

# Guess who?

Sebastian Kraemer *kraemer@cs.uni-potsdam.de*

March 7, 2005

## Abstract

A description of the capabilities of the SSH penetration toolset *guess-who*.

## 1 Introduction

The *guess-who* package contains various tools for testing the weaknesses of SSH server setups based on the SSH2 protocol. This ranges from single hosts to large networks, from basic password authentication to public key authentication, from public-key scanning to dictionary attacks against identity files. The available tools for these tasks are described throughout in this short paper.

## 2 Guessing passwords

The basic functionality you would expect from a SSH penetration tool is the ability to bruteforce passwords against a server:

```
linux:guess-who $ ./b -l root -h 127.0.0.1 -1 -N 5 < /usr/share/dict/words  
[ 00056 ][ 00003 ][ 00000018.660447 ][      root ][      Abelson ]
```

The first row shows the number of passwords tried, the second is the number of seconds elapsed, followed by the rate the login and the currently tried password. You can specify the number of threads to use via the `-N` switch. There are two concurrency modes you can use. The dumb mode which just spawns `N` concurrent threads via the `-2` switch, and the producer-consumer model via the `-1` switch which spawns `N` consumer threads and one thread for constantly opening new connections to the server, keeping open one constant virtual connection, regardless of the remote peer closing it due to wrong passwords.

Newer OpenSSH servers deprecate the use of the password authentication method and rather use a method called *keyboard-interactive*. It is basically the same (you can still use passwords to login) but the underlying mechanism in the protocol for the authentication changes. You can detect such server:

```
linux:guess-who $ ssh -x -v -oPreferredAuthentications=blub 127.0.0.1
OpenSSH_3.5p1, SSH protocols 1.5/2.0, OpenSSL 0x0090609f
...
2730: debug1: service_accept: ssh-userauth
2730: debug1: got SSH2_MSG_SERVICE_ACCEPT
2730: debug1: authentications that can continue: publickey,keyboard-interactive
2730: debug1: no more auth methods to try
2730: Permission denied (publickey,keyboard-interactive).
2730: debug1: Calling cleanup 0x8068d70(0x0)
```

This server only allows the *publickey* and the *keyboard-interactive* authentication methods. Using the *password* authentication method would be a waste of time here. So better check what the remote peer is speaking. If the server speaks *keyboard-interactive*, use the `-K` switch:

```
linux:guess-who $ ./b -l root -h 127.0.0.1 -2 -K -N 5 < /usr/share/dict/words
[ 00005 ][ 00001 ][ 00000004.995005 ][ root ][ abaft ]
```

It might be that certain messages are not accepted by the remote peer:

```
linux:guess-who $ ./b -l root -h 127.0.0.2 -1 -K -v -N 5 < /usr/share/dict/words
SSH2::userauth_kbd_interactive: 'ssh-userauth' message not accepted(51)
SSH2::userauth_kbd_interactive: 'ssh-userauth' message not accepted(51)
[ 00002 ][ 00001 ][ 00000001.998002 ][ root ][ Aaron ]
```

So better use the `-v` switch the first times to see whether you are trying the correct authentication methods and whether the number of threads is small enough to not overload the peer:

```
linux:guess-who $ ./b -l root -h 127.0.0.2 -1 -v -N 35 < /usr/share/dict/words
SSH2::banner_exchange::recv() Success[ root ][ aberrant ]
SSH2::banner_exchange::recv() Success
[ 00111 ][ 00005 ][ 00000022.195561 ][ root ][ abodes ]
```

The weird message will disappear if you decrease the number of threads hammering the SSH server. Once a valid login is found, it beeps, and a bang marks the correct password.

### 3 Scanning for weak logins

During pentest it might also be necessary to scan for weak logins in a large network. For example for `test/test` or `guest/guest` accounts:

```
linux:guess-who # ./pubscan -n 3 -l test -k test -P -r 127.0.0.1-127.0.0.11
127.0.0.1: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.2: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.3: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.5: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.4: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.6: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.8: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.7: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.9: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.11: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.10: SSH-2.0-OpenSSH_3.5p1 No
linux:/cvs-work/guess-who #
```

In this session, there is no `test/test` account available. It is using 3 threads (`-n`) for scanning. If the login would be valid, a `Yes` would tell you. As a special gift, the banner string is also printed out.

## 4 Scanning for public keys

In a similar way how you would scan for weak logins, you can scan for a certain public RSA or DSA key:

```
linux@guess-who # ./pubscan -n 3 -l test -k ~/.ssh/identity.rsa.pub -r 127.0.0.1-127.0.0.11
127.0.0.2: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.1: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.3: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.4: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.5: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.6: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.9: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.8: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.7: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.11: SSH-2.0-OpenSSH_3.5p1 No
127.0.0.10: SSH-2.0-OpenSSH_3.5p1 No
```

There's even no account *test* which uses this public-key file for authentication. There comes a tool called *keygen* along with the *guess-who* package which allows for creation of very short RSA files which even fit into email signatures. On DHCP networks you can identify hosts this way even if they obtain a different IP address each time.

There is a commercial version of the scanner available which is able to scan for logins and public keys on large networks (class-A for example) with a scan rate of multiple 1000 hosts per seconds, depending on the uplink.

## 5 Cracking passphrases of identity files

Eventually you can also mount a dictionary attack on identity files. These identity files which are used for remote login are usually protected by a passphrase. The *phrasier* program is able to run a dictionary against it and to unprotect these identity files.

## References

- [1] guess-who: <http://bh.segfault.net/SSH>