***Abstract*— *In this project, we will discuss the vulnerabilities of overlapping trust boundaries and explore how to protect against them. Overlapping trust boundaries become an issue when a cloud provider shares their IT resources with multiple customers. We will experiment by casting each member as cloud consumers and one member as the cloud provider. We will then ask the members to attempt to retrieve data belonging to another member. The cloud provider will also be tasked with retrieving data from the consumers as that is another possible vulnerability. The goal of this project is to design a system in which the cloud consumers cannot retrieve each other's data and the cloud provider cannot view the data.***

1. Introduction

Overlapping of trust boundaries involves one cloud provider and multiple different cloud consumers or users. With cloud services sometimes sharing resources among many different clients, they must take care of the boundaries that have been established between them. It provides attackers with opportunities to threaten IT resources that are shared between multiple cloud consumers. An attack commonly used is guest-hopping, this is where an attacker possibly has access to an Operating System (OS) and uses its access to compromise a different OS within the same cloud system. An ideal method for diminishing this threat is to provide traffic isolation.

Our solution would be to not allow other members of the group to be able to retrieve data from other members by adding multiple layers of security. With adding security, this allows the user who owns the directory will be the only one able to read from and write to the files in their directory. Each user will have the opportunity to do this using the chmod command.

We are using Amazon’s EC2 in order to host live Ubuntu instances configured with users for each member of the group.

In the context of the cloud, a trust boundary is a logical perimeter representing which users, resources, or organizations within a system are trusted. This often is used in the context of trusting the provider of cloud resources to have secured the system so that no other users can access private data. In this project, the trust boundaries are each of the users trusting the provider of the Ubuntu instance to secure the system to prevent access by other users. This means that the provider is inside of the trust boundary and the other users are outside of it.

The cloud here is representative of any computing or storage resources owned by an external organization that manages the privacy and security of the data contained. For this project, the Ubuntu provider is acting as the cloud service.

1. Related Work

Cloud computing gives many opportunities for large-scale computing projects at lower costs than if implemented by a single user [4]. However, as more resources are distributed it becomes difficult to track where different resources are used creating an issue of compliance with local regulation.

Data privacy is another concern because the data is often stored on machines owned by different organizations and if it is not secured properly can lead to breaches of privacy [4].

Security is problematic because it relies on all participants to be fully protected. If one organization is breached, the security of all others working together may be compromised. With more ways to access the data, it becomes increasingly likely that an attacker can find a vulnerability [4].

Pearson, Siani, and Azzedine Benameur offer a few suggestions as to how these challenges might be handled after a breach [4]. It is suggested to carefully define what organizations are in charge of what parts of the security and accountability in the case of breaches or data loss. However, these measures are reactive and do not actually answer how security will be addressed.

Clients do not have direct control over the cloud which leads to data privacy issues [5]. The users rely on cloud providers to protect their information from being stolen or the cloud providers from leaking their information. The cloud providers cryptography to protect the user's data. Cryptography alone cannot fix this issue. The users of the cloud services will need to rely on other forms of privacy enforcement to protect their data. Privacy preservation may be a key or a way to help protect the cloud provider’s clients’ information, but there may be other ways to protect the client’s information that has not been found or proposed.

1. Background

When multiple users use the same cloud provider, there is a risk for overlapping trust boundaries. This happens when a cloud provider shares multiple resources between different clients. This situation can become a potential opportunity for attack. Our goal was to recreate this scenario by setting up multiple users on the same cloud server and having them attempt to get to each other's data. Once we had measured the success of each user to find someone else’s data, we had them implement security measures to see what the new success rate was.

In order to design our system, we used an Amazon Elastic Compute Cloud (EC2). EC2 is a web service that provides secure and scalable computing capacity within Amazon Web Services (AWS). Amazon makes it simple to create an account and use multiple cloud resources for free. We were able to use this to host our operating system. For our OS, we opted for an Ubuntu instance. Ubuntu is a complete Linux operating system that is freely available to use.

Once the OS was up, each user was set up using Linux commands through the command-line. Instructions on how to do this were available on Amazon. In order to gain access to the server, each user had to use secure shell (SSH) and use their specific private key. SSH is a network protocol that gives users the ability to have a secure way to access a server over an unsecured network. For this to happen, all that was needed was knowledge of commands used through the terminal or command-line interface.

After the user had logged into their instance, It was time to focus on the security aspect. Each user first needed to change their permissions in order to make their directory private to them using terminal or command-line interface. Using SSH, the user also had the capability to add another layer of security by creating the jail. This means that the user their process and children form the rest of the system. Because none of the user’s processes were running as root, it made them much harder to break out of. Each user would now use their home directory as the root meaning only the user was able to read/write in their own home directory. This would help protect each user from overlapping trust boundaries.

1. Methods

For our cloud server, we decided that it should be accessible at any time and to prevent over complexity that we would only use features already available through the linux terminal. Unfortunately, this will mean that we will not get into the encrypted information that is stored on the cloud server nor how we will be protecting from outside threats. We only care about how secure we can make a simple cloud server from internal threats using minimal security measures. Using only features available through a linux terminal, means that we will not be allowed to install any tools or features for use in this project. By incorporating this rule we hope to show how secure we can make a linux server without the assistance of tools that do a lot for the user already.

1. Experiments

Before we can begin on the experiments, we need to set up an environment for us to work in. This environment will be set up on an Ubuntu instance using Amazon EC2 that everyone will be able to access.

* 1. Setup

Amazon has directions on how to add users to Linux instances [1]. First, is to add a new user account to the instance using “sudo adduser new\_user”. Second, we need to switch to the new user so that the files created later on have the correct permissions using “sudo su new\_user”. Next, we need to create a .ssh directory that only new\_user has read/write/open access to. This is accomplished using three commands in sequence, “cd” to go to new\_user’s home directory, “mkdir .ssh” to create the .ssh directory, and “chmod 700 .ssh” to set the permissions to the directory. Lastly, we need to create an authorized\_keys file in the .ssh directory. We use a string of commands to accomplish this, “cd” to go to new\_user’s home directory, “touch .ssh/authorized\_keys” to create the file, and “chmod 600 .ssh/authorized\_keys” to set the permissions on the file. This process will need to be repeated for every member of the team.

The next step is to allow everyone to ssh into the instance. To accomplish this we created a key pair for every member of the team and handed out the private key to the member it belongs to. Once the key pairs are made and downloaded we get the public key using the “ssh-keygen -y” command and give the private key file “New\_user.pem” when prompted. Once we had the public key we copy it from the terminal and paste it into the authorized\_keys file on the instance using vim. Once the file is saved, every member of the team will be able to ssh into the instance.

From this point, every member of the team has access to the Ubuntu instance and starts trying to access each other's data. We started with this level of security in order to understand what steps need to be taken to protect a user's data on a cloud platform starting with little to no security.

* 1. Adding Security

We encountered our first issue when one of the users decided to use the “cd ..” command to back out of their home directory. By doing so they could see other user accounts on the instance an navigate into them. To resolve this issue, the cloud provider logged into each user’s account and used the “chmod” command to set the user’s home directory to 700, meaning only the user that owned the account could read/write files into the directory [3]. By doing so, other users could not navigate into their directory and access their files.

We encountered our second issue when a user tried to change a root file. Since the members of the team do not have sudo privileges, this member could not change the file they were attempting to. However, they could change files that were not similarly protected. For this issue, we decided to implement a chroot jail. A chroot jail is a way of isolating a process and its children from the rest of the system [2]. Chroot jail will essentially make the system think that the user’s home directory is root-preventing users from navigating out of their home directory. The first step is to create a group for the users, “sshusers”, using the addgroup command, “addgroup -g sshusers”. After adding every user to the group using “usermod -a -G sshusers new\_user”, we need to emulate the root directory by copying the necessary folder from root to the user's new root folder along with other necessary files. This can be accomplished using this string of commands:

mkdir -p /home/{user}/{dev,etc,lib,lib64,usr,bin}

mkdir -p /home/{user}/usr/bin

chown root.root /home/{user}

mknod -m 666 /home/{user}/dev/null c 1 3

cd /home/{user}/etc

cp /etc/ld.so.cache ./

cp /etc/ld.so.conf ./  
 cp /etc/nsswitch.conf ./

cp /etc/hosts ./

Lastly, we need to configure sshd to chroot the users of the group sshusers. This can be done by adding the following script to the end of the sshd\_config file in etc/ssh/:

Match group sshusers  
 ChrootDirectory /home/%u  
 X11Forwarding no  
 AllowTcpForwarding no

After creating the chroot jail, users could not back out of their home directory. Since they cannot back out, they are unable to see the other users in “home” and cannot edit system files, other than the ones in their chroot directory.

1. Results

While initial security measures were able to be circumvented by users, each iteration reduced what unauthorized files a user could access. In the end, users were locked to their user accounts unable to even view the directories their home folders were nested within. This meant that they could not even identify what other users possessed accounts on the cloud.

Users were also left with a limited subset of the tools available to a standard installation which also helps to prevent unauthorized use of programs to view other users information. Being limited to the sshusers group means that other administrative users can be added without the limitations of cloud users. This being the case, other users and administrators can be added and removed relatively easily.

1. Conclusions

Overlapping trust boundaries will be an issue for a while, until there are other ways for cloud providers to provide all users access to the same resources without actually sharing the small instance or resource. As technology advances the cloud providers need to change how they protect users information without the user having to go and find other enforcement to protect themselves.

Based on the results, setting up a small cloud server with minimal security is possible, but not good from an industry standpoint. While most unauthorized access attempts were averted by setting up the chroot jail, a determined hacker could gain access through other means. This project is meant as an exploration into overlapping trust boundaries and what it takes to secure one at a basic level.

In the future it would be useful to streamline the setup by creating a simple batch script that can be used to create users. It is also possible to use a synchronization system to allow many virtual machines to be created and managed at once. This would allow different users to manage their own systems and create their own user accounts giving more detailed control of permissions and tools.

1. Appendix A
   1. Table 1

Table showing the progress of the project, issues that were found, and how they were resolved.

|  |  |  |
| --- | --- | --- |
| Attempt # | Issue Found | Resolution |
| 1 | Could get into other user’s home directories. | Added permissions using chmod so that only the user had access. |
| 2 | Could cd into the root directory and change files. | Used chroot jail forcing the users to navigate only as far back as their home directory. |
| 3 | Could run commands causing issues on the server. | Planned restriction of commands through the chroot jail process. |

1. Appendix B
   1. **Carmen Nicholson**
      1. Acted as one of the “Companies” storing data on the cloud.(25%)
      2. Helped work on the Background section before feedback (50%) and after feedback (100%)
      3. Assisted in writing the Abstract section. (15%)
      4. Helped with Related Works. (15%)
      5. Assisted in initial research of topic (30%)
      6. Helped write Appendix B (20%)
   2. **Chris Conway**
      1. Acted as the Cloud Provider and setup Cloud Server with Amazon EC2 (100%)
      2. Generated Private Key Pairs (100%)
      3. Fixed security breaches brought up by the “Companies” (75%)
      4. Wrote Methods Section (100%)
      5. Wrote Appendix A section (100%)
      6. Wrote majority of Experiments section (70%)
      7. Helped in writing the Abstract (20%) and Conclusion (33%) sections
      8. Formatted and turned in project assignments (75%)
      9. Supervised Team Meetings (75%)
   3. **Doug Lindsay**
      1. Acted as one of the “Companies” storing data on the cloud storage. (25%)
      2. Worked on Methods before feedback/revision. (40%)
      3. Helped clean up initial Works Related references. (40%)
      4. Helped write the Background. (50%)
      5. Helped in testing of the EC2 instances when they were first being set up. (25%)
      6. Helped ensure meetings happened as often as possible. (80%)
      7. Assisted in the initial search for the desired Cloud Provider tools. (65%)
   4. **Michael Towns**
      1. Acted as one of the “Companies” storing data on the cloud.(25%)
      2. Acted as the primary threat to other user's data in the cloud.(90%)
      3. Assisted in configuring security settings to prevent intrusion.(25%)
      4. Found articles (100%) for the related works section and helped writing sections of related works.(25%)
      5. Wrote future improvements section of the conclusion.(100%)
   5. **Yasmine Hines**
      1. Acted as one of the “Companies” storing data on the cloud. (25%)
      2. Helped with the Abstract Section (15%)
      3. Wrote Introduction Section. (100%)
      4. Helped write the Related Works Section (45%)
      5. Helped write the Conclusion Section (80%)
      6. Helped with getting meeting times set up. (80%)
      7. Helped write Appendix B (25%)

References

1. “Add New User Accounts with SSH Access to a Linux Instance.” Amazon, Amazon, 2017, <https://aws.amazon.com/premiumsupport/knowledge-center/new-user-accounts-linux-instance/>.
2. Feid, Allan. “ Creating a Chroot Jail for SSH Access.” *Allan Feid*, 2018, allanfeid.com/content/creating-chroot-jail-ssh-access.
3. Hope, Computer. “Linux Chmod Command Help and Examples.” *Computer Hope*, 29 Dec. 2017, [www.computerhope.com/unix/uchmod.htm](http://www.computerhope.com/unix/uchmod.htm).
4. Pearson, Siani, and Azzedine Benameur. "Privacy, security and trust issues arising from cloud computing." *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010.
5. Van Dijk, Marten, and Ari Juels. "On the impossibility of cryptography alone for privacy-preserving cloud computing." *HotSec* 10 (2010): 1-8.