Introduction to Information Security Systems (CS-457)
Assignment 2

Assigned: 1/4
Tutorial: 1/4
Due: 14/5

Office Hours: 4/4, 8/4, 11/4, 15/4, 2/5, 6/5, 9/5, 13/5 14:00-16:00 at B208-B210-B212 and https://meet.jit.si/hy457

Assignment 2.1: Password Stealing

Introduction

In this assignment, you are asked to implement a script called *my_ssh.sh* which invokes *ssh* but at the same time "steals" the typed password. Use **strace** inside the *my_ssh.sh* to monitor all system calls and their arguments, find the password, steal it and store it.
Key ideas.

- Monitor system calls and their arguments . Use strace(1) to complete this step. e.g strace <*command* or *process*>
- Filter and parse monitored system calls and find which of them may contain the password. You are free to use any methods necessary.
- Store the password (not in plain sight).

Experiment

Important: For this part, locking home directory and spare is required.

Suppose that you managed to inject *my_ssh.sh* in a user's directory.
The .bashrc file is a script file that's executed when a user logs in. The file itself contains configurations for setting up or enabling coloring, completion, shell history, command aliases and more. Try to set your *my_ssh.sh* as an **alias** for the original ssh so that when the user calls *ssh* from the command line, *my_ssh.sh* is called instead and the credentials are stolen.

When this part is completed do not forget to remove ssh's alias.

Submission

You will submit only the *my_ssh.sh* script.
The experimental part won't be graded.

Hints and Links

- Parse the credentials by observing specific system calls
- strace(1) - Linux manual page - man7.org
- .bashrc is a hidden file
- After changing .bashrc activate changes using: $ source .bashrc

Assignment 2.2: Intrusion Detection System

Introduction

In this assignment, you need to implement a simple Intrusion Detection System (IDS). The IDS will analyze all the packets from a file with the help of the *pcap* library and generate alerts based on provided rules. The rule format should be the following:

<src IP address> <src port> <dst IP address> <dst port> "ALERT"

For example *192.168.1.2 55 192.168.1.7 55 "this is a rule for testing"*

Then, Generalize the above format so that the IP addresses can be provided in CIDR notation. For example: *192.0.2.0/24 123 192.168.1.5 123 "this is another rule"*

Develop your program using C. The IDS should be able to do the following::

1. Read a file named "alerts.txt" which will contain the alerts, each one in a separate line
2. Read the traffic from a .pcap file
3. Inspect the packets for matching rules and print the alerts

Submission

You will submit all the .c and .h files required for your intrusion detection system, a Makefile that builds the source tree as well as the .pcap and rule files you used for testing. Also, provide a README file that describes how the IDS works and how it should be executed.

Hints and Links

- https://www.tcpdump.org/ the pcap library

Assignment 2.3: Secret Sharing

Introduction

Three colleagues, Alice, Bob and Carol would like to find a novel way to share a secret number S. They would like each to take a piece of a number and only if all three pieces are presented then all of them will be able to reconstruct the secret number. One simple way is to divide the secret number in three pieces and each of them to take a piece. However, they found a more sophisticated solution.

They will create a polynomial: f(x) = a*x^2 + b*x + c, where c is the secret number S. Each one of them will take a point of the polynomial. Alice will take (1, f(1)), Bob will take (2, f(2)), and Carol will take (3, f(3)). Only when **all three** of them present their points they are able to find the polynomial f(x).

1. Write a program called *secret_sharing* using C, to share the secret number S. Using the "split" option and a secret number S (e.g. $ ./secret_sharing split 72), the program will generate the 3 points (1, f(1)), (2, f(2)), and (3, f(3)) and will write them in the standard output.

2. Extend the *secret_program* with the "join" option so that it recreates the secret number S given the 3 points (1, f(1)), (2, f(2)), and (3, f(3)) and writes it in the standard output. For example, $ ./secret_sharing join f(1) f(2) f(3) will output the secret number S.

3. Generalize your solution for N friends: each of the friends gets one point: (1, f(1)), (2, f(2)), (3, f(3)), …, (N, f(N)). When any three out of the N friends present their points they will be able to reconstruct the secret.

Submission

You will submit all the .c and .h files required for the *secret_sharing* program, a Makefile that builds the source tree as well as a README file that describes how *secret_sharing* works and how it should be executed.

Assignment 2.4: Anonymous Voting

Introduction

Alice, Bob, and Carol decided to use the above approach to implement anonymous voting. That is, they would like to vote on an issue, they would like to know the final result of the voting, but they do not want to reveal their individual votes. At the end they will all know the sum of the votes but not the individual votes.

They decided to proceed as follows:

1. Alice creates the polynomial A(x) = a2*x^2 + a1*x + a0 where a0 is her vote (an a0 value equal to 1 means “yes” and equal to zero means “no”). The coefficients a1 and a2 are known only to Alice.

2. Bob creates the polynomial B(x) = b2 x^2 + b1*x + b0 where b0 is his vote.

3. Carol creates the polynomial C(x) =c2*x^2 + c1*x + c0 where c0 is her vote.

4. Alice computes A(1) and gives A(2) to Bob and A(3) to Carol.

5. Bob computes B(2) and gives B(1) to Alice and B(3) to Carol.

6. Carol computes C(3) and gives C(1) to Alice and C(2) to Bob.

7. Alice computes D(1) = A(1) + B(1) + C(1).

8. Bob computes D(2) = A(2) + B(2) + C(2).

9. Carol computes D(3) = A(3) + B(3) + C(3).

10. Alice, Bob, and Carol contribute their points (1, D(1)), (2, D(2)), and (3, D(3)) to create polynomial D(x)= d2*x^2+d1*x+d0. Note that d0 is equal to a0+b0+c0: the sum of their votes.

Write a program called *anon_voting* using C that implements this anonymous voting. Generalize your program for N voters. You can choose how the program performs I/O and its command line options. Explain its operation and command line options in the README file. Also, explain why d0 is the sum of their votes.

Submission

You will submit all the .c and .h files required for the *anon_voting* program, a Makefile that builds the source tree as well as a README file that describes how *anon_voting* works and how it should be executed.

Notice

1. This is **NOT** a group assignment. Submitted code will be tested for plagiarism using plagiarism-detection software.You have to include your name and login in every file that you are going to submit.

2. Create a README file (30 lines max) that describes your implementation for each part of the assignment.

3. Place all the required .sh, .c, .h, Makefile and README files you created in separate folders based on the task named “2_1”, “2_2”, “2_3”, and “2_4”. Place all four folders in a directory and submit them using: $ turnin assignment_2@hy457 <folder_name>

4. You can use the course’s mailing list for questions. However, read the previous emails first since your question might have already been answered.

5. Avoid messaging the TAs directly, using their personal email addresses, since other students might have the same question and everyone deserves the answer.

6. Do not send source code snippets, containing parts of your implementation, to the mailing lists.

7. If you decide to use revision tools, such as git, mark the repositories as private.

8. This assignment should be implemented using shell scripts and C (based on each part’s requirements) on CSD’s Linux-based machines.