ScanSSH – Scanning the Internet for SSH Servers

Niels Provos and Peter Honeyman – University of Michigan

ABSTRACT

SSH is a widely used application that provides secure remote login. It uses strong cryptography to provide authentication and confidentiality. The IETF SecSH working group is developing SSH v2, an improved SSH protocol that fixes cryptographic and design flaws in the SSH v1 protocol. SSH v2 compatible server software is widespread.

Recently discovered security flaws make it critically important to find vulnerable SSH servers and update them. In this paper, we describe a method to determine with good precision how many servers supporting the various protocol versions have been deployed on the net.

We describe the design and implementation of ScanSSH, a scanner that probes SSH servers for their software version, and discuss the results of scanning the Internet and our local networks for several months.

Introduction

SSH is a client/server application that provides secure remote login [6]. SSH uses strong cryptography to provide authentication and confidentiality. The IETF SecSH working group is developing SSH v2, an improved SSH protocol that fixes cryptographic and design flaws in the SSH v1 protocol.

Among all security sensitive services, access to SSH servers is widely unfiltered. SSH v2 compatible server software is widely available, yet recently discovered security flaws in SSH software make updating many older servers on the Internet critically important. In this paper, we describe a method to determine with good precision how many servers supporting the newer protocol version have been deployed on the net.

In particular, we designed and implemented a scanner that probes SSH servers and classifies them according to their advertised version number. ScanSSH is the result of this effort.

In this paper, we describe the design and implementation of ScanSSH and our experiences in scanning the Internet and our local nets for a period of several months. We illustrate changes in the deployment of SSH protocols as measured by our software.

ScanSSH supports fast scanning of large networks, so we have used it to classify random SSH servers on the Internet and thereby obtain a rough proportion of the fraction of SSH servers of various breeds.

Our Internet measurements scan over two million addresses per run on a regular basis. The scans can be distributed over several machines. It is possible to pick non-repeating random addresses from specified network ranges, as well as to exclude networks with alert and overly cautious administrators. We use ScanSSH on our internal network to find all SSH servers and to ensure that all vulnerable ones have been updated.

Coincidentally, during the development and testing of ScanSSH, a major security hole was discovered in most of the known versions of SSH. We describe our experiences in using ScanSSH as a tool to help local administrators update their SSH servers to address these vulnerabilities.

The remainder of this paper is organized as follows. In the first section, we present the design goals for ScanSSH. We present implementation details and a performance analysis in the next two sections. Subsequently, we discuss the results from probing the Internet and local university networks. We present related work in and conclude in the final two sections.

Design Goals

In this section, we describe the design goals that underly the development of ScanSSH.

The scanner has two questions to answer:

- What is the ratio of deployed SSH v2 servers to SSH v1 servers on the Internet?
- Which hosts on a network run vulnerable SSH servers?

To determine a server's protocol version, it suffices to look at the first message in the SSH protocol. The SSH v2 transport draft states that when a connection has been established, both sides must send an identification string of the form SSH-protoversion-softwareversion [5].

Servers that support the SSH v2 protocol use a protocol version of either 2.0 or 1.99. We use the information contained in the softwareversion field to determine whether a server is running a vulnerable server version.

To retrieve this information, it is necessary to establish a TCP connection to the destination host and read the first protocol message in the SSH protocol.

We can estimate SSH server deployment by randomly sampling hosts on the Internet. The distribution obtained from a random sample is close to the actual distribution, given that the random sample is large enough.

IPv4 can address about four billion addresses, so we want to be able to scan addresses quickly, even for a large sample. IPv6 is not widely deployed and the address space is too large for random sampling, so we ignore IPv6 addresses.

To find vulnerable SSH servers on a given network, we need to be able to scan specific networks. Inevitably, we run afoul of network administrators suspicious of our scans. For that reason, we need a way to exclude particular hosts or networks from a scan.

In summary, we need an efficient scanner that takes random or complete samples of a number of networks while excluding particular hosts that suspicious administrators do not want scanned.

Implementation

In this section, we discuss how the implementation of ScanSSH achieves the design goals from the previous section.

Producer-Consumer Model

ScanSSH implements a producer-consumer model [1]. The producer is implemented as a single process that discovers reachable hosts. The number of consumers is configurable. The producer feeds addresses of reachable hosts to the consumers. Once a consumer reads an address, it establishes a TCP connection to obtain the SSH version string. The result is returned to the central producer and printed to stdout.

Address Generation

Networks and hosts to be scanned are specified on the command line either as a single IP address or in classless inter-domain routing (CIDR) [2] notation.

The addresses can be modified with special prefixes that determine if the scan is distributed over more than one machine or if addresses are randomly generated.

The following example line specifies random address generation for the class B network 192.168 and the class C network 10.1.0:

```
random(0,Apollo)/(192.168.0.0/
                16 10.1.0.0/24)
```

The zero specifies that all addresses should be scanned. Any other number specifies a limit on the number of hosts that should be scanned. The string ‘‘Apollo’’ is a seed for the pseudo-random number generator. Having a seed allows us to repeat a scan or distribute it on multiple hosts, because the generation of random numbers remains the same.

ScanSSH generates chunks of 64,000 addresses. The addresses in an address range are represented by a counter. The counter starts at zero and counts up to the number of available addresses. If the end of one network is reached, the next value of the counter represents the address at the beginning of the next network.

It is simple to generate addresses at random. Encrypting the counter with a block-cipher yields a random number that we can map to an Internet address. Using a counter guarantees unique addresses. Because encryption is bijective, the encrypted counter also produces unique addresses.

Most block ciphers use 64-bit blocks or larger. However, if we want to generate addresses for a /25 network, we need a block cipher using 7-bit blocks. We create a a variable block size that is related to the Tiny Encryption Algorithm (TEA) [4]. While the cryptographic security of TEA is not known with certainty, we don’t rely on its security, just its bijectivity.

A single round of the variable block size cipher looks as follows:

```
sum += 0x9e3779b9
cnt ^= rndsbox[(cnt^sum) & sboxmask]
                          << kshift;
cnt += sum;
cnt = ((cnt << left) | cnt >> right))
                              & mask;
```

We use the following constants:

```
kshift = left = bits / 2;
right = bits - left;
sboxmask = (1 << kshift) - 1;
```

The input to the block cipher is the block size bits and a counter cnt. After iterating the round 32 times, cnt contains the encrypted result. It is easy to see that the encryption function is bijective by observing that each round is reversible.

Producing Addresses

The producer process takes an address from the generated chunk of addresses and sends a TCP SYN packets to the host specified by the address. The destination port is set to 22, for SSH.

In the default configuration, ScanSSH supports 4096 outstanding TCP SYN packets. If we do not receive a reply in a certain time period, the TCP SYN packet is resent and the timeout increased. This process continues until the retry limit has been received. In that case, ScanSSH reports a timeout.

If we receive a response from a host, it is either a TCP RST segment or a TCP SYN/ACK segment. In the former case, ScanSSH reports a refused connection, and in the latter case, the address is put on a queue from which the consumer processes can feed.

Consuming Addresses

The consumers feed from the address queue in a round-robin fashion. We use standard inter-process communication to send an address to a consumer process. The consumer then establishes a TCP connection to the remote host, and waits for the SSH identification string.

After the identification string has been received, it is printed to stdout and the consumer sends its own version string SSH-1.0-SSH_Version_Mapper to the SSH server. The protocol number 1.0 causes a SSH server to close the connection, because protocol 1.0 is not specified.

Address Exclusion

ScanSSH reads an exclusion file that specifies the hosts that should not be scanned. If an address that falls in an excluded network range is generated, the address is ignored. In addition, private and multicast networks are ignored by default.

Performance Analysis

The performance of ScanSSH depends on the percentage of responsive hosts. By default, ScanSSH manages a list of 4,096 outstanding TCP SYN packets. When ScanSSH receives a reply to a SYN packet, the corresponding address is removed from the list and replaced by a new address. However, if a host does not respond to any packets, its address entry remains on the list for about half a minute. That means that the unresponsive hosts are the limiting factor for the scan rate.

Scanning the CITI network consisting of 384 addresses connected via a 100 MBit network takes about 36 seconds. The scan reports 62 active hosts with 44 of them running SSH servers. With a primed ARP cache, scanning the active hosts takes 0.2 seconds. Scanning only the 322 unresponsive hosts takes 36 seconds.

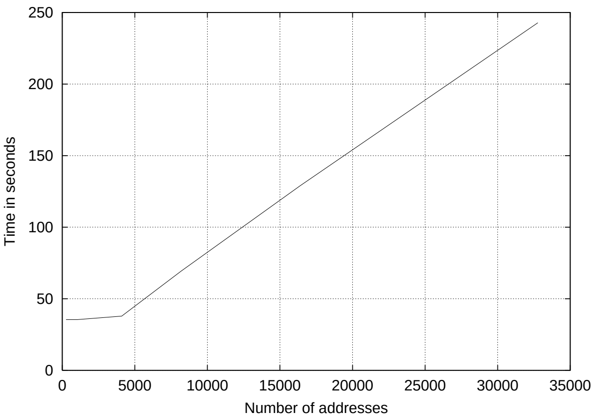

Figure 1: Worst case performance of ScanSSH when scanning only unresponsive hosts.

We measure the worst case performance of ScanSSH by scanning unroutable networks; see Figure 1. The average scan rate is about 135 hosts per second. We expect the worst case behavior to be a good estimate for scanning random addresses on the Internet, as most are unresponsive.

Measurement Results

In this section, we present results from measuring the deployment of SSH servers on the Internet and from assessing the number of vulnerable SSH servers running at the University of Michigan.

SSH Server Deployment

We have been scanning over two million random addresses on the Internet since July 2000. The scans are repeated monthly.

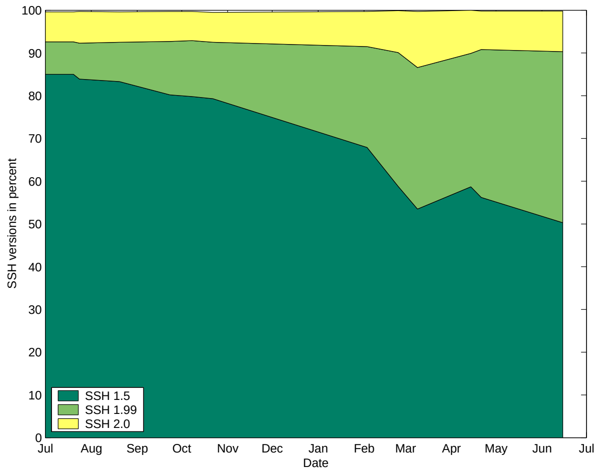

Figure 2: Deployment of SSH protocol versions on the Internet.

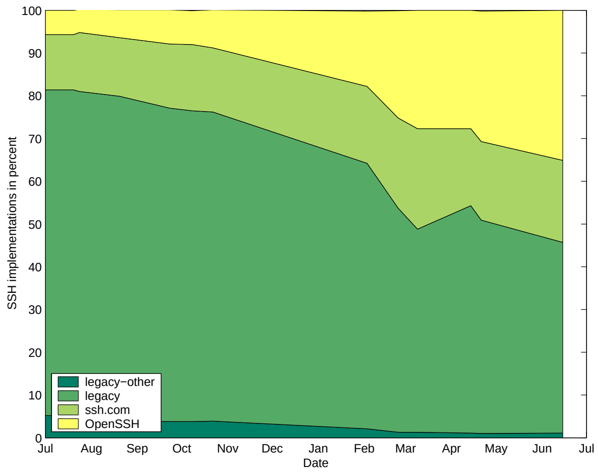

Figure 3: Deployment of different SSH server software on the Internet.

Figure 2 shows the deployed SSH protocols from July 2000 until the end of June in 2001. In July 2000, about 15% of all SSH servers supported SSH v2. At the end of our measurements in June 2001, almost 50% of all SSH servers support the version 2 protocol. Interestingly, the fraction of servers that support only SSH v2 remains almost constant. They change from 7% to 10%, whereas the percentage of servers that support both SSH v1 and SSH v2 increases from 8% to 40%. Examining Figure 3 provides an explanation for this behavior. The increase of SSH v2 capable servers is mostly due to OpenSSH. OpenSSH's contribution of SSH v2 servers increased from 1.7% in July 2000 to 30% in June 2001.

Because we sample hosts randomly, the data can be used to estimate the number of responsive hosts on the Internet, *i.e.,* hosts that are connected to the Internet and not hidden behind a firewall. Given the number of responsive hosts, we can estimate the percentage of hosts that run SSH services.

The upper graph in Figure 4 shows the data as measured. There is no significant change in the percentage of responsive hosts during our scan period. The slight variations might result from changes in the location from where the scans were conducted. However, we notice stronger variations in the percentage of hosts that run SSH servers. In February 2001, there is a significant drop in hosts running SSH servers. The drop might be correlated to a serious security problem in the "CRC32 Compensation Attack Detector" [7]. The thicker line in the graph represents a linear fit under that assumption.

However, a closer examination of the data shows that the distribution of SSH servers is not uniform. Figure 5 shows that certain areas in the network run significantly more SSH servers than others. We notice a strong decrease of servers for these addresses between our scans. We observe a similar decrease in the number of responsive hosts. One possible explanation is that the address of our scanning hosts has been filtered. When we remove the part of the network in which we measured the drop in responsive hosts, the number of SSH servers increases linearly, as shown in the lower graph of Figure 4.

SSH Vulnerability Scanning

Because SSH servers are a critical component of the Internet infrastructure, it is important to react quickly to security holes discovered in SSH server software. The remote root hole described in the previous section was a motivation for us to help to update vulnerable SSH servers at the University of Michigan.

To find vulnerable SSH servers, we scanned about 400,000 addresses on a daily basis. We found about 2,300 hosts running SSH services. To identify if a host was vulnerable, we check the software version returned by the SSH server against a table of software known to be vulnerable. We assign the category "unknown" to hosts for which we have no vulnerability information. The results are shown in Figure 6.

In February 2001, we found that 90% of the SSH servers running at the University of Michigan had known security problems. In the course of our research, we sent lists of vulnerable hosts to the administrators of the respective networks. After two months, the percentage of vulnerable SSH servers went below 20%. As security researchers, we think of

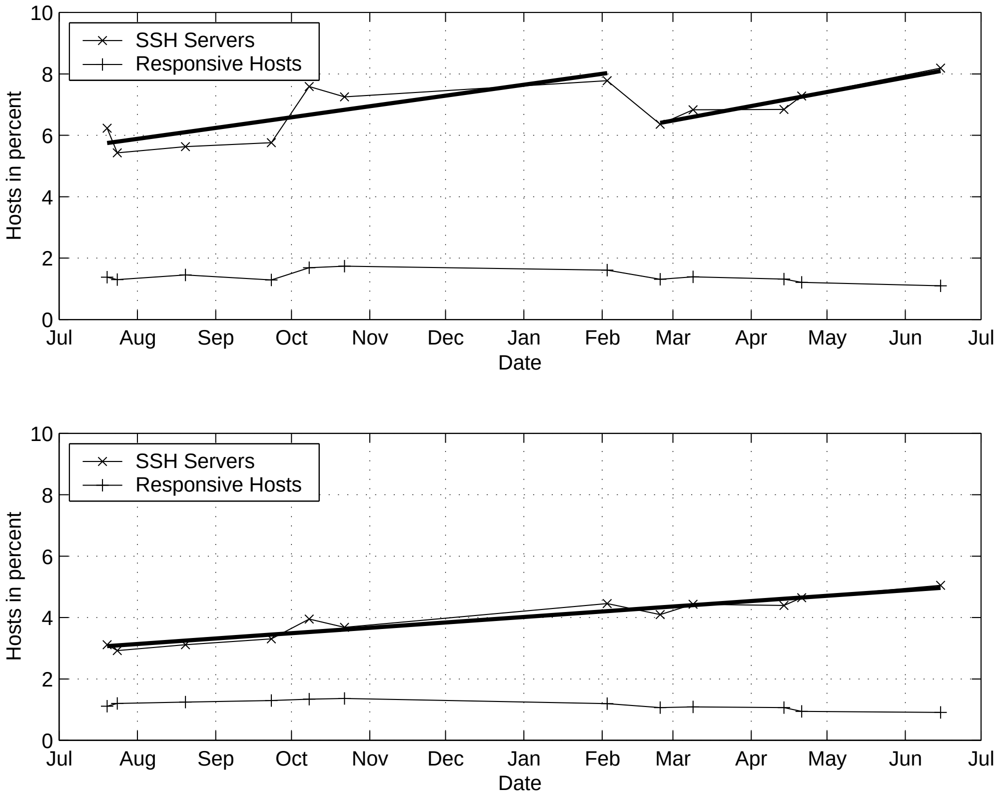

Figure 4: Percentage of responsive hosts and percentage of hosts running SSH servers. The upper graph shows the data as measured, the lower graph shows adjusted data.

this as a long time period. However, given the administrative needs and structure of the university (or any bureaucracy), the reaction time is not surprising.

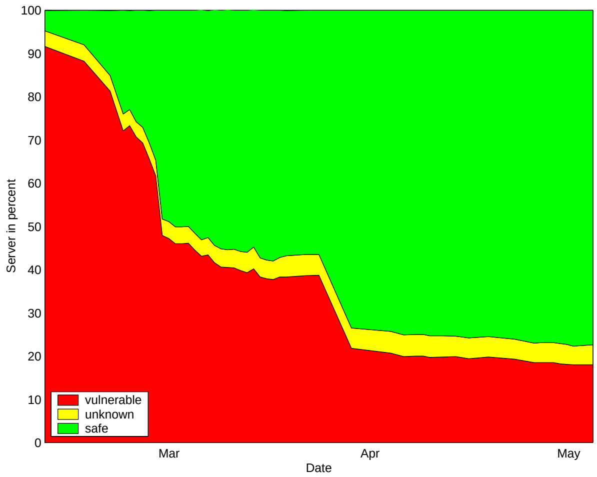

Figure 6: Percentage of SSH servers that are either safe or vulnerable to known security holes.

Related Work

Nmap is a port scanner that allows scanning multiple hosts at once [3]. Nmap scans hosts sequentially, i.e., a port scan on one host must complete before the next host is scanned. While that might be adequate for small networks, it is too slow for scanning many addresses. ScanSSH scans hosts in parallel and thus achieves higher scanning speeds.

Conclusion

We described the problem of measuring the deployment of the SSH v2 protocol and finding vulnerable SSH software.

To solve these problems, we implemented an efficient SSH scanner. We reasoned that efficiency was our main design goal and showed how our implementation meets it.

We measured the deployment of SSH servers on the Internet and showed how ScanSSH has been used to update vulnerable servers to more secure software.

ScanSSH has been released as UNIX source code under a BSD license and is available at http://www.monkey.org/provos/scanssh/ .

Acknowledgments

We thank Bob Beck, the University of Alberta, CORE-SDI and Peter Galbavy for providing network and computing resources for our scans. We thank David Wagner for helpful conversations about random number generation and Theo de Raadt for reviews and helpful suggestions about data analysis. We also thank Adam Moskowitz for shepherding this paper.

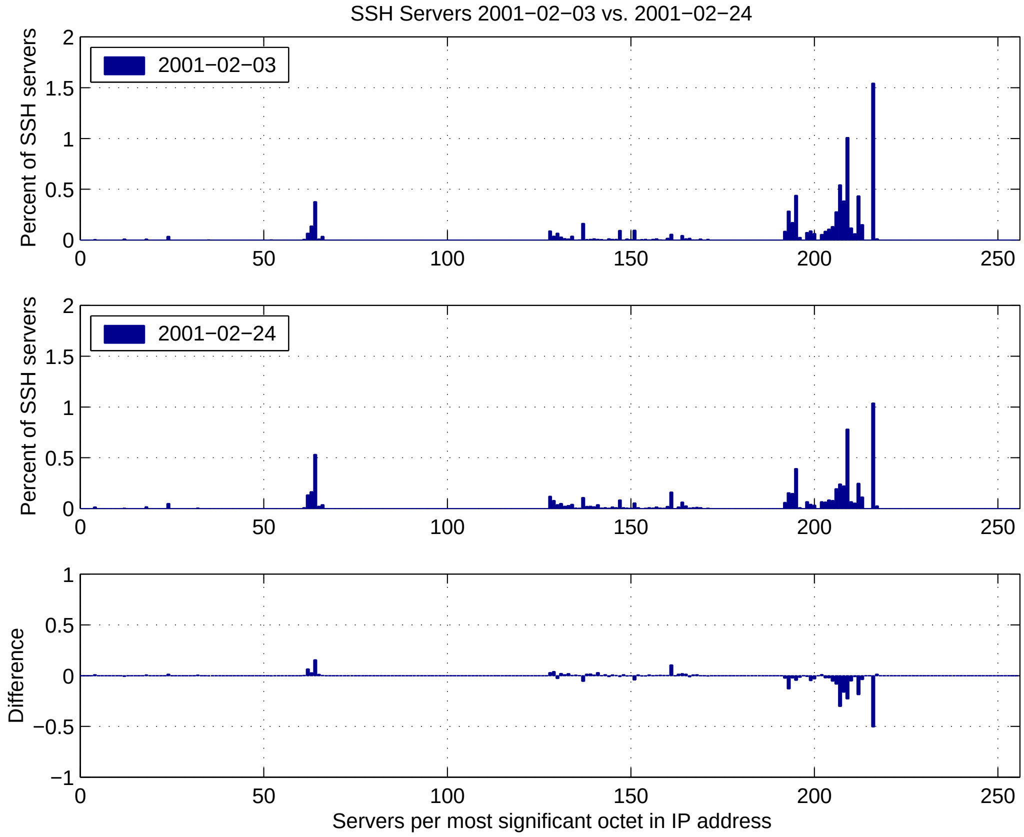

Figure 5: Comparison of SSH server distribution at the beginning and end of February 2001.

References

[1] Andrews, G. R. and F. B. Schneider, "Concepts and Notations for Concurrent Programming," *ACM Computing Surveys*, Vol. 15, Num. 1, pp. 3-43, March, 1983.

[2] Fuller, S., T. Li, J. Yu, and K. Varadhan, "Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy," *RFC 1519*, September, 1993.

[3] Fyodor, "Remote OS Detection via TCP/IP Stack Fingerprinting," http://www.nmap.org/nmap/nmap-fingerprinting-article.html , October, 1998.

[4] Wheeler, D. and R. Needham, "A Tiny Encryption Algorithm," *Technical Report 355*, University of Cambridge, December, 1994.

[5] Ylönen, T., T. Kivinen, M. Saarinen, T. Rinne, and S. Lethinen, *SSH Transport Layer Protocol,* Internet Draft, work in progress, January, 2001.

[6] Ylönen, Tatu, "SSH – Secure Login Connections over the Internet," *Proceedings of the Sixth USENIX Security Symposium*, pp. 37-42, July, 1996.

[7] Zalewski, Michael, "Remote Vulnerability in SSH Daemon crc32 Compensation Attack Detector," *RAZOR Bindview Advisory CAN-2001-0144*, Feburary, 2001.