page entry for the ssh command and explain how the "`-i identity_file`" and "`-o PasswordAuthentication=no`" options can be used to login to `big.secret.net` as Alice. Write down the *exact* command that you would use.

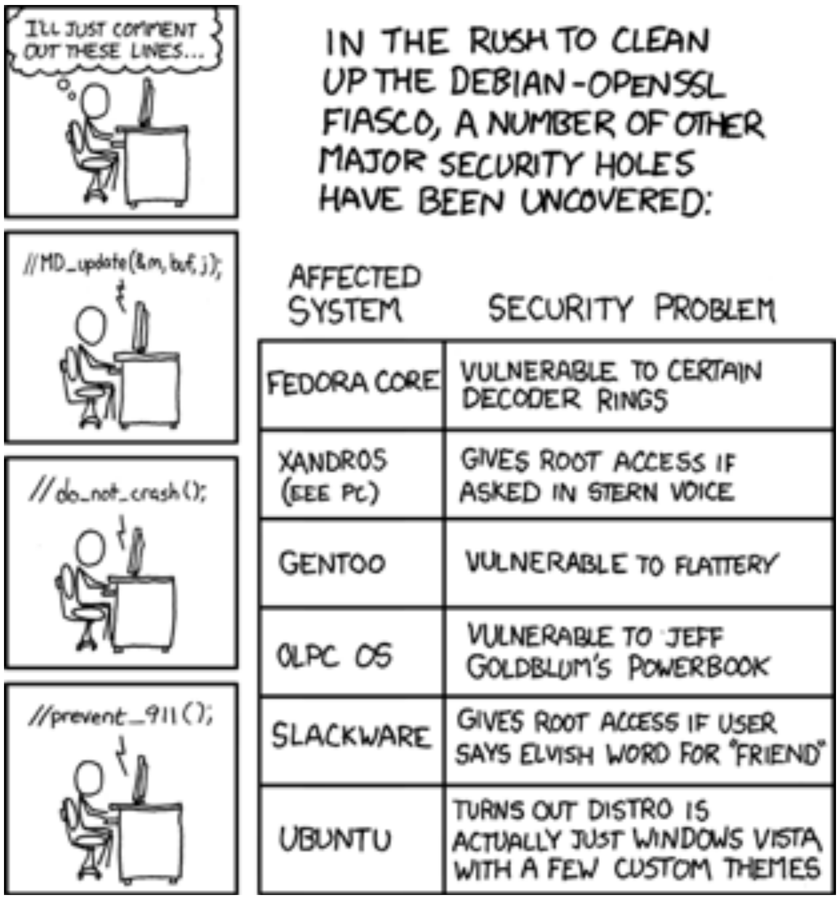
2. For your answer to question #1, what assumption did you have to make about the SSH server configuration on `big.secret.net`?

So, if we can see a user's public key and determine it is weak, then we can derive the private key and use it to login to the user's account – provided the server is configured favorably. Unfortunately, we have already seen that the client's public key is not transmitted as part of the SSH Transport Protocol.

The Debian "Fiasco"

A problem with Debian OpenSSL was discovered in May 2008, and it was determined that the error been in place since September 2006 when a code change eliminated the entropy source for the OpenSLL random number generator, rendering the output predictable.

Reduced entropy in the random number generator results in weak cryptographic keys. Affected protocols included SSH, OpenVPN, DNSSEC, and key material in X.509 certificates for SSL/TLS. The original Debian Security Advisory describes the problem and steps users should take to secure their systems (http://www.debian.org/security/2008/dsa-1571)


I'LL JUST COMMENT OUT THESE LINES...
//MD_update(&m, buf, j);
//do_not_crash();
//prevent_911();
IN THE RUSH TO CLEAN UP THE DEBIAN-OPENSSL FIASCO, A NUMBER OF OTHER MAJOR SECURITY HOLES HAVE BEEN UNCOVERED:
AFFECTED SYSTEM
SECURITY PROBLEM
FEDORA CORE
VULNERABLE TO CERTAIN DECODER RINGS
XANDROS (EEE PC)
GIVES ROOT ACCESS IF ASKED IN STERN VOICE
GENTOO
VULNERABLE TO FLATTERY
OLPC OS
VULNERABLE TO JEFF GOLDBLUM'S POWERBOOK
SLACKWARE
GIVES ROOT ACCESS IF USER SAYS ELVISH WORD FOR "FRIEND"
UBUNTU
TURNS OUT DISTRO IS ACTUALLY JUST WINDOWS VISTA WITH A FEW CUSTOM THEMES

The change was prompted by a report that Valgrind (a memory check tool) was flagging use of uninitialized memory by any program that used the OpenSSL RNG. There was some discussion between two Debian maintainers as to how to fix the bug, but in the end they decided to comment-out some code to stop the Valgrind messages; unfortunately, the code that they removed introduced random data into a buffer used for random number generation. As a result, there was very little entropy (randomness) in the RNG output. For SSH in particular, there were only 2^{15} = 32,767 different keys for each algorithm (RSA or DSA) and size (1024, 2048, etc.).

Demonstration — Exploiting Weak SSH Keys

It is easy to reconstruct the Debian SSH error — it is not necessary that the target server be running the old version of ssh, just that the user keys were generated with the old version. In this demonstration, we will

- Review how public keys are used for user authentication in SSH.
- Examine a public-available "blacklist" of weak SSH keys and discuss how it may be used to attack a vulnerable user.
- Demonstrate a Python script to automate the attack; examine the use of the pexpect module.
- Demonstrate a tool published by Debian to help users and administrators identify vulnerable keys on their systems.

Who was responsible?

It may appear that the developers were acting irresponsibly, but they had in fact discussed their plans, though perhaps not entirely clearly, on the opnessl-dev mailing list (http://marc.info/?t=114651088900003&r=1&w=2), and no one told them it was a bad idea. It's also interesting to read the discussion between the maintainers, which is recorded by the Debian bug-tracking system (http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=363516).

Once the problem was discovered, Debian quickly fixed it and released a tool that could determine if a given key had been generated using the weak RNG (dowkd.pl.gz, referenced in the Debian Security Advisory). In addition there was debate over who was really to blame for the error:

- Should the Debian maintainers have known better than to comment-out crypto code they didn't fully understand?
- Should the OpenSSL developers have been following the openssl-dev discussion?

There is one very good point that can be made: had the Debian developers reported the bug and proposed change to the OpenSSL developers directly, the error would not have been made. The article by Ben Laurie also brings up the fact that Debian made fixed code available five days before they issued the security advisory. Experienced hackers could easily have noticed the change to the code and used that information to exploit vulnerable systems before the fixed code could be installed.

References

Debian Security Advisory, 13 May 2008, http://www.debian.org/security/2008/dsa-1571

Bruce Schneier, *Random Number Bug in Debian Linux*, https://www.schneier.com/blog/archives/2008/05/random_number_b.html

Steinar Gunderson, *Some maths* , http://plog.sesse.net/blog/tech/2008-05-14-17-21_some_maths.html

Ben Laurie, *Debian and OpenSSL: The Aftermath*, http://www.links.org/?p=328#comment-177420

Openssl-dev discussion, http://marc.info/?t=114651088900003&r=1&w=2

XKCD, *security_holes.png*, http://imgs.xkcd.com/comics/security_holes.png