CS 577 Cybersecurity Lab

Lab 2 – due 9/18/14 6:15pm

Georgios Portokalidis – Stevens Institute of Technology

This lab will be done on the linux-lab.

You will need ssh, PuTTY, and a terminal to connect.

If you need to open multiple terminals to the same host for the exercise make sure you are indeed connected to the same host. linux-lab.cs.stevens.edu is just an alias that connects you to one out of many hosts. For instance, if after connecting to linux-lab you are in host1, make sure to connect to host1.srcit.stevens-tech.edu from the second terminal.

Exercise 1. PassCracker (60%)

Create a password cracker able to break passwords hashed with the MD5 hash function. Your cracker should use the following techniques:

Dictionary attack: Utilize a dictionary.

Brute-force: Generate all possible combinations of short 3-5 character long passwords.

Combination of the above**:** Combine Dictionary and Brute-force attacks. For instance, try to break passwords that include a word followed by a small number of digits (cybersecurity14).

Dictionaries can be found in: https://wiki.skullsecurity.org/Passwords

Keep in mind that in some cases you should add some heuristics, to find the correct password. E.g., try to crack passwords in leetspeak. That is, replace 'e' with '3', 'a' with '4' and so on. (http://en.wikipedia.org/wiki/Leet).

You are given two files with passwords, which you can use for testing your tool during development:

The 1st one has just hashed passwords.

hash = MD5(password)

The 2nd one has passwords hashed along after appending a salt with the value "id14".

hash = MD5(password||salt)

a) Your password cracker tool should takes a password file, salt, and time out (seconds) as arguments.

Are you able to crack any of the passwords in the files? [You will not be graded based on the number of passwords cracked, however it may help.]

How does salt make the cracking process harder?

In the two files can you tell if any users have the same passwords? How?

How much slower will you cracker run, if you try to brute-force longer passwords like 15 characters long. [You do not have to run your tool for longer passwords, just reason about it.]

Submit you source code (including makefile), and a short report of what you have done, answering the above questions. **Do not forget** to document your code when developing by using comments!

Exercise 2. Password hardening based on keystroke dynamics (40%)

Use keystroke dynamics as a 2nd factor authentication in order to prevent intruders that know your password access your account. Rely on machine learning, to create behavior profiles based on the way each user writes.

To do list:

- Create a program that **collects** keystrokes
  - Use a ***key listener*** in order to collect the pressed or released buttons.
  - For every event collect also the ***timestamp***
  - Save the keystrokes into a **txt** file

    For help refer to :
    http://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html

- Collect 20 times the keystrokes for the same password (e.g., ap2lv9!) for each team member.

- Create the behavior profiles (.arff files)
    - Store all keystrokes into one file (from all team members)
    - Add a new column into the file, named as ***Intruder***
    - Make a copy of this file, based on the number of your team members
    -
- Using **Weka** evaluate various classifiers and features performing experiments (10-fold cross validation).

  For Help refer to:

  http://www.cs.waikato.ac.nz/ml/weka/

  http://www.cs.waikato.ac.nz/ml/weka/documentation.html
- Report your findings (Accuracy/experiment)
- Use the features and the classifier that provides the best results for your profiles
- Create a **second** program that:
    - Asks for a username and password
    - Collect the keystrokes for your password
- Checks if the password was entered by the legit user by utilizing machine learning
    - Based on the username, load the corresponding behavior file in order to train your classifier

```
Instances train = new Instances(new FileReader("user1.arff"));
        train.setClassIndex(train.numAttributes()-1);
```

    - Select and train your classifier

```
IBk knn = new IBk();
knn.buildClassifier(train);
```

    - Evaluate the password

```
eval.evaluateModel(knn, test);
```

    - Get the results and various information about the classification

```
eval.toSummaryString();
knn.classifyInstance(test.instance(0));
```

- Perform experiments trying you team members to access with your account and discuss the results

- **Submit**:
    - The source code with comments and any needed documentation.
    - Report