Secure Shell (SSH)

Remote Access

- idea is to run commands on another computer
  - use a keyboard and monitor you can hold to control another machine
  - e.g., server machines, headless machines, routers, remote machines
- original designs had no security
  - plaintext transmission

Telnet

- designed in 1969, formalized in 1970
- defines a protocol for two machines to communicate
  - typically uses tcp/23
  - sends keystrokes from client to server
  - sends printed characters from server to client
  - includes telnet control information
    - interrupt process, abort output, etc.

r-commands

- designed in 1982
- suite of remote access commands
  - rlogin: remote login
  - rcp: remote copy
  - rexec: remote excute

Both Telnet and rlogin had no security

Both Telnet and rlogin had no security
Usernames and passwords were sent plaintext

Both Telnet and rlogin had no security
Usernames and passwords were sent plaintext
Then in 1995 Tatu Ylönen witnessed a
password-sniffing attack in his university network

Both Telnet and rlogin had no security
Usernames and passwords were sent plaintext
Then in 1995 Tatu Ylönen witnessed a
password-sniffing attack in his university network
i.e., eavesdropper who gathered login
and passwords by looking at the traffic.

Both Telnet and rlogin had no security
Usernames and passwords were sent plaintext
Then in 1995 Tatu Ylönen witnessed a
password-sniffing attack in his university network
i.e., eavesdropper who gathered login
and passwords by looking at the traffic.
So he read about cryptography and invented ssh.

SSH

- secure shell
  - widely used popular tool
- allows remote login
  - run commands on other computer
  - does it securely
- supports secure authentication
  - password, public key
- supports X forwarding for graphics
- other programs can build on the secure channel
  - e.g., sshfs, git over ssh, scp, socket proxy

SSH and TLS

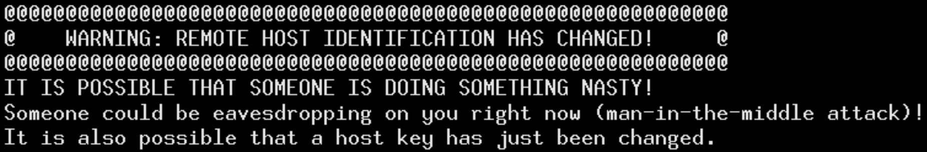
- while both offer strong security guarantees they are quite different
- TLS concerns only transport
  - application layer implemented by the program separately
- SSH has authentication protocols for login
  - password based
  - signature based
- SSH has connection protocols
  - implementing remote access
  - X forwarding
  - port forwarding

SSH-1: login sequence

- client contacts server ("host")
  - provides SSH protocol version and implementation version
- server replies
  - its public RSA "host-key" (1024 bits)
    - this is permanent, stored in a config file
  - random RSA "server-key" (768 bits)
    - changed hourly
    - never saved to a file
  - eight random bytes
  - list of supported ciphers

SSH-1: login sequence (con't)

- both client and server compute session identifier
    - md5 hash of host key, server key, and random bytes
- client checks a local cache of host keys
    - if host is not in the cache, show the key to user and ask to add
        - connection may abort here
    - else if the host is in the cache and the host key matches all good
    - else warn the user that the key has changed
        - connection may abort here
- client randomly generates a session key
    - based on supported ciphers offered by server
    - encrypts with server key and host key
        - why doubly encrypt?
    - sends to the server
- client and server now have a secure channel



```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
```

SSH-1: login sequence (con't)

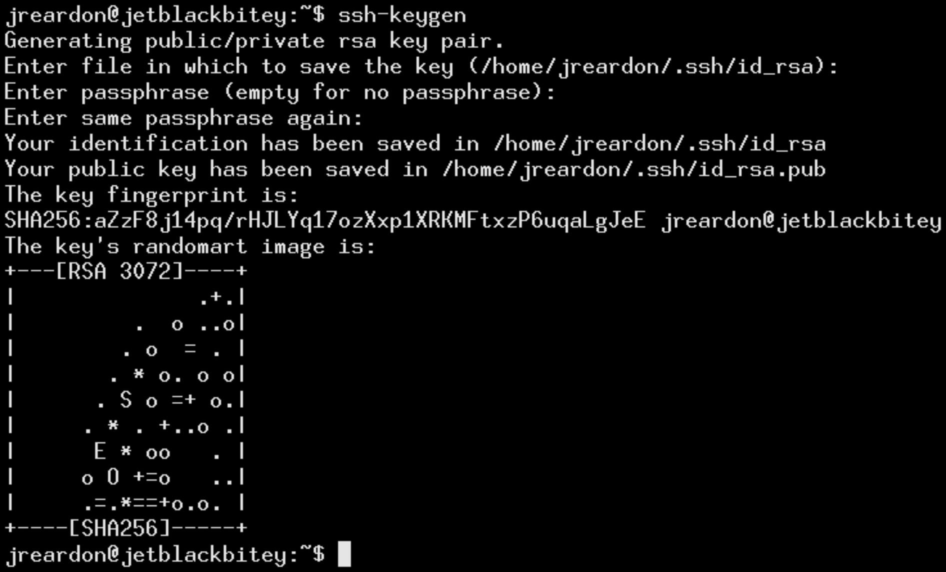
- client now has to authenticate to the server
  - supports Kerberos, password, public key
  - best practice is public key
- Kerberos
  - user gets a ticket to login to system
  - ssh server is just like a printer or other service
- password authentication
  - client types their password into the terminal
  - sends to server over encrypted channel
  - server checks that it is the password for that user on this machine

SSH-1: login sequence (con't)

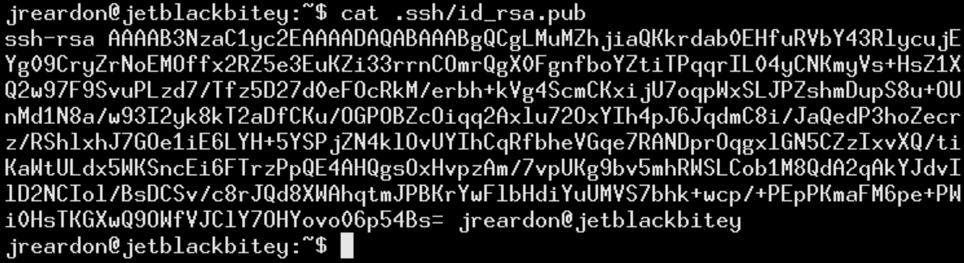
- public key authentication
  - client sends server the public key they want to use
  - server checks if that key is authorized for that user
    - /home/user/.ssh/authorized_keys
    - if not listed in that file, reject
  - server challenges client
    - server generates a random 256-bit string
    - encrypts with client's public key and sends it
  - client answers challenge
    - client decrypts it with private key
    - combines the challenge with session identifier
    - hashes the result with MD5 and sends it
  - what **purposes are** served by adding the session identifier?
    - what are two things that can go wrong?

Why is public key more secure than password based?

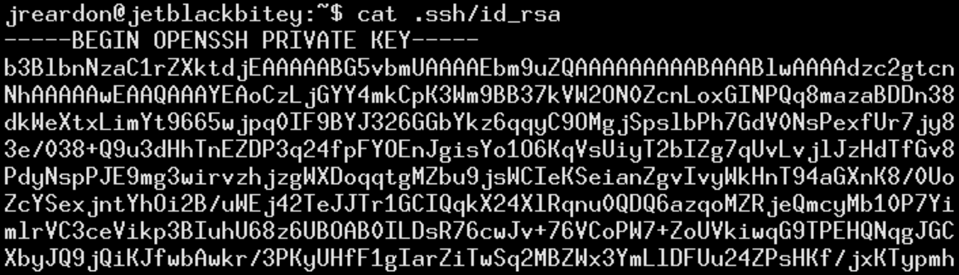
How is the client’s public key stored?



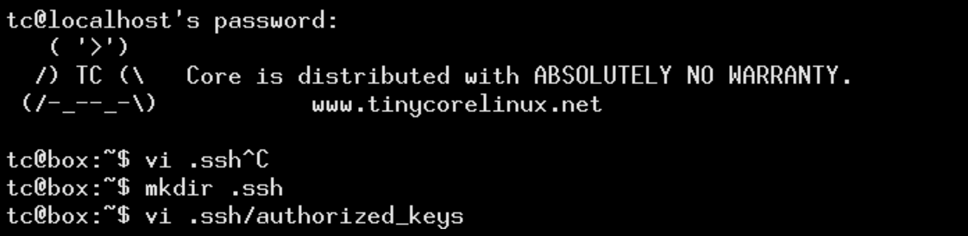
```
jreardon@jetblackbitey:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jreardon/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jreardon/.ssh/id_rsa
Your public key has been saved in /home/jreardon/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:aZzF8j14pq/rHJLYq17ozXxp1XRKMFtxzP6uqaLgJeE jreardon@jetblackbitey
The key's randomart image is:
+---[RSA 3072]----+
|              .+.|
|         .  o ..o|
|        . o  = . |
|       . * o. o o|
|      . S o =+ o.|
|     . * . +..o .|
|      E * oo   . |
|     o O +=o   ..|
|     .=.*==+o.o. |
+----[SHA256]-----+
jreardon@jetblackbitey:~$ █
```



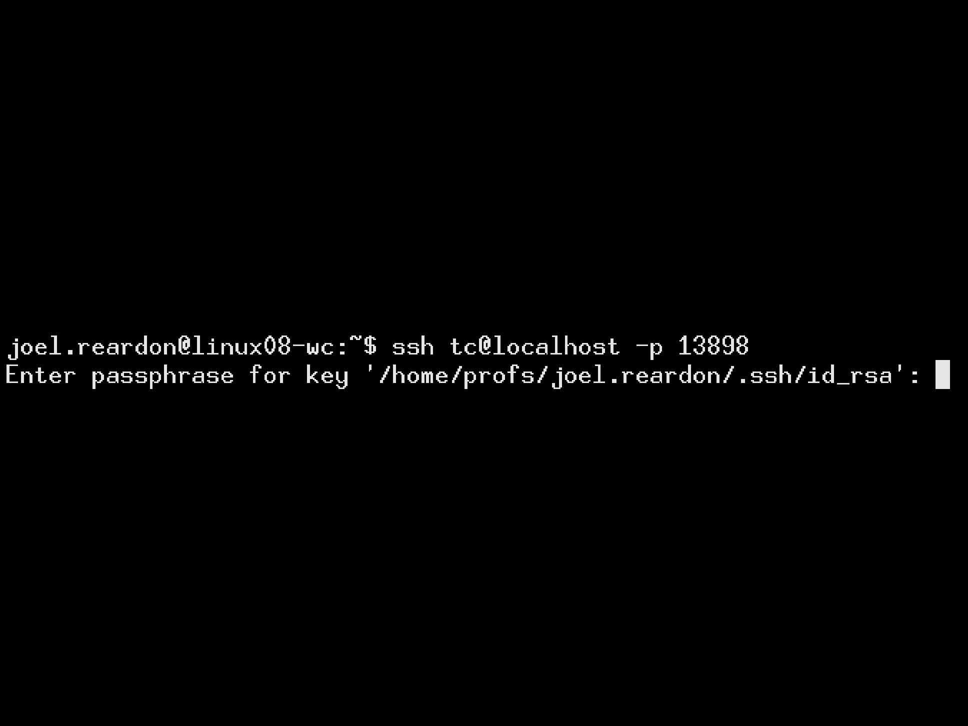
```
jreardon@jetblackbitey:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCgLMuMZhjiaQKkrdab0EHfuRVbY43RlycujE
Yg09CryZrNoEM0ffx2RZ5e3EuKZi33rrnC0mrQgX0FgnfboYZtiTPqqrIL04yCNKmyVs+HsZ1X
Q2w97F9SvuPLzd7/Tfz5D27d0eF0cRkM/erbh+kVg4ScmCKxijU7oqpWxSLJPZshmDupS8u+0U
nMd1N8a/w93I2yk8kT2aDfCKu/0GP0BZc0iqq2Axlu720xYIh4pJ6JqdmC8i/JaQedP3hoZecr
z/RShlxhJ7G0e1iE6LYH+5YSPjZN4kl0vUYIhCqRfbheVGqe7RANDpr0qgxlGN5CZzIxvXQ/ti
KaWtULdx5WKSncEi6FTrzPpQE4AHQgs0xHvpzAm/7vpUKg9bv5mhRWSLCob1M8QdA2qAkYJdvI
lD2NCIol/BsDCSv/c8rJQd8XWAhqtmJPBKrYwFlbHdiYuUMVS7bhk+wcp/+PEpPKmaFM6pe+PW
i0HsTKGXwQ90WfVJClY70HYovo06p54Bs= jreardon@jetblackbitey
jreardon@jetblackbitey:~$
```



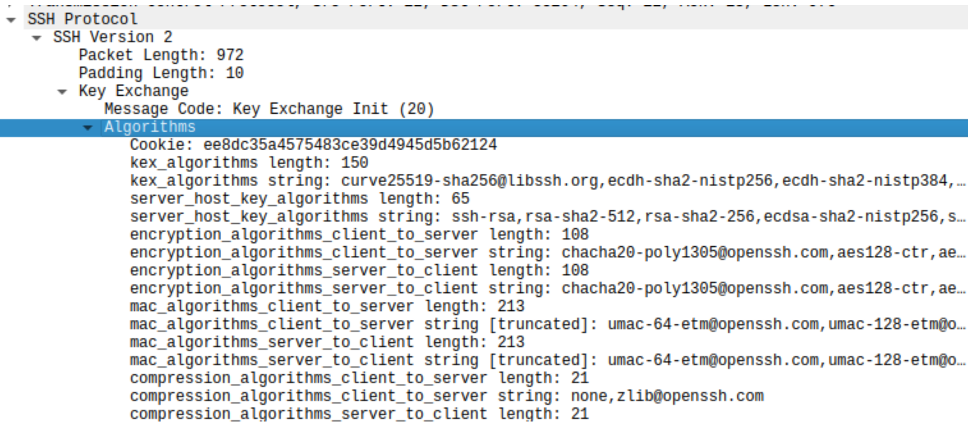
```
jreardon@jetblackbitey:~$ cat .ssh/id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAoCzLjGYY4mkCpK3Wm9BB37kVW2ON0ZcnLoxGINPQq8mazaBDDn38
dkWeXtxLimYt9665wjpq0IF9BYJ326GGbYkz6qqyC9OMgjSpslbPh7GdV0NsPexfUr7jy8
3e/038+Q9u3dHhTnEZDP3q24fpFYOEnJgisYo1O6KqVsUiyT2bIZg7qUvLvjlJzHdTfGv8
PdyNspPJE9mg3wirvzhjzgWXDoqqtgMZbu9jsWCIeKSeianZgvIvyWkHnT94aGXnK8/0Uo
ZcYSexjntYh0i2B/uWEj42TeJJTr1GCIQqkX24XlRqnu0QDQ6azqoMZRjeQmcyMb10P7Yi
mlrVC3ceVikp3BIuhU68z6UBOAB0ILDsR76cwJv+76VCoPW7+ZoUVkiwqG9TPEHQNqgJGC
XbyJQ9jQiKJfwbAwkr/3PKyUHfF1gIarZiTwSq2MBZWx3YmLlDFUu24ZPsHKf/jxKTypmh
```



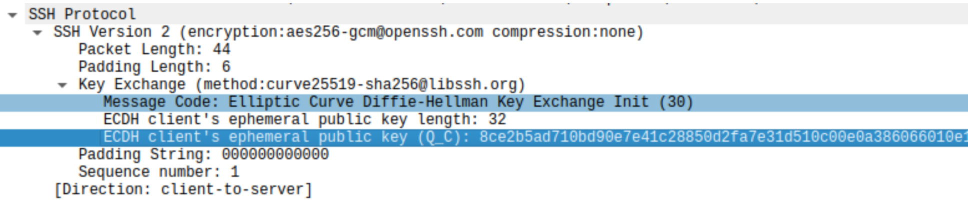
```
tc@localhost's password:
   ( '>')
  /) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
 (/-_--_-\)           www.tinycorelinux.net

tc@box:~$ vi .ssh^C
tc@box:~$ mkdir .ssh
tc@box:~$ vi .ssh/authorized_keys
```



```
joel.reardon@linux08-wc:~$ ssh tc@localhost -p 13898
Enter passphrase for key '/home/profs/joel.reardon/.ssh/id_rsa': 
```



```
▾ SSH Protocol
  ▾ SSH Version 2
      Packet Length: 972
      Padding Length: 10
    ▾ Key Exchange
        Message Code: Key Exchange Init (20)
      ▾ Algorithms
          Cookie: ee8dc35a4575483ce39d4945d5b62124
          kex_algorithms length: 150
          kex_algorithms string: curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,…
          server_host_key_algorithms length: 65
          server_host_key_algorithms string: ssh-rsa,rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nistp256,s…
          encryption_algorithms_client_to_server length: 108
          encryption_algorithms_client_to_server string: chacha20-poly1305@openssh.com,aes128-ctr,ae…
          encryption_algorithms_server_to_client length: 108
          encryption_algorithms_server_to_client string: chacha20-poly1305@openssh.com,aes128-ctr,ae…
          mac_algorithms_client_to_server length: 213
          mac_algorithms_client_to_server string [truncated]: umac-64-etm@openssh.com,umac-128-etm@o…
          mac_algorithms_server_to_client length: 213
          mac_algorithms_server_to_client string [truncated]: umac-64-etm@openssh.com,umac-128-etm@o…
          compression_algorithms_client_to_server length: 21
          compression_algorithms_client_to_server string: none,zlib@openssh.com
          compression_algorithms_server_to_client length: 21
```



```
▾ SSH Protocol
  ▾ SSH Version 2 (encryption:aes256-gcm@openssh.com compression:none)
      Packet Length: 44
      Padding Length: 6
    ▾ Key Exchange (method:curve25519-sha256@libssh.org)
        Message Code: Elliptic Curve Diffie-Hellman Key Exchange Init (30)
        ECDH client's ephemeral public key length: 32
        ECDH client's ephemeral public key (Q_C): 8ce2b5ad710bd90e7e41c28850d2fa7e31d510c00e0a386066010e1
      Padding String: 000000000000
      Sequence number: 1
    [Direction: client-to-server]
```

SSH-2

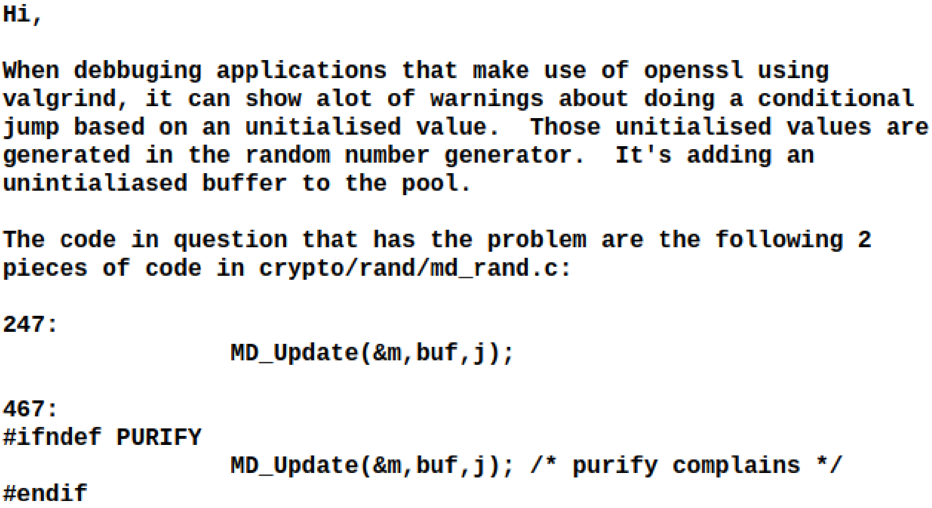
- complete rewrite of SSH
  - separate out the connection, transport, and authentication
- support for better cryptographic primitives
  - SSH-1 did not support AES
  - replaces server key with Diffie-Hellman key negotiation
  - adds support for using certificates
  - uses HMAC for message integrity
    - SSH-1 did not have real integrity
  - allows support for rekeying
    - idea was adversary with more ciphertext may be easier to break
  - different keys for direction and integrity
  - drops Kerberos
  - public key challenge is changed
    - client signs session identifier along with other connection information

Client Authentication

- SSH's key difference with TLS
  - the client is always authenticated
- makes ssh useful for accessing resources that are otherwise protected
  - e.g., git uses ssh to access a repository
  - now you don't have to send a username and password to pull and push code
  - or save a working copy of your password in memory or on disk
  - private key is stored on disk, but passphrase protected

CVE-2008-0166

- random number generator in Debian's openssl was predictable
  - all client keys for public key authentication were from a small domain
  - possible to brute force generate them all
  - log into any system that accepted a weak key
- issue was that key generate used uninitialized memory as randomness
  - not a good source of randomness

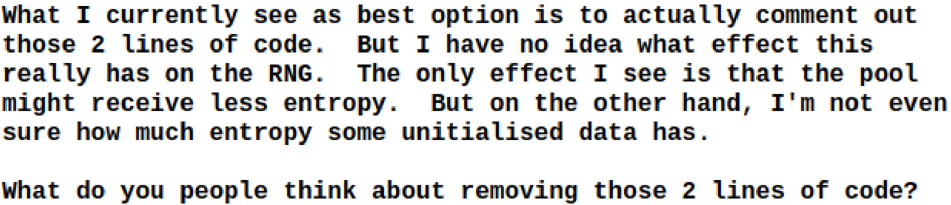


Hi,

When debbuging applications that make use of openssl using valgrind, it can show alot of warnings about doing a conditional jump based on an unitialised value. Those unitialised values are generated in the random number generator. It's adding an unintialiased buffer to the pool.

The code in question that has the problem are the following 2 pieces of code in crypto/rand/md_rand.c:

```
247:
                MD_Update(&m,buf,j);

467:
#ifndef PURIFY
                MD_Update(&m,buf,j); /* purify complains */
#endif
```



What I currently see as best option is to actually comment out those 2 lines of code. But I have no idea what effect this really has on the RNG. The only effect I see is that the pool might receive less entropy. But on the other hand, I'm not even sure how much entropy some unitialised data has.

What do you people think about removing those 2 lines of code?