Inoculating SSH Against Address Harvesting

Stuart E. Schechter* MIT Lincoln Laboratory ses@ll.mit.edu
Jaeyeon Jung MIT CSAIL jyjung@mit.edu
Will Stockwell* MIT Lincoln Laboratory bigwill@mit.edu
Cynthia McLain* MIT Lincoln Laboratory cdmclain@ll.mit.edu

Abstract

Address harvesting is the act of searching a compromised host for the names and addresses of other targets to attack, such as occurs when an email virus locates target addresses from users' address lists or mail archives. We examine how host addresses harvested from Secure Shell (SSH) clients' `known_hosts` *files can aid those attacking SSH servers. Each user's* `known_hosts` *file contains the names of every host previously accessed by its owner. Thus, when an attacker compromises a user's password or identity key, the* `known_hosts` *file can be used to identify those hosts on a network that are most likely to accept this compromised credential. Such attacks are not theoretical – a single attacker who targeted host authentication via SSH and employed* `known_hosts` *address harvesting was able to gain access to a multitude of academic, commercial, and government systems. To show the value of* `known_hosts` *files to such attackers, we present results of a study of* `known_hosts` *files and other data collected from 173 hosts distributed over* 25 *top level domains. We also collected data on users' credential management practices, and discovered that* 61.7% *of the identity keys we encountered were stored unencrypted. To show how host authentication attacks via SSH could evolve if automated, we survey mechanisms used to attack and their suitability for use in self-propagating code. Finally, we present countermeasures devised to defend against address harvesting, which have been adopted by the OpenSSH team and one of the two main commercial SSH software vendors.*

1 Introduction

The SSH protocol has done much to popularize the use of cryptography for remote command execution, file transfer, and other services. However, cryptographic channels alone are not enough to ensure these services will only be accessed by their intended users and for the purposes they authorize.

In 2004, weaknesses in the host authentication practices employed by SSH were exploited by a single attacker to compromise systems at a multitude of major universities, corporations, national laboratories, supercomputing centers, and even military installations [8, 17, 33, 22], with significant consequences. Operating system source code used to control routers was stolen from Cisco Systems [13]. Other sites had to be taken offline for multiple days [17, 8, 33]. Logs at one of the compromised sites showed SSH connections to hosts in the `known_hosts` file being initiated immediately after that file was read by the attacker [8].

Attackers are drawn to `known_hosts` files, maintained by the SSH client for each user, because it provides an abundant harvest of names and addresses of new target hosts to attack. Each file contains a list of every host previously contacted via SSH by the user, for the purpose of mapping these hosts to their public keys. Hosts are listed in the order in which they were first contacted, such that the bottom of the list contains the hosts at which the user's access credentials are most likely to remain valid. Such reliable target lists reduce both the time required to find vulnerable hosts and the likelihood that attacks will raise alarms due to failed TCP connection attempts or SSH authentication attempts.

To better understand the consequences of attacks that harvest addresses from SSH, we have initiated the first multi-institution study on SSH `known_hosts` relationships and key management, collecting data from 2,477 user accounts on 173 hosts. We use the data from this study, presented in Section 2, to explain how these `known_hosts` files have enabled attackers to repeatedly compromise host after host, and network after network. We show that a small number of promiscuous users are responsible for the majority of `known_hosts` entries, and that there exists a set of gateway hosts that contribute significantly to the vulnerability of the network.

In Section 3 we discuss the countermeasures that can be used to safeguard against address harvesting and the tradeoffs they require. In Section 4 we describe an additional enhancement to OpenSSH that we implemented to make it easier for users of gateway hosts to avoid risky practices that

* This work is sponsored by the United States Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

host information
OS name and version
SSH client vendor and version
IP address*
Netmask
user identification
Username**
IP address* of host exporting user's home dir
`known_hosts` file for each user
IP address* of each destination host
`authorized_keys` file for each user
Public identity keys**
identity key files for each user
Public identity keys**
Flag: set if matching private key is encrypted
SSH key version

Table 1. The contents of the report generated by `collect-ssh.pl`, organized by data source

	Hosts			*Hosts*	
TLD	*root*	*all*	*TLD*	*root*	*all*
.edu	12	73	.net	8	22
.com	10	17	.org	2	7
.de	4	5	.it	2	4
.uk	3	3	.kr	1	3
.pl	2	3	.ca	0	3
.fi	2	3	.dk	1	2
.nl	2	2	.be	1	2
.pt	1	1	.cl	1	1
.ph	1	1	.biz	1	1
.nu	1	1	.cx	1	1
.au	1	1	.ee	1	1
.gov	0	1	.se	0	1
.sk	1	1			

Table 2. Root submitting hosts and all submitting hosts grouped by top level domain (excludes hosts with undefined PTR records)

expose authentication credentials to abuse.

In anticipation of more sophisticated attacks, we examine mechanisms used to attack host authentication via SSH for their applicability to self propagating malware in Section 5. As we could find no past survey of mechanisms for impersonating users to authenticate to hosts via SSH, we provide such a survey in Appendix A.

Related work and related threats are discussed in Section 6. We conclude in Section 7 and discuss industry adoption of our proposed countermeasures in the epilog that follows.

2 Quantifying Harvestable Data

To better understand how attacks can spread via SSH, we have undertaken a multi-institution effort to collect data from users' `known_hosts` database entries and their overall SSH configuration. We made available a data collection and reporting script, written in Perl, that could be run on each host either by individual users to collect data from their own account or by system administrators to collect data from all user accounts. The data collection and reporting script is publicly available at `http://nms.csail.mit.edu/projects/ssh/`.

2.1 Collection methodology

The categories of information examined by our data collection and reporting script are summarized in Table 1.

All IP addresses collected, marked with a star (*) in Table 1, have been anonymized using the prefix preserving permutation algorithm of Xu *et al*. [31]. The prefix preserving property of the permutation ensures that two addresses within the same network before anonymization will fall into the network after anonymization. The public keys and usernames reported by our script, marked with two stars (**) in Table 1, are replaced by their SHA1 [15] hashes.

When our data collection script runs on a submitting host it queries the host's IP address and includes the anonymized address in its report. We call this the *submitter-view* IP address. When we receive the submitted report over a TCP connection, we collect the source IP address of the connection as our data collection server observed it and refer to the anonymized form of this address the *recipient-view* IP address. Submitter-view IP addresses allow us to differentiate two hosts behind a network address translation (NAT) box that may appear to our data collection server, which receives the submissions, to originate from the same address. The recipient-view IP addresses are needed to differentiate hosts behind two different NATs that may have the same submitter-view IP address, but are actually on different networks.

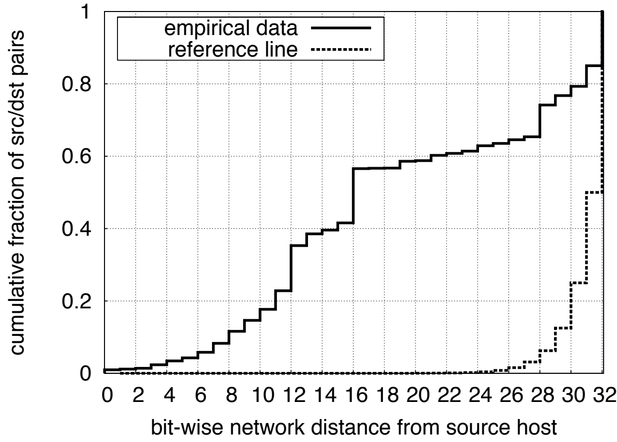


Figure 1. The distribution of bit-wise network distances between each unique `known_hosts` source/destination address pair, where the source is the host from which the `known_hosts` file containing the destination address was read.

2.2 Demographics

At the time of writing we have received data from 173 distinct hosts, which we refer to as *submitting hosts*. These hosts contain 2,477 user accounts with `known_hosts` files, which in turn contain a total of 37,765 `known_hosts` entries to 12,035 unique destination addresses. Of those 2,477 users, 2,359 were submitted by the 63 *root submitting hosts*, those submitting hosts on which the collection script was run as root and on which data were submitted from all users. For the remaining 110 submitting hosts, we received a total of 118 individual user submissions with at most two user submissions per host.

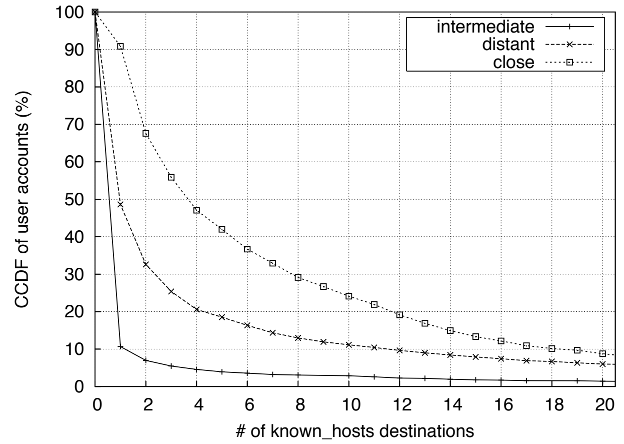
Our study drew its participants from various locations. Table 2 shows the distribution of submitting hosts across 25 top level domains. We infer the top level domain of a submitting host using a reverse DNS lookup of the recipient-view IP address before the address is anonymized.

2.3 The Reach of `known_hosts` Entries

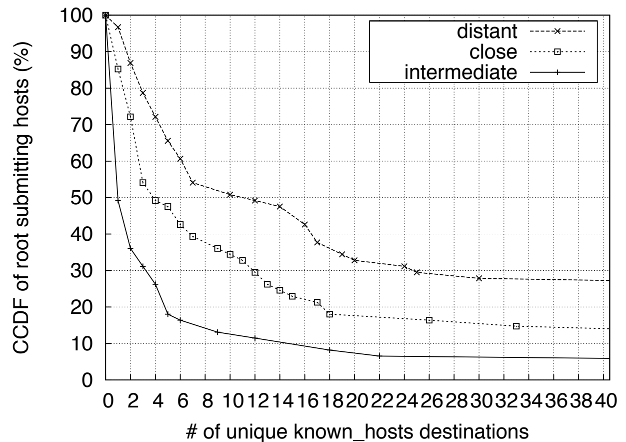
If an adversary gains access to one or more `known_hosts` files, where are the entries likely to lead? To answer this question we measured the bit-wise network distance between each source host and every unique destination address in its `known_hosts` files.[1] The bit-wise network distance between two addresses a_1 and a_2 is the number of bits that follows their longest common prefix. Figure 1 shows the cumulative fraction of all unique `known_hosts` source/destination pairs that fall within a given bit-wise network distance of each other. To illustrate that `known_hosts` destinations are locally biased, we placed a reference line representing the distribution of distances for address pairs chosen randomly from the uniform distribution (half of destinations would have a different high order bit and a distance of 32, one quarter a distance of 31, and so on.)

We segregated `known_hosts` destination addresses into three categories based on the bit-wise network distance from their source host. The roughly 60% of destinations found to be within the same /16 (Class B) as their source were categorized as *close* to that source. About 30% were in a different /8 than their source and were categorized as

[1] When the submitter-view of the source address differed from the recipient-view (see definitions in Section 2.1), we measured the distance from the source address that was closer to the destination address.



(a)



(b)

Figure 2. The complementary cumulative distribution function of unique `known_hosts` destinations of each distance category per user (a) and per root submitting host (b).

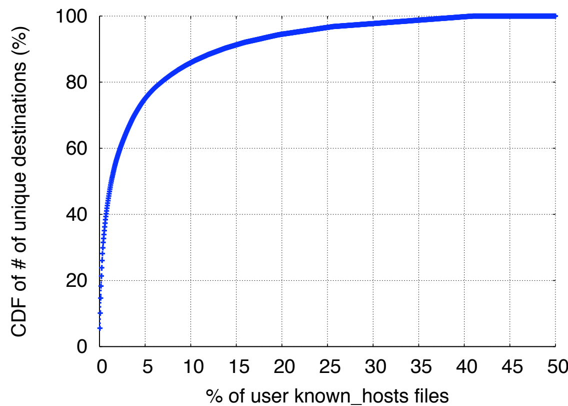


Figure 3. This figure shows the smallest possible percentage of user known_hosts files (X axis) needed to identify a given percentage of the unique destinations observed in our study (Y axis).

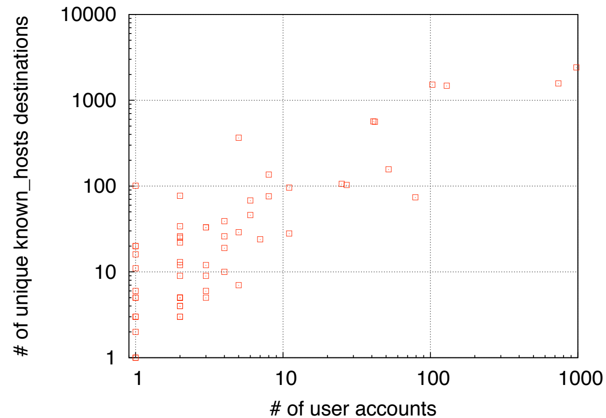


Figure 4. The relationship between the number of users with known_hosts files and the number of unique destinations over the set of root submitting hosts.

distant. The roughly 10% that remained, which fell within the same /8 as their source but not within the same /16, were categorized as *intermediate*.

We then asked, how many known_hosts destinations of each category could an adversary expect to discover on a host if he compromised one account or all of its accounts?

Figure 2(a) shows the distribution of unique destination addresses of each distance category over the known_hosts files collected. Nearly half of all users' known_hosts files contained entries to one or more distant destinations (49%) and four or more close destinations (47%). Over 10% of all users had more than ten distant known_hosts destinations and eighteen or more close destinations.

We show in Figure 2(b) the distribution of unique known_hosts destinations over the set of root submitting host. Unlike the distribution of unique destinations per user, there are more unique distant destinations per root submitting host than unique close destinations, as different users on the same host tend to connect to the same nearby hosts but different distant hosts. Half of all hosts contain known_hosts entries to fourteen or more distant destinations, and more than one quarter contain entries to over forty distant destinations.

Returning to our original question of where known_hosts entries lead, the data show that an adversary can not only identify nearby hosts but also those on distant networks. In fact, the known_hosts entries from the 173 submitting hosts collectively contain destinations to 107 different /8 (class A) prefixes, or 67% of all valid /8 networks.[2]

2.4 Evidence of Promiscuous Users and Gateway Hosts

The median-based analysis above de-emphazized the heavy-tail of the distribution that contains the 'promiscuous' users and hosts that contributed the most known_hosts entries. From Figure 3 we observe the 5% of users that were most promiscuous collectively had known_hosts entries to over 75% of the unique destination addresses collected in our study.

Figure 4 shows the expected correlation between the number of users with known_hosts files per root submitting host and the number of known_hosts entries per root submitting host. In the upper right-hand corner there are four hosts with more than 100 users and $1,000$ unique known_hosts destinations. We infer from the plethora of users accounts that these hosts act as gateways – hosts to which users connect via an incoming connection to the SSH server and then initiate outgoing connections via the SSH client. When users initiate outgoing connections via the gateway host's client, either the user's credentials must be sent to the client or a connection must be forwarded to the user's signing agent on the client. Should an adversary control a gateway host when a user initiates an outgoing connection, the adversary will either be able to steal the user's credentials or request that agent to sign on his behalf.

[2] The 160 valid class A networks are those that exclude two private /8 networks and 94 unallocated /8 networks, as documented by the Internet Assigned Numbers Authority [9, 10]

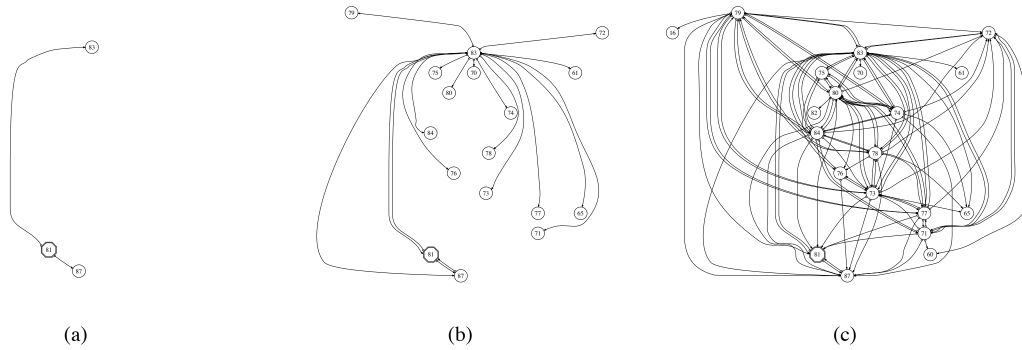


Figure 5. Submitting hosts within one network that are reachable in 1 (a), 2 (b), and 3 (c) steps from a source host by traversing `known_hosts` edges.

2.5 The Impact of `known_hosts` Harvesting on Remote Exploit Attacks

How much more quickly can an attacker identify a new target by using address harvesting than by using alternatives such as scanning? The answer depends on whether the attacker is looking for any host running an SSH server or only those hosts that accept one of the credentials that he has acquired. In this section, we focus on the former case, and assume that an attacker can break into most SSH servers by using a software or protocol exploit. We address the latter case in the following section.

Figure 5 illustrates how, given a remote exploit in an SSH server, an attacker starting from a single host could use `known_hosts` entries to rapidly walk through an institutional network from our study. Figure 5(a) shows the origin host and the submitting hosts that are destinations of its `known_hosts` entries. The submitting hosts that are destinations of these hosts are added in Figure 5(b). A third step yields the collection of submitting hosts in Figure 5(c). To simplify these graphs, the large set of destinations that are not submitting hosts or not within the same network are not shown.

To better quantify the rate of propagation, we computed, for each of the 173 submitting hosts, the number of additional target hosts that are reachable in a given number of steps. Figure 6 shows the median number of reachable hosts at each step. Over a hundred nodes are reachable in the second step and thousands in the third. The propagation then diminishes as we reach the limits of the data collected in our study.

To put address harvesting in context, we looked at the alternative of scanning for SSH servers. We used data from the 2001 study of Provos and Honeyman [18], who found that /8 networks with the densest population of responding SSH servers would respond to between 1% and just over 1.5% of scans. Within a densely populated academic network, they found that more than 10% responded to SSH scans. Randomly scanning the full IPv4 address space, they found the success rate was closer to 0.05%.

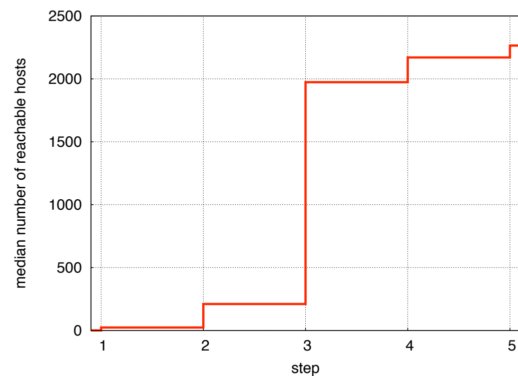


Figure 6. The median number of hosts that can be reached from an origin host by traversing all `known_hosts` edges.

In summary, a valid `known_hosts` entry can replace 10 to $2,000$ scans, depending on whether the attacker has already identified a densely populated network or if he is scanning randomly. Whereas `known_hosts` entries can be used to reach 2000 hosts in just three propagation cycles, hundreds of thousands to millions of scans would be required to reach this many hosts once nearby targets had been exhausted. As even a 20% connection success rate is low enough to indicate malicious behavior [19], forcing attackers to scan can be an effective means of making attacks detectable while simultaneously slowing them down.

2.6 The Effect of known_hosts on Speed of Credential Management Attacks

Attackers have used password cracking attacks and passwords collected via trojaned SSH clients to attack SSH servers without exploiting any flaws in the software itself. They could also easily modify these tools to crack the pass phrases of encrypted identity keys. For attacks that employ compromised credentials, the attacker must identify specific SSH servers that accept these credentials. As the attacker may discover and connect to SSH servers at which the credentials he has appropriated are not accepted, the connection success rate will be lower when measured as a fraction connections that result in successfully authenticated connections than those that simply complete the TCP handshake. We thus recommend that systems built to detect scans be instrumented to observe the result of the authentication process for SSH connections.

Finding distant hosts that accept a given set of credentials requires vary large numbers of scans. Even if stolen credentials are valid at 1000 distant accounts, an average of over two million random scans would be required to find the first viable target within the valid IPv4 space. Without a remotely exploitable vulnerability, attacking from network to network requires an approach more effective than scanning. An attacker will need additional information, whether it be obtained by harvesting addresses or waiting for the user of a compromised account to type in the next destination.

So long as known_hosts files map users to the IP addresses of hosts that are likely to accept their credentials, attackers need look no further when identifying new targets.

2.7 Evidence of Poor Credential Management

In addition to collecting known_hosts data, our collection script checked to see if identity key files were present and whether users had encrypted them with pass phrases. Of 447 identity key files collected, 276 (61.7%) were unencrypted and open to abuse by anyone with access to read them. Of the 12,035 unique known_hosts destination addresses discovered in our survey, 4,314 (or 36% of the total) originated from user accounts in which an unencrypted identity key was present.

3 Countering Address Harvesting Attacks

Host names and addresses may be stored in a program's configuration before execution, in its state during or between executions (known_hosts), and in logs for accounting or forensic purposes.[3] Each type of file has different restrictions regarding which parties need to be able to read or write to it, as illustrated in Table 3, and so different countermeasures (or variants on countermeasures) to address harvesting may apply to each. The most challenging file to manage is known_hosts, as it must be read and modified both by SSH and by those that use and administer it. We will take advantage of the fact that the common case is for the file to be read and appended to by the SSH client, and that users only need to access the file manually when locating or removing an entry.

	SSH		User/Admin	
	Reads	Writes	Reads	Writes
known_hosts	X	X	X	X
config files	X		X	X
log files		X	X	

Table 3. Of the files read or written by SSH that contain host names/addresses of other hosts, only known_hosts must be readable and writable by both SSH and its users.

3.1 Protecting known_hosts

To understand how SSH implementations could hide addresses in known_hosts databases, it is instructive to look at how password databases evolved to defend against similar threats. Early multi-user computers stored passwords in plaintext files and, like known_hosts files, relied upon the file system to prevent their misuse by keeping them secret. In 1974, Evans, Kantrowitz and Weiss [5] proposed that passwords be hashed with a one-way function before being stored in the password file.[4] Their key observation was that the host did not need to store the passwords themselves, but only enough information to later verify that a password provided to the host was the same one the user had previously provided. The same holds true for any sensitive datum, such as a host name or address, that must be tested for equality against a value provided in the future.

We present three similar approaches, summarized in Table 4, that also use one-way collision-resistant hash functions to obscure the names and addresses of hosts in known_hosts files. Each solution is more harvest-resistant than the last, but each comes with an additional usability cost.

[3] Addresses entered as command line parameters may also be stored in shell history files. The security community has known about the dangers of history files for some time. Fortunately, these files are unlike those discussed in this section in that they are not needed by the program, not needed for security purposes, and trivial to cleanse.

[4] For details on the adoption of this approach, see the early work of Robert Morris and Ken Thompson [14] or more recently Garfinkel *et al*. [7].

	Contents of known_hosts entry	Harvest resistance	Additional usability cost
(0)	name, ip_addr, key	None	None
(1)	$(s_1, h(s_1 \circ \text{name}))$, $(s_2, h(s_2 \circ \text{ip_addr}))$, key	Resists plaintext harvesting	New commands required to find/delete entries
(2)	$(s_1, h(s_1 \circ \text{name}))$, $(s_2, h(s_2 \circ \text{name, ip_addr}))$, key	Resists offline dictionary attacks on IPv4 address sace	User can't locate known_hosts entries using only their IP address
(3)	$(s_1, h(s_1 \circ \text{name} \circ \text{key}))$, $(s_2, h(s_2 \circ \text{name} \circ \text{ip_addr} \circ \text{key}))$, date_and_time_entry_added	Resists offline dictionary attacks on the IPv4 address space and on host names	User can't distinguish new key sent by host in known_hosts from key sent by unknown host. Adds need for key revocation lists.

Table 4. A summary of possible organizations for SSH known_hosts entries, where h is a one-way collision-resistant hash function and salts s_1 and s_2 are randomly generated for each entry. Each approach is incrementally more resistant to harvesting than the one above it, but incurs an incremental cost in usability.

Approach (1) – Simple name/address hashing

The simplest approach to prevent harvesting of plaintext host names and addresses is to hash their values as one would hash a password in the password file. Randomly generated salts, s_1 and s_2, are used to ensure that the work required to stage a dictionary attack against one entry cannot be re-used on other entries. The contents of the known_hosts file for this simple hashing strategy is summarized in row (1) of Table 4.

We first implemented this approach into OpenSSH 3.9 using SHA1 [15] as our hash function h and base64 encodings of random 64 bit numbers as salts. In response to earlier drafts of this paper, the OpenSSH development team coded their own implementation of this approach, which first appeared in OpenSSH 4.0.

When the SSH client is called upon to initiate a new connection, it checks the destination host name and address against the known_hosts database entry by entry. A special string ('<' in our implementation and '|1|' in the OpenSSH 4.0 implementation) indicates that the host name or address has been replaced with a hashed token. In this case, the destination host name or address is hashed using the salt extracted from the token, base64 encoded, and then compared to the hash encoded in the token. Matching encodings imply with high probability that the addresses match. To maintain backwards compatibility with earlier SSH implementations, a plaintext comparison between addresses takes place when the address in a known_hosts entry is not hashed.

Since entries in the known_hosts database are created and verified automatically by the SSH client, its behavior will remain unchanged from the user's perspective. We implemented two new commands for manipulating the known_hosts file should the user need to do so. remove-knownhost deletes a host entry from known_hosts by name and ssh-showkey returns the key of a host specified by name or address. In the OpenSSH 4.0 implementation, these commands are integrated as options in ssh-keygen.

To speed the transition to hashed host addresses we provide a program, ssh-hostname-encoder, that hashes all of the addresses in an existing known_hosts file. In OpenSSH 4.0, this functionality is accessible via a command option in ssh-keygen. We have also provided a Perl script, convert_known_hosts.pl, that can be run to convert all known_hosts files on a given filesystem into hashed host address format. As no such script was provided by the OpenSSH 4.0 team for their implementation, we have provided one at our project website (http://nms.csail.mit.edu/projects/ssh).

Approach (2) – Resisting IPv4 dictionary attacks

As with password files, the above hashing approach is potentially vulnerable to an offline dictionary attack. On IPv4 networks, the attacker can expect to identify an IP address with a worst-case average of 2^{31} SHA1 calculations. While this might be time consuming enough to slow spread and raise alarms, an attacker can decrease the expected work by starting with addresses near that of the compromised host (recall Figure 1). All of the nodes on the victim host's class C can be identified by performing less than 256 SHA1 calculations per known_hosts entry.

The possibility of dictionary attacks leads us to suggest that SSH client implementations may not want to store IP addresses at all. It should only be necessary to associate

the key with the address used by the user on the command line, which is most often the domain name. If hashed IP addresses must be stored, than we propose that it should be salted both with a random salt and with the host name, as illustrated by the `known_hosts` format in row (2) of Table 4. This will significantly increase the computation cost to attack networks where reverse DNS lookups and zone walking are disabled, and increase the likelihood of detection where reverse DNS lookups are enabled but monitored.

Approach (3) – Resisting all offline dictionary attacks

Host names are also subject to dictionary attack, especially if common names such as "gateway", "mail", and "database" are used or if DNS servers are configured to allow subdomain enumeration ("zone walking"). A design approach to eliminate offline dictionary attacks requires more fundamental changes to the way that SSH clients confirm that the host being contacted is indeed one that was last contacted at the same address. We propose that rather than storing entries that consist of hashed names mapped to the host's key, the SSH client should instead concatenate the host key onto the value to be hashed for the name and address entries as illustrated in row (3) of Table 4.

When a host is contacted in the future, its key will be retrieved before the `known_hosts` file is searched and so it is still possible to check whether the key is associated with any known host name/address pairs. Obtaining the keys requires that the attacker stage an online dictionary attack, contacting hosts that it may not be able to authenticate to and increasing the likelihood of detection.

The additional benefit incurs a significantly higher usability cost than the previous approaches. First, both the host name and the key are required in order to identify or remove an entry from the `known_hosts` database. If a key was lost and needed to be revoked, a revocation list would need to be employed to revoke all keys assigned to that host before the date on which the key was replaced. What's more, users would not be able to differentiate between the response received when they first contacted a host and the response received when a host's key changed. Fortunately, the correct security behavior in both cases should be the same – the user should check the hash of the server's key against a hash obtained through a secure alternate channel.

3.2 Protecting configuration files

Host address hashing can also be used to protect addresses in user-configured files such as the trusted host file (`.shosts`) and the user's main configuration file, so long as the host name need not be read until the host to be contacted has been identified. However, using incomprehensible tokens in place of plaintext addresses in these files may raise concerns for any sophisticated user or system administrator who may want to audit these files to ensure they do not place trust in the wrong remote hosts.

Fortunately, there is more flexibility in designing solutions to this problem than that of the `known_hosts` database, as configuration files are not written by SSH. Thus, solutions do not need to support mechanisms through which the SSH client or server can change the file.

To ensure that configuration files could be audited, a deterministic public key encryption function could be used instead of a hash function in order to obfuscate host names. An auditor with the private key would be able to reverse the function to verify its contents.

3.3 Protecting log files

The log files generated by the SSH server not only contain the names of other hosts running the SSH protocol suite, but also the names of users' with accounts on those hosts. While this information is dangerous in the hands of an attacker, it is essential to those tasked with detecting and tracking intrusions. Fortunately, in exploring the solution space to this problem we can take advantage of the fact that logs need not be written by users or administrators and, more importantly, that they need not be processed by anyone other than the system's administrators.

We can prevent log entries from being harvested if we can encrypt these entries to ensure that, once written, the data can not be read by the recording host. Only an auditor with a secret key should be able to translate the log back to its original plaintext form. A naïve algorithm to accomplish this would encode each entry in the log using a public key cryptosystem. Less computationally intensive approaches to securing audit logs have been introduced by Yee and Bellare [32], Schneier and Kelsey [20], and Waters *et al.* [28].

A simplified algorithm that meets our requirements can be constructed using a public key pair. When the SSH server begins executing, it creates a random session key k_0 for use with a faster symmetric cryptosystem. It then encrypts this k_0 with the public key and writes it to the log. Each log entry then begins with its sequence number, i, followed by the entry contents encrypted with symmetric key k_i, where $k_i = h(k_{i-1})$. Once the logging function has encoded the entry, it immediately calculates k_{i+1} and discards k_i from memory. To read a previous entry would require one to derive k_i from k_{i+1}, which in turn would require breaking the one-way hash function.

4 Encouraging Safer Use of Gateway Hosts

In Section 2.4 we saw evidence of the existence of large gateway hosts, ideal targets for credential theft attacks such as password cracking (see Section A.2) and for harvesting addresses of hosts that may accept stolen credentials.

While hindering attempts to harvest addresses can help to thwart the spread of attacks through gateway hosts, it is better to avoid running SSH clients on these hosts altogether. An ideal SSH gateway is one on which the SSH server, but not the SSH client, is installed, and through which users can forward TCP connections but execute no other operations. To make it easier to use such gateways, we have implemented a new SSH client command option, `-H`, which indicates the start of a new connection within a chain of cascading connections.

```
ssh -H gateway -H server
```

In the above example, the client establishes one connection to `gateway` and then establishes a secure channel from the client to `server` through `gateway`. The syntax supports the cascading of connections through any number of gateway hosts, all with encrypted connections back to the user's immediate client. Options specified before the first `-H` are applied to all hosts in the chain, whereas options specified between a `-H` and a host name are applied only to the connection to that host. The implementation uses UNIX domain sockets for communication in order to avoid opening TCP/IP ports accessible to other users.

Existing methods do exist for chaining through local hosts and are documented in the text *SSH, the Secure Shell: The Definitive Guide* [1], but each has serious drawbacks that have kept them from being widely used. One such approach uses local TCP/IP port forwarding to establish the connection from the client to the server through the gateway. One of the disadvantages of this approach is that other users on the client host could connect to the port to bypass the gateway as TCP/IP cannot distinguish between these users. Another disadvantage is that the method required proper use of the `HostKeyAlias` configuration option so that the connection forwarded through the local port isn't treated as a connection to `localhost` in the `known_hosts` database. Finally, the user must use two different shells on the client host to initiate clients and their connections to the gateway and server.

SSH connections could be forwarded through gateway hosts by setting up a proxy in the user's SSH configuration file, but proxy entries must be created for each gateway. The presence of gateway proxy commands in the user's configuration would also become an attractive target for harvesting and cannot be trivially obfuscated using the techniques presented in this paper. Proxy forwarding also presents problems if the gateway uses any form of interactive authentication, such as the use of passwords. Given the complexity and limitations of the available options, it's little wonder that most users have taken the route that's simplest for them: issuing a command to the SSH client on the gateway host if one is available.

5 Anticipating Future Attacks that Target Host Authentication via SSH

The widespread SSH attacks of 2004 are believed to be the work of a single individual [8, 13] and were not fully automated. Once he had obtained a user account on a host, the attacker would attempt to gain root access or find other ways to compromise other accounts on the host, such as by exploiting a vulnerability in NFS [22]. A self-propagating worm armed with a common vulnerability for escalating user privileges to root privileges (or at least gaining access to password/key files) could use the same set of impersonation attacks. Such a fully automated attack would not be limited by the time available to the attacker and could cause significantly more damage. Many of the components for such a worm, such as trojan SSH client code, password cracking algorithms, and tools to perform online dictionary over SSH, are readily available. A list of the potential mechanisms of attack are summarized in Table 5, and surveyed in detail in Appendix A.

To spread quickly, an SSH worm would need to infect as many new hosts as possible immediately after each host is compromised. Upon compromising a gateway host, a worm could impersonate that host's users by taking over outgoing SSH client sessions (row I1 in Table 5) or by using forwarded agents to authenticate on its behalf (T2) and then adding keys to the remote `authorized_keys` file (I2). Upon compromising a host of any type, a worm could immediately search for unencrypted identity keys in the file system (T1) and extract identity keys from running agents (C2). Obtaining root access to the compromised hosts would enable these attacks to be carried out using data from all of the host's users, and would then enable the worm to begin an offline dictionary attack to obtain any password and key credentials that it does not already have. The `known_hosts` file enables the worm to immediately identify hosts on distant networks that may accept stolen credentials.

A worm can also steal credentials by interacting with users, recording passwords as users login to SSH servers it has compromised (C1) and observing passwords sent to SSH clients and agents that it has compromised (C5). By starting this process immediately, the worm may be able to steal credentials from administrators should they detect the worm and login in an attempt to remove it.

	Attack	Event triggering attack opportunity	root not required	non-interactive	stealthy
T1	Extract unencrypted identity key stored on host trusted by credential holder	User's account or host compromised	*	X	X
T2	Forwarded agent used to authenticate attacker	Compromise of account or host already running forwarded agents	*	X	X
C1	Password stolen by compromised SSH server	New password-authenticated session to compromised server	X		X
C2	Identity key extracted from SSH agent processes	Compromise of host running agent processes	*	X	X
C3	Online dictionary attack on password file	Authentication initiated with correct username/password guess	X	X	
C4	Offline dictionary attack on passwords and identity keys	Password hash computation completed with correct password guess		X	X
C5	Password or key entered into previously compromised SSH client or agent	SSH client/agent executed on compromised host	X		X
I1/2	Session/Credential insertion attack	User's account or host compromised	*	X	X

Table 5. Attacks on SSH and the properties that make them amenable for use in a worm. An X indicates either that an attack can be run from a user account (*root not required*), need not wait for interactive user events in order to spread (*non-interactive*) or would not require excessive network traffic (labeled *stealthy*). A star (*) indicates that the attack can run without root privileges, but only against accounts available to the compromised user.

6 Related Threats & Related Work

The Morris worm of 1988 harvested target addresses from files such as `.rhosts` and `.forward`, and used offline dictionary attacks to crack passwords [23]. Because the Morris worm predated the advent of SSH, `known_hosts` files were not available for harvest.

Trojaned SSH clients that collect passwords have also become widespread, and a number of these have been lifted from compromised hosts [4]. Such a tool was used by the perpetrator of the major SSH attacks of 2004, who compromised hosts in the U.S. military, NASA, supercomputing centers including those supporting the TeraGrid, and a slew of universities [8, 17, 33, 22]. However, to date there has been no evidence of a true SSH worm which could spread without the need for user interaction.

Worms targeting protocols other than SSH, such as Lovgate [27], Deloader [25, 6], and Gaobot [26] already use online dictionary attacks to bypass host authentication without user interaction. While such brute force attacks are among the least effective, they are frequently found in the wild because they are among the easiest to write. The tools required to carry out online dictionary attacks against SSH have been automated and made publicly available [21]. Evidence of their use appears in our logs, anecdotal reports of other network researchers, and publicly available reports from the SANS Internet Storm Center [3].

7 Conclusion

We showed in our study that `known_hosts` files provide ample remote targets, with a median of fourteen addresses to distant networks identified per host. Each distant `known_hosts` entry an attacker discovers saves hundreds to thousands of scans to find a distant SSH server to attack, and many more scans if the attacker seeks a server that will accept a specific user's credentials. We also identified important outliers in the data, such as gateway hosts that contain hundreds of `known_hosts` files with entries to thousands of unique destinations. The distribution of unique destinations per user is heavy-tailed, as just 5% of users' `known_hosts` files collected contained 75% of the unique destination hosts discovered in our study.

In anticipation of future attacks, we explored how the mechanisms of attack used in the past and their suitability for use in self-propagating malware that could result in significantly greater damage than prior attacks.

In response to last year's attacks and in anticipation of future attacks, we presented a series approaches for hiding `known_hosts` destination addresses. These approaches include countermeasures not only against plaintext address harvesting, but also those that protect against dictionary attacks on local IP addresses and host names. In the epilog, we discuss the adoption of these countermeasures.

Epilog

This paper was first conceived in early 2004 and drafts have been in private circulation since June of that year. Only late in the year did we first learn of the major attacks against SSH that were underway.

On February 15, 2005 an updated draft was submitted to officials at F-Secure, SSH Communications Security Corp., and the OpenSSH development team.

OpenSSH responded by creating their own implementation of host address hashing as part of OpenSSH 4.0 on March 9, 2005. Unfortunately, this implementation does not come with a script with which a system administrator can update all of `known_hosts` files on a system. We have provided such a script and instructions for turning hashing on at `http://nms.csail.mit.edu/projects/ssh/`.

WRQ, which took over the F-Secure SSH product line in October of 2004 and re-branded the product under the "reflection" name, has also since committed to provide an option to store host names in a hashed format [29].

Petri Sakkinen of SSH Communications Security Corp. wrote in an email [16] on May 17, 2005 that "SSH will consider adding key hashing support in future versions of SSH Tectia, if our enterprise customers want to deploy that approach." The company also posted a statement to its web site [24].

Acknowledgments

The authors are indebted to a number of unnamed individuals who have contributed their time and effort to either anonymously contribute data to our efforts or to cajole others to do so.

The authors would like to thank Lou Anschuetz, Hari Balakrishnan, Robert Cunningham, David Dagon, Victor Hazlewood, Glenn Holloway, Roger Khazan, David Molnar, Scott Pinkerton, Michael D. Smith, Bill Yurcik, and our anonymous reviewers for their advice and comments. Stuart Schechter would like to thank the National Science Foundation for supporting this work while he was at Harvard under grant number CCR-0310877, as well as the MIT Lincoln Laboratory Advanced Concepts Committee and the Air Force for their support since his arrival at the laboratory. Jaeyeon Jung and Will Stockwell would like to thank the NSF for support under Cooperative Agreement number ANI-0225660.

References

[1] D. J. Barrett and R. E. Silverman. *SSH, the Secure Shell: The Definitive Guide*. O'Reilly Media, Inc., Sebastopol, CA, Feb. 2001.

[2] M. Bishop and D. V. Klein. Improving System Security via Proactive Password Checking. *Computers and Security*, 14(3):233–249, 1995.

[3] S. I. S. Center. Port Graph (for port 22). `http://isc.sans.org/port_details.php?port=22&days=70`.

[4] D. Dagon. Georgia Institute of Technology, Email correspondence, Dec. 10, 2004.

[5] A. Evans Jr., W. Kantrowitz, and E. Weiss. A User Authentication Scheme Not Requiring Secrecy in the Computer. *Communications of the ACM*, 17(8):437–442, 1974.

[6] F-Secure. F-Secure Virus Descriptions: Deloder. `http://www.f-secure.com/v-descs/deloader.shtml`.

[7] S. Garfinkel, G. Spafford, and A. Schwartz. *Practical UNIX & Internet Security*. O'Reilly Media, Inc., Sebastopol, CA, 3rd edition, Feb. 2003.

[8] V. Hazlewood. Security Technologies Manager, San Diego Supercomputer Center (SDSC), Telephone correspondence, Jan. 18, 2005.

[9] Internet Assigned Numbers Authority. Internet Protocol v4 Address Space. `http://www.iana.org/assignments/ipv4-address-space`.

[10] Internet Assigned Numbers Authority. *RFC 3330: Special Use IPv4 Addresses*. IETF, Sept. 2002.

[11] M. Kaminsky. *User Authentication and Remote Execution Across Administrative Domains*. PhD thesis, Massachusetts Institute of Technology, Sept. 2004.

[12] M. Kaminsky, E. Peterson, D. B. Giffin, K. Fu, D. Maziéres, and M. F. Kaashoek. REX: Secure, Extensible Remote Execution. In *Proceedings of the 2004 USENIX Annual Technical Conference*, pages 199–212, June 2004.

[13] J. Markoff and L. Bergman. Internet Attack Called Broad and Long Lasting by Investigators. *The New York Times*, May 10, 2005.

[14] R. Morris and K. Thompson. Password Security: A Case History. *Communications of the ACM*, 22(11):594–597, 1979.

[15] N. I. of Standards and Technology. Secure Hash Standard. FIPS PUB 180-1, Apr. 17, 1995.

[16] Petri Sakkinen, Director, Solution Marketing, SSH Communications Security. "SSH Worm" and SSH Communications Security. Email correspondence, archived at `http://nms.csail.mit.edu/projects/ssh/SSHIncEmail.html`, May 17, 2005.

[17] S. C. Pinkerton. Network Solutions Manager, Argonne National Laboratory, Email correspondence, Feb. 4, 2005.

[18] N. Provos and P. Honeyman. ScanSSH - Scanning the Internet for SSH Servers. In *Proceedings of The 15th USENIX Systems Administration Conference (LISA)*, 2001.

[19] S. E. Schechter, J. Jung, and A. W. Berger. Fast Detection of Scanning Worm Infections. In *Proceedings of the Seventh International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, Sept. 15–17, 2004.

[20] B. Schneier and J. Kelsey. Secure Audit Logs to Support Computer Forensics. *ACM Transactions on Information and System Security (TISSEC)*, 2(2):159–176, 1999.

[21] K.-O. Security. SSH Remote Root Password Brute Force Cracker Utility. `http://www.k-otik.com/exploits/08202004.brutessh2.c.php`, Aug. 20, 2004.

[22] A. Singer. Tempting Fate. *;login: The USENIX Magazine*, 30(1), Feb. 2005.

[23] E. H. Spafford. The Internet Worm Program: An Analysis. Technical Report CSD-TR-823, Purdue University Department of Computer Sciences, 1998.

[24] SSH Communications Security. Statement: Secure Shell and Address Harvesting. `http://www.ssh.com/company/newsroom/20050518_mit.html`, May 18, 2005.

[25] Symantec. Security response–W32.HLLW.Deloder. `http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.deloder.html`.

[26] Symantec. Security Response–W32.HLLW.Gaobot.AA. `http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.gaobot.aa.html`.

[27] Symantec. Security Response–W32.Lovgate.mm. `http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.lovgate@mm.html`.

[28] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an Encrypted and Searchable Audit Log. In *Proceedings of the 11th Annual Network and Distributed Security Symposium (NDSS '04)*, Feb. 1–6, 2004.

[29] WRQ. A Hypothetical Threat to SSH: What Customers Need to Know. `http://www.wrq.com/products/whitepapers/0976.html`, June 2005.

[30] T. Wu. The Secure Remote Password Protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, Mar. 1998.

[31] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon. Prefix-Preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme. In *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP '02)*, Nov. 12–15, 2002.

[32] B. S. Yee and M. Bellare. Forward Integrity for Secure Audit Logs. Technical report, University of California at San Diego Department of Computer Science and Engineering, Nov. 1997.

[33] W. Yurcik. Senior Systems Security Engineer, The National Center for Supercomputing Applications (NCSA), Telephone correspondence, Jan. 28, 2005.

A Host Authentication Attacks via SSH

The mechanisms described below exploit access to one or more compromised accounts on one host in order to impersonate those users who also have accounts on other hosts in order to gain access to those other hosts.

A.1 Exploiting misplaced trust

SSH servers and user accounts are often configured to trust other hosts to act on their behalf, to authenticate users, or to safely store user credentials. All of these practices are potential targets of attack.

T1 — Exploitation of trust in other hosts to authenticate user or secure store credentials
If an attack comes from a compromised host that is listed in the `shosts.equiv` or `hosts.equiv` file in the target server's `/etc` directory, or the `.shosts` or `.rhosts` file of the targeted user, the attacker will be permitted to connect to a target user's account without presenting user credentials.

Even if no hosts are explicitly trusted to authenticate on behalf of the target host, such trust is often implicit. Many users place their public identity keys in their `authorized_keys` files on SSH servers and leave their secret identity key unencrypted on hosts they use as SSH clients, trusting that these accounts will not be compromised. If one such client account or host is compromised, then the attacker can read the unencrypted identity key and use it to authenticate to the target host.

T2 — Abuse of forwarded authentication agent
Authentication agents are programs employed by users to authenticate on their behalf. They free users from the need to retype the pass phrases that protect their identity-key credentials each time that they authenticate.

A user can configure his agent to authenticate on his behalf when accessing services from an application run on a remote host. However, most SSH agents do not verify that the actions a remote host performs are the actions the user intended to authorize. Thus, when the user believes he is authorizing a CVS transaction he may instead be authorizing an SSH connection to a host targeted by the attacker.[5]

A.2 Credential theft

An attacker who can obtain a user's credentials can impersonate that user on any host that accepts these credentials. An attacker may choose from any of a number of approaches to steal credentials.

[5]The agent in Michael Kaminsky's remote shell client, REX [11, 12], provides a partial solution to this problem by verifying that the service being authorized (but not the command or parameters passed to the service) is indeed the one that the user intended.

C1 — Password theft by compromised SSH server
When authenticating via passwords, the SSH client will send the user's password credentials to the server over an encrypted channel. When the user's password arrives, it is then decrypted into plaintext before it is checked against the password file. If the server belongs to, or has been compromised by the attacker, then the attacker can modify the SSH server to collect these passwords. The attacker can then proceed to gain access to other hosts on which this password is used for authentication.

This attack can be thwarted if the client is configured to authenticate via a challenge-response protocol, such as SSH identity-key authentication or the Secure Remote Password (SRP) extension [30].

C2 — Extraction of keys from authentication agents
To free users from the need to retype the pass phrases that protect their identity key credentials, an authentication agent must keep these credentials in its memory.

Once an account is compromised, an attacker can search the process table for active authentication agent processes. While the SSH agent takes care to instruct operating systems not to allow its memory to be dumped, an attacker with sufficient privileges will be able to inspect its memory space in order to locate identity key credentials.

C3 — Online dictionary attacks
An online dictionary attack is staged by repeatedly attempting to authenticate to a remote host using common passwords. Intrusion detection systems can be trained to detect these attacks and terminate communications with attacking hosts. However, if an attacking host is permitted to continue these attacks and chooses a large set of targets, it will eventually find servers that allow continued connection attempts and employ common passwords.

C4 — Offline dictionary attacks
After obtaining the password file on a compromised host, an attacker can test candidate passwords against the password file or try to decrypt identity key files in user home directories. While it is likely that an attacker who could access the password file could compromise this account without the password, chances are that the user employs this password to authenticate to other hosts as well. Such offline dictionary attacks also differ from their online counterparts in that the attacker need not run the authentication protocol. This is advantageous because executing a network protocol increases the risk that alarms will be activated and introduces a network delay for each password tested. Once a user's credentials have been compromised, the attacker can use them to gain access to other hosts on which they are accepted.[6]

[6]A 1995 study by Bishop and Klein [2] showed that 40% of passwords were crackable. More recent reliable statistics on the percent of crackable passwords are harder to find. Suffice it to say that while user awareness of weak passwords may have improved since then, the sophistication of

C5 — Eavesdropping by client software or host
A patient attacker who has compromised a user's account can modify or observe the SSH client and agent to collect passwords and identity key pass phrases as the user types them. The host address, username, and password triplets discovered can either be stored or sent directly to the attacker.

Many users find it convenient or necessary to open SSH clients on hosts to which they are already connected via SSH. We use the term *gateway hosts* to describe those hosts to which a user connects via SSH from a client and from which the user then initiates a new SSH connection to another host running the SSH server. It is often necessary to use gateway hosts when firewalls prevent direct access from the user's immediate client to his or her desired destination host. SSH may also be employed to protect file transfers, CVS commands, or other services required by software that is run at a gateway host. Attackers can strike users on these gateway hosts even if an SSH server is not run on the user's immediate client.

A.3 Insertion attacks

An attacker may be able to insert his own commands into a user session or insert his own credentials in place of a legitimate user's credentials. The first attack described below, in which the attacker impersonates the user for part of the SSH session, can be used to perform the latter attack, which allows the attacker to impersonate the user in future sessions.

I1 — Session capture and command insertion
While proper use of identity keys, authentication agents, and agent forwarding can protect against credential theft at gateway hosts, these practices cannot protect the user if the host he runs the SSH client on is compromised. All communications are decrypted and then re-encrypted at the client, and software at this host can insert, modify, or delete information at will.

I2 — Credential insertion or replacement
An attacker can insert an identity key into the user's `authorized_keys` file. The SSH server depends on this file to determine which keys the user has authorized to serve as his credentials. If the compromised user's home directory is located on a shared file system, the attacker then uses the inserted identity keys to authenticate as that user to other hosts that mount the user's shared home directory.

If the attacker can write to the system password file, he can replace any or all user passwords with those of his choosing.

---

cracking algorithms has also improved and the speed of computers used to crack passwords has followed the expontential growth of Moore's law.