AuthKeys: A PKI Validation Tool for OpenSSH

Patrick Cable, Akshada Pol, Richard Rockelmann, Kriti Sharma, Peter Wilson
A final report for 4055.841: Advanced Computer Forensics
Professor Yin Pan

Description and Motivation

For this project we have chosen to develop a SSH Authorized Keys Validation Tool ("AuthKeys") that would verify a particular user's ability to access keys to a particular account on a specific machine. If the keys do not match an email will be sent to that Administrator of the system notifying them of a change in the file. Our motivation for the tool was to use concepts introduced in Advanced Computer Forensics, specifically hashing, in a tool geared towards System Administrators to enhance IT security and further knowledge.

Background and Tool Use Scenarios

OpenSSH is a utility built on OpenSSL that allows users to securely connect to remote servers and get a command prompt, or shell. OpenSSH has a variety of authentication methods - one could use one-time passwords, regular passwords, public key authentication, Kerberos, among others. This tool focuses specifically on public key authentication, and a specific problem: many times an individual - not a company or business unit - has physical access over the file that controls access to their account, and could use this access for malicious reasons.

In more detail, a user's "authorized_keys" file can be used to give - depending on how the private key was created - access to a system *without* a password. As a result, It is important to monitor this file for changes and analysis each time a user logs in, as changes to the file could allow a malicious to access a machine. We implement SHA and MD5 hashing to ensure that a user's authorized_keys file remains untouched by the end user - and alert the administrator

For example, let's imagine a particular service account on a particular machine has no password (and thus does not accept password-based login attempts), but requires a group of authorized users to have access to it. For this example, let's assume the account is used to run a Human Resources Information System that stores information about employee reviews and salaries. The account is setup to accept PKI-based logins from Alice, Bob, and Charlie. However, Charlie decides to add his friend Eve in Accounting to the system - simply by editing the "authorized_keys" file.

Had the file not have a mechanism of monitoring, the system would honor the request. However, the business would most certainly not have wished for Eve to have access. If Eve did have a need to know, then it's likely that Charlie would not have the authorization to perform such an action. This tool allows for an administrator to know that a change happened, and be able to perform more detailed forensic analysis.

Design, Implementation and Testing

In our design process for AuthKeys we wanted to create a relatively simple way in which we could verify that the SSH authorized_keys for a given user have not been tampered with. We decided that using a database in which to store the keys as well as MD5 and SHA1 hashes of those keys would be the best way to accomplish this along with some custom made scripts:

- **mysql_database.sql** - A MySQL database dump containing the underlying database structure for AuthKeys (authkeys_hashes, authusers, usercerts)
- **authkeys-config.pl** - A perl script that configures basic global variables for the AuthKeys toolset
- **adminauthkeys.pl** - Perl script that provides administrator management functionality for AuthKeys, granting the ability to add/remove/list users, and add/remove/list certificates for a particular user
- **makeauthkeys.pl** - Perl script that generates and stores the hashed SSH authorized_keys file in the MySQL database.
- **checkauthkeys.pl** - Perl script that verifies the integrity of the SSH authorized_keys file on login by comparing the MD5 and SHA1 hashes of the file against previously stored hashes within the database.

Implementation

To implement AuthKeys, one must:

1. Check out the current version of AuthKeys using git (git clone https://github.com/patcable/authkeys)
2. Copy the files into a globally accessible directory
    a. It is key to ensure that no one can read the password file.
3. Create a MySQL user and database to store the authkeys data
4. Import the MySQL Database (mysql -u user -p databasename < mysql_database.sql)
5. Store the password for the database in a file you define in authkeys-config.pl.
6. Add the appropriate keys to the database with adminauthkeys.pl
    a. For each user, run:
    adminauthkeys.pl --user [username] --addcert [path to public key]
7. Link accounts to their authorized users
    a. For each user, run:
    adminauthkeys.pl --user [username] --adduser [other user to access acct.]
8. Create the appropriate authorized users file for each user
    a. For each user, run:
    su -c user - makeauthkeys.pl
9. Test accordingly
    a. Ensure a user can ssh to the other user
    b. Ensure that when you change the $HOME/.ssh/authorized_keys file, the you can not log in.
10. Implement system-wide
    a. Add the checkauthkeys.pl file to all installed shell's system-wide shell init files.

Testing

The application was tested on a hosted virtual machine. We performed testing similar to what we would recommend an administrator use: simple expectations-based testing

Language and OS targeted
The code was written in in PERL. Realistically, the scripts could run on any operating system that needed to run OpenSSH and had a working PERL implementation. However, the scripts would have the most use on Linux/UNIX-based operating systems.

Limitations and Future Work
This is a prototype script, and we feel more use cases are necessary in determining its suitability in all environments. As it stands right now, the application logs the fact that the file has changed using integrity analysis and sends an email for each time that the file changes, however it may be more desirable to include more information on that email - specifically information provided by a unix stat() call on the file and time and date information to allow an administrator to dig deeper.

Integration of tools contained within the Forensics SleuthKit would provide more post-incident analysis and information about the file. We wanted to use forensic concepts in being able to tell an administrator that a particular issue was occurring, however.

Description of other similar tools (if available) and how your tool is different.

Other tools are a bit broader in reach with regard to what they control. For example, various configuration management platforms, such as puppet, cfengine, salt, bcfg2, among others, can ensure the consistency of a particular file; however they apply changes to ensure consistency, and don't raise a particular flag that the file has changed.

There are alternative solutions to file-based public key storage, specifically the OpenSSH-LPK extension, however it requires SSH to be recompiled - this is a risky operation as it could impact the integrity of the system if the recompiled sources are not kept in a way that only limited administrators have access to it.

Encase has a remote droplet that can be used to collect evidence, however it's not a way of managing a particular need on a regular basis. Encase is a tool with great use for post-incident analysis.

Conclusion
From our design and testing, we believe that we have created a very simple prototype application that verifies the integrity of a user's SSH authorized_keys. This tool was created as a means to demonstrate particular concepts used in computer forensics, and implement their use in a real-world way that would connect with other people outside of forensics professionals.

Appendix A: Source Code

Source code is browsable for AuthKeys at https://github.com/patcable/authkeys. A zip file is specfiically available at https://github.com/patcable/authkeys/archive/master.zip.