CSCI 400 Security Lab
Topic: Access Control, Insiders, and that Man in the Middle

Prof. Dietrich

September 27, 2017

Group size: 4 — Setup needed: Windows/MacOS X/Linux, DETER.

1 General description

In this lab we are experimenting with forms of access control, as well as issues with insiders and the man in the middle. This lab is based on exercises by Peter Peterson and Peter Reiher.

2 File with samples and results

The files are in lab-access.zip. The files contain examples for limiting access to a service as far as resources are concerned. In some cases, you still may have to develop some scripts or write your own code to improve the results.

3 Overview

3.1 Access control for objects and subjects

In this part of the lab, we will be setting up policies for objects, such as files and resources, and subjects, such as users. The purpose of this part of the exercise is to introduce you to filesystem and network access control schemes and the "principle of least privilege" through the use of UNIX filesystem permissions.

For this portion of the lab, you will:

- understand the UNIX permissions structure including SUID and SGID bits
- understand the essence of the sudo utility and how to configure and use it securely.
- be able to apply that knowledge to configure permissions in multiple scenarios, such as:
- shared system directories
- user home directories and private directories
- privileged system directories

- unprivileged temporary directories
- editing important configuration files
- restarting system processes
- potential privilege escalation problems

3.2 Insiders and the Man in the Middle

The purpose of this exercise is to introduce you to the concepts of sniffing, insertion, replay, and Man-in-the-middle (MITM) attacks, and to give you opportunities to execute these attacks in a testbed. You should have a solid grasp of networking as from an undergraduate networking course, and be familiar with basic programming principles.

After this portion of the lab, you will:

- understand the basics of network sniffing
- understand the basics of replay attacks
- understand the basics of insertion attacks
- including the development of your own filters for Ettercap
- including common programmatic defenses against replay attacks
- understand the general concepts of man-in-the-middle attacks
- execute sniffing, replay, and insertion attacks against cleartext data
- execute MITM attacks against strong cryptography

3.3 Requirements

- You use the instructions provided with this lab. You need to use DETER. You have had the chance to get familiar with DETER with the warm-up exercise (Lab 0).
- Your output should demonstrate that you have done the experiments, such as screenshots, Unix typescripts, or logs, and the tar file generated by the `submit.sh` command.

4 Tasks

4.1 The FrobozzCo Permissions Test

You are the Security Administrator of FrobozzCo ("You name it, we make it"). You are looking to hire a new system administrator to replace the guy you just fired – unfortunately, even though his resume looked great on paper and he interviewed well, once you put him at a console he clearly had no idea what he was doing.

To him, proper permissions meant that the application worked; if it was secure then that was just icing on the cake. You should have done something when the other admins starting calling

him “Triple-7” – a.k.a. “wide open”. But it was when he changed the company fileserver directory permissions to `"ugo+rwx"` and the projected budget was “released” early – including a list of the employees being laid off – that he signed his own pink slip.

To keep this from happening again, you’ve created a little test for your new applicants to take. Unfortunately, before you can grade the applicants, you need to create an answer key.

Setup for creating the exam You’re going to create an answer key for your ”hiring exam.”

1. Follow the instructions in the **Introduction to DETER** document.
2. Log into DETER.
3. You’ll need to swap in your nodes and complete the exercises below. Create an instance of this exercise by following the instructions here, using

   `/share/education/PermissionsFirewalls_UCLA/permissions.ns`

   as your NS File.

   - In the ”Idle-Swap” field, enter ”1”. This tells DETER to swap your experiment out if it is idle for more than one hour.
   - In the ”Max. Duration” field, enter ”6”. This tells DETER to swap the experiment out after six hours.
4. Swap in your new lab.
5. After the experiment has finished swapping in, log in to the node via ssh.

As mentioned in DETER Instructions, changes to DETER nodes are lost when the nodes are swapped out. This means that you must manually save your work between working on nodes in order to keep it. However, this exercise includes experimental scripts to help you save and restore your work.

Saving your Work After completing the UNIX exercises on the `server` node, cd into the `/root` directory and execute the script `/root/submit.sh`; like this:

```
$ sudo ./submit.sh
```

... it will make a tarball of all the relevant files and their permissions, including the file `/root/permissions-answers.txt`. (The script will also back up and restore the `firewall.sh` script, which is used in a different exercise – you can disregard it.) **You must copy or move the created tarball into your home (or project group) directory, or it will be lost upon swapout.**

Restoring Your Work The experimental script `/root/restore.sh` on the `server` node, takes as input the path to a tarball created by `submit.sh` described above and restores the files to their proper locations on the system. This includes all the files you are asked to create or change in this exercise.

WARNING: These scripts do not back up all arbitrary changes you may have made to the node (e.g., changing a peripheral configuration file), and it does not "merge" system files with submission files – it only restores submission files copied by `submit.sh`. You shouldn't need to change anything else, but see `submit.sh` and `restore.sh` to see exactly what those scripts do, and do not delete any of your submission tarballs so that you can go back to an earlier version if need be.

To use the `restore.sh`, copy a tarball created by `submit.sh` into the `/root/` directory and execute this command:

```
$ ./restore.sh username-permissions-123123123.tar.gz
```

You will be asked if you want to automatically overwrite files, and if you want to selectively restore some files and not others. The options are self-explanatory.

Finally, if you don't trust the scripts, you can always make your own backups into your group directory and restore them by hand if you prefer.

NOTE: You do not need to run the submit and restore scripts with sudo. However, if you use sudo to run `submit.sh`, your tarball will be named after the root user. This is OK – just run `sudo ./restore.sh root-permissions-2134234243.tar.gz`.

Your task is to create an answer key for the following test:

This test is a part of the application process for the administrator position at FrobozzCo. You will be presented with a number of tasks and questions; the tasks must be executed on the server node of the DETER experiment we have created for this purpose (you can use the client node for testing), while the questions must be answered in a plain text file that will be submitted with the results of your tasks. See below for further instructions.

4.1.1 UNIX File Permissions and 'sudo'

Instructions The following section includes a number of permissions and file creation exercises. This test will be performed on a live network testbed with a full complement of standard utilities and editors. In addition, you are fully encouraged to use the Internet, man pages, help screens, and any other resources available to you in the execution and answering of these problems. Please read and use the disambiguation rules in the lab file for setting the correct permissions. Also, make sure that you test your answers – UNIX permissions are simple in theory, but in practice many combinations have counterintuitive effects!

Your answers will consist of an archive, created by the `submit.sh` script described above. The archive will contain:

1. A copy of the necessary server resources containing the results of your file creation and permissions modifications. You will be graded only on the correctness of your answers, not their elegance nor how efficiently you came to them.

2. Your written answers to the numbered short answer questions. These should be written in the file `/root/permissions-answers.txt` on the `server` system using an editor (such as `vim`).

See below for instructions on submitting your answers.

4.1.2 Home Directory Security

Note: Admins – members of the `wheel` group have sudo access. However, unless instructed to do so, use only standard UNIX ACLs – don't give user accounts `sudo` permissions or consider the `sudo` access the `'wheel'` group already has. Of course, you need to use `sudo` to create the accounts, edit root files, etc... this is exactly what `sudo` is for.

Your server needs two home directories, the usual `/home` and also `admins` for your team. Normal home directories are private, while the home directories in `/admins` will be publicly viewable and somewhat collaborative.

You can test the permissions settings on `server` by logging in as the various accounts you've created! For example, after creating the account `larry`, you can log in by executing `ssh larry@localhost` and entering the password you used for `larry`. In this way, you can see if the permissions you set meet the requirements.

1. Create the `admins` directory.
2. Create the groups `emp` and `wheel` (see `man groupadd`).
3. Create the user accounts `larry`, `moe`, and `curly` using `adduser`. You may set the passwords to anything you like.
4. Add the three accounts you just made to the `emp` group by editing `/etc/group` or using `usermod`. In our system, accounts in the `emp` group are "normal" (i.e., non-admin) accounts.
5. Next we will set up our administrators. Create the user accounts `ken`, `dmr`, and `bwk` – specifying that the home directory for each admin should be located at `/admins/username` – where username is `ken`, `dmr` or `bwk`. In other words, admin homedirs are not in `/home`. (See `man adduser` for special homedirectory options.) You may set the passwords to anything you like.
6. Add the admin accounts to the wheel group by editing `/etc/group` or using `usermod`. (Ensure that admins are not part of the `emp` group.) Being in the wheel group is what gives these users administrator rights (i.e., `sudo` powers).
7. add `larry`, `bwk`, and `dmr` to `ken`'s group.
8. add `moe`, `dmr`, and `ken` to `bwk`'s group.
9. add `curly`, `ken`, and `bwk` to `dmr`'s group.
10. On this system, default permissions for new home directories allow other users (i.e., users that are not the owner or in the specified group) to read and execute files in the directory. This is too permissive for us. Set the mode on the home directories in `/home` so that owner can read, write, and execute, group can read and execute and other has no permissions. (Set the mode on the homedir only – do not set it recursively.)
11. Individual home directories should now be inaccessible to other users. Now, set the permission mode on `/home` itself so that normal users can't list the contents of `/home` but can still access their home directories and so that members of the `wheel` group have full access to the directory (without using `sudo`).

12. By default, each homedir is owned by its user, and the homedir's group is set to the group named after the user. (For example, `ken`'s homedir is set to `ken:ken` – i.e., `ken` is the owner and the group is set to `ken`'s group.) Set the permission modes recursively on the individual homedirectories in `/admins` (see `man chmod`) so that:

    - owners have full access
    - group users (users who are in the group associated with a user's home directory) can read and create files in that homedir
    - other users can read and execute any files (unlike the home directories in `/home`)
    - files created by a group member in that homedir should be set to the homedir owner's group. (Hint: Look up what the SUID and SGID bits do on directories.)
    - Example: `larry` is in `ken`'s group. `larry` can create files in `ken`'s homedir, and those files are owned by `larry`, but are assigned to `ken`'s group rather than `larry`'s group. `moe`, not in `ken`'s group, can only read and execute files. (Note that permissions for normal `emp` user homedirs are not changed.)

4.1.3 The Ballot Box

All regular employees use this directory to submit votes for "employee of the month".

1. Create the `/ballots` directory.
2. Set the permissions on `/ballots` so that it is owned by root and users can write files into the directory but cannot list the contents of the directory.
3. Furthermore, set it so that members of the `wheel` group have no access (not including `sudo`).
4. Short Answer 1: Is there any way that employees can read the ballots of other users? If so, what could be done to stop this? If not, what prevents them from reading them?
5. Short Answer 2: What does the 'x' bit mean when applied to directories?

4.1.4 The TPS Reports Directory

Admin employees submit TPS reports in this partially collaborative directory.

1. Create the `/tpsreports` directory.
2. Create the `tps` user.
3. Set the permissions on `/tpsreports` so that it is owned by `tps` and that `tps` and members of the `wheel` group have full access to the directory, but so that no one else has access to the directory.
4. Furthermore, configure it so that files written into it are still owned by the wheel group, but so that one member of `wheel` cannot delete another member's files.
5. Short Answer 3: Which users on the system can delete arbitrary files in the `/tpsreports` directory?

6. Short Answer 4: Is there any way that non-wheel employees can read files in the `/tpsreports` directory?

7. Short Answer 5: What do '0' permissions mean for the owner of a directory? What privileges do they have over files in that directory?

4.1.5 sudo: Editing Configuration Files

For the following three short answer questions, assume that your answers are unrelated; that is, your answer for question 6 does not impact the answer for questions 7 or 8.

All members of the `wheel` group do system administration on the `server`. Because of this, they all have full `sudo` access to root privilege with the `/etc/sudoers` directive

```
'%wheel ALL=(ALL) NOPASSWD: ALL'
```

The "NOPASSWD" means they don't have to enter a passwd upon `sudo` invocation.

The employee `larry` is the webmaster for the system, so he has some extra privilege compared to the other employees. For example, `larry` has `sudo` access to the command

```
/usr/bin/vim /etc/apache2/apache2.conf
```

... with the directive

```
larry ALL=(ALL) /usr/bin/vim /etc/apache2/apache2.conf
```

... so that he can update the configuration of the webserver.

Short Answer 6: Is this safe? Why or why not? If it is not safe, is there a better way to give `larry` this access? If it is safe, how do you know it is safe? (Hint: search online for common `sudo` issues.)

4.1.6 sudo: Restarting System Processes

As a part of his webmaster duties, `larry` often requests the `wheel` group to restart the Apache server with the command `'/etc/init.d/apache2 restart'`. It would make everyone very happy if there was a secure way to let `larry` restart the Apache server (but not give him any other access).

Short Answer 7: Assuming the init script `/etc/init.d/apache2` has no unknown vulnerabilities, is it safe to grant `larry` `sudo` access to the command `/etc/init.d/apache2` restart? If this is not secure, explain why.

4.1.7 UNIX and sudo: Two Wrongs Make a Much Bigger Wrong

Carefully examine the permissions on the `server`. Look at `/etc/group`, `/etc/sudoers`, and the files in the `/admins` and `/home` directories (and their subdirectories).

Short Answer 8: Is there some way that `moe` or `curly` could subvert this system to gain root privileges? If not, how do you know this is true?

Hint: Consider what happens during the UNIX login process – the time between when the user enters the correct password and when they can interact with their shell.

4.2 Insiders, and that Man in the Middle

Introduction It turns out that the recent leak of some memos at FrobozzCo regarding imminent downsizing (due to insecure software and permissive filesystem ACLs) made the top management scared of malicious insiders. As a result, they've ordered a vulnerability analysis.

You've been hired at FrobozzCo ostensibly to replace one of the office assistants that has recently quit. None of the other employees know who you really are and think that you're just a temp. In reality, you've been hired by FrobozzCo's IT division to perform a Red Team exercise against a portion of the network.

Your assignment is to pretend that you are a disgruntled computer-savvy employee at FrobozzCo who recently discovered that your job is going to be eliminated. Since your job is over, you're supposed to steal as much information as you can and create as much chaos as possible; in other words the management wants to know how much damage a sufficiently motivated insider could do.

You have a desktop computer on the company LAN, which is a switched network. You share a network subnet with several computers that have some important responsibilities. The management fears that someone could do a little corporate espionage on their way out the door and take it to Zembor Corporation, FrobozzCo's biggest competitor.

WARNING: Red Team exercises are performed as a part of vulnerability analyses. However, they are only performed with express, written permission, non-disclosure agreements, and well-defined boundaries. Doing this without permission is illegal and carries severe legal penalties. So don't try this at home.

Performing this task and all subsequent tasks requires you to successfully perform an ARP Spoofing attack against the other nodes in your network using Ettercap. Other tools that might be useful for this task include `tcpdump` and `chaosreader`.

Experiment setup

1. If need be, read the **Introduction to DETER** document.
2. Log into DETER.
3. Create an instance of this exercise by following the instructions here, using

   `/share/education/MITM_UCLA/mitm.ns`

   as your NS File.

   - In the "Idle-Swap" field, enter "1". This tells DETER to swap your experiment out if it is idle for more than one hour.
   - In the "Max. Duration" field, enter "6". This tells DETER to swap the experiment out after six hours.
4. Swap in your new lab.
5. After the experiment has finished swapping in, log in to the node via `ssh`.

For this task, you will only have console access to the host eve.mitm.JJC400.deterlab.net; the hosts `alice` and `bob` will at be configured so that you can't log into them (at least to start!).

It is **very possible** that actions you take in the course of this lab could kill important processes, erase files, or generally cause other havoc in the system. If you think that you have "broken" your lab nodes, you can always reboot the nodes, or if things are really screwed up, swap out the experiment and swap it back in to start fresh. As always, anything you save in your group directory on DETER will stay there – make sure you save any important work somewhere safe.

4.2.1 Eavesdropping

Your first goal is to eavesdrop on all cleartext network traffic and document what the different traffic streams are communicating.

Warning! Do not ARP Poison any control interface (192.168.x.x)! Check which interface on attacker node has 10.x.x.x address and use that interface with -i option with ettercap, to make sure your poisoning stays on the experimental network.

Eavesdropping Tasks Answer these questions as completely as you can:

1. What kind of data is being transmitted in cleartext?
   - What ports, what protocols?
   - Can you extract identify any meaningful information from the data?
   - e.g., if a telnet session is active, what is happening in the session? If a file is being transferred, can you identify the data in the file?
   - Make sure you eavesdrop for at least 30 seconds to make sure you get a representative sample of the communication.
2. Is any authentication information being sent over the wire? e.g., usernames and passwords.
   - If so, what are they? What usernames and passwords can you discover?
   - Note: the username and password decoding in ettercap is not perfect – how else could you view plain text authentication?
3. Is any communication encrypted? What ports?

4.2.2 Replay Attack against the Stock Ticker

You know that stock information is communicated over your network and logged on an internal server. It takes a feed from downtown in the Financial District that updates the current price. The local server shows the current price and all previous prices since the last reload. This local server is the information that all internal company management uses for their reports and personal stock purchases. You also know that the stock program uses some kind of encryption, but you think it may be vulnerable to replay attacks because it was written by the CTO's 13-year-old nephew who doesn't have the greatest security track record.

Can you perform a replay attack against the stock ticker in order to make FrobozzCo's stock look bad and Zembor Corp's stock look good? You like imagining the internal chaos this would cause.

To do this, you will need to:

- set up ARP spoofing with Ettercap
- capture traffic with tcpdump
- analyze the captured traffic
- generate new requests to the stock quote service by "replaying" old requests.

You will probably find it easiest to create URL-based replay attacks using `elinks`, `wget`, or `curl` (all text-based web clients). Likewise, you'll probably want to set up port forwarding (see DETER instructions) so you can play with the application in your desktop web browser.

In this way, you are "replaying" the remote procedure call (RPC), but not the actual TCP packets. In reality, there is software that can be used to actually replay real TCP sessions back on to the network, but it is too complicated for this lab. (What about TCP packets makes replaying difficult?)

Replay Tasks Once you figure out how to create the replay attack, include the following information in a report:

1. Explain exactly how to execute the attack, including the specific RPCs you replayed.
2. Explain how you determined that this strategy would work.
3. Execute your replay attack and show the results of your attack with a screen capture, text dump, etc. showing that you are controlling the prices on the stock ticker.

4.2.3 Insertion Attack

`Ettercap` has the ability to execute regular expression-like filters on cleartext traffic. This allows you to change the contents of packets as you sniff them. To do this, you need to create `ettercap` filter modules and load them into a running `ettercap` that is performing ARP spoofing.

For this part of the lab, you'll write some `etterfilters`, compile them, and load them into `ettercap` so you can use them on the network stream.

`etterfilters` can be used to intercept and change traffic in either or both directions, but only between `bob` and `alice`. This means that you can change a request going to the server, and/or you can change the results that ticker users see when they view the ticker (by changing outgoing results). However, changing requests to the server is hard because of the nonce in the stock ticker request. This nonce hinders replay attacks and thus makes it difficult to correctly modify requests that are inbound to the ticker. However, but since the ticker replies in cleartext, it's easy to change outgoing results! Again, remember that requests from eve will not be modified – so "testing" from `eve` with `elinks` or a tunneled connection will not see the changes.

Insertion Tasks For these filters, you won't change the data that's on the stock server – just what it looks like to someone viewing the page.

1. You can change the symbols a viewer of the ticker sees by intercepting the HTML bound for their browser. Write a filter to change the symbol FZCO to OWND.

2. Write a filter to affect the prices a user of the stock ticker sees.

3. Include your filter sources with your submission materials.

Make sure you comment your code (use the # character) to explain what the filter does and how.

Warning! It's hard to test whether your filtering setup is working, because `ettercap` is only filtering traffic to and from `alice` and `bob`. If you run `elinks` on `eve`, or forward a port through `eve` with `ssh` tunneling, the traffic you see as a viewer will not be modified! You can run `tcpdump` on `eve` to see the modified packets leaving `eve`, but only the outgoing packets will be modified.

For these questions, you don't need to write filters, just write short answers or pseudocode explaining how you would do it.

1. Given the power of etterfilter and the kinds of traffic on this network, you can actually make significant changes to a machine or machines that you're not even logged in to. How?

2. Of the cleartext protocols in use, can you perform any other dirty tricks using insertion attacks? The more nasty and clever they are, the better.

Hint: There is a sample `etterfilter` source file in `/root/` on the host `eve`. You can use this filter as a starting point for your own filters. See `man etterfilter` and irongeek's etterfilter tutorial.

4.2.4 MITM vs. Encryption

There is at least one encrypted network stream in use on your network subnet. Lucky for you, your coworkers blindly click OK any time a certificate error pops up on their screen – you know this because they complain loudly every time it happens – so you are hopeful that a man-in-the-middle attack against the certificates would be successful. What's so important about that encrypted stream – what is the data being transferred?

`Ettercap` will not perform SSL decryption automatically – you have to uncomment a few configuration items in `/usr/local/etc/etter.conf` in order to properly forward the data. Examine that file, use any available documentation and make the necessary changes so that you can man-in-the-middle their SSL connection.

When you quit using Ettercap after having made these changes, it will make some complaints – don't worry about them.

MITM Tasks

1. What configuration elements did you have to change?

2. Copy and paste some of this data into a text file and include it in your submission materials.

3. Why doesn't it work to use tcpdump to capture this "decrypted" data?

4. For this exploit to work, it is necessary for users to blindly "click OK" without investigating the certificate issues. Why is this necessary?

5. What is the encrypted data they're hiding?

4.2.5 The Encryption Token

You have a powerful suspicion that the encryption token used in the stock ticker application is not particularly strong.

Encryption Token Tasks Answer one or more of the following to receive extra credit points.

1. What observable software behavior might lead you to believe this?
2. Can you reverse engineer the token? How is the token created?
3. If you can reverse engineer it, can you write a script in your favorite language to post data of your choice?
    - Hint: all the necessary pieces are available on the servers for both Perl and bash.
4. What would be a better token? How would you implement it on both the client and server side?

5 Deliverables

1. Submit the tarball created by `submit.sh` for the permissions task 4.1. You must use a tarball created by this script so that it will correctly save the permissions of the directories.
2. The answers from the MITM exercise, your etterfilter source files, and anything else you think should be included etc. Make sure to double check that you have answered all the questions.
3. A zip file containing:
    - The two reports from sections 4.1 and 4.2.
    - Any additional answers from 4.2.5.
4. Submit by the class on October 4, 2017.

6 Grading

Points will be subtracted if any of the pieces of the deliverables are missing or incomplete.