CS4615 - P1 - AUTHENTICATION PRACTICAL

Overview

In the lectures we have discussed authentication methods and have looked at how Linux passwords are used and stored. In this practical we will use python (you could use a different language if preferred) to build our own authentication system and we will then try to break it in various ways. The aim is to understand how passwords can be stored safely and to get a feeling for how difficult or easy it is to break them.

PART1: Authentication Scenario

We assume a very simple system where we have exactly one user who is using a 4 digit Personal Authentication Number (PIN) as a password to log in. We assume here to use Linux in the labs with python 3.x. The whole system is represented as a single python file:

```
#!/usr/bin/python
import getpass

#our test system only has one user and one pin for it
s_user = "utz"
s_pin  = "1234"

#check if user and password match
def login( user, pin ):
        if user == s_user and s_pin == pin :
                print("user/PIN : " + user + "/" + pin)
                return True
        return False

#ask for the username and password
user = input("user: ")
pin  = getpass.getpass("PIN :")

#check if we can log in
if login(user, pin):
        print("access granted")
else :
        print("access denied")
```

The system stores internally the username as *s_user* and the PIN number as *s_pin*. In a real system these two pieces of information would sit in a database or a file (e.g. Windows registry or */etc/passwd* on Linux). When starting the program the user is asked to provide username and PIN. The PIN input is handled by the *getpass* library to prevent printing of the typed PIN in cleartext. Thereafter the function *login* is called to authenticate the user; within this function the username and PIN are compared with the stored *s_user* and the PIN number *s_pin*. When provided and stored information matches the user is authenticated. In a real system, the *login* function should be provided by an environment the user cannot manipulate (e.g. kernel of the OS).

PART2: Hash Function

It is obviously a bad idea to store the PIN in cleartext within the program (or in a file such as */etc/passwd*) as an attacker can simply see the PIN. A way to get around this issue is to store the PIN as a hash value.

Have a look at the python packet *hashlib*. The library provides an MD5 hash function which can be used to transform the PIN into a hash. Modify the program to store a hash instead of the PIN (i.e. *s_hash*) and use this for the login procedure. Use *hexdigest()* to avoid issues with non printable characters.

PART3: PIN Guessing

Now modify the program such that the user is not asked to input the PIN, instead write a routine which tries all possible PIN combinations. For each PIN the hash must be calculated and then provided to *login* for checking.

Measure the time it takes for your routine to check all possible PIN combinations. You can use the python library *datetime* an the following code to measure execution time in milliseconds.

```
...
import datetime
...
def millis_interval(start, end):
        diff = end - start
        millis = diff.days * 24 * 60 * 60 * 1000
        millis += diff.seconds * 1000
        millis += diff.microseconds / 1000
        return str(millis)
 ...
 start = datetime.datetime.now()
 ...
<some code>
 ...
 stop = datetime.datetime.now()
 print ("Time to exec code: " + millis_interval(start, stop) + " ms")
```

How long does it take to test all possible combinations for a 3 digit PIN, a 4 digit PIN, 5 digit PIN, ... Plot a graph and look at the relationship between time and number of digits.

PART4: Preventing PIN Guessing

To prevent fast guessing of the PIN something needs to be done. We can make the hash function more computationally expensive. If it takes more effort to compute the hash, the time it takes to try PIN combinations increases.

Replace the hash function by a recursive hash function calling itself 10000 times: $h = H_{MD5}(H_{MD5}(...H_{MD5}(PIN)))$. Now, see how long your program takes now to find the right PIN number. Again, plot a graph and look at the relationship between time and number of digits and compare this to the results in PART3.

PART5: Better PIN Guessing

To improve the guessing we can change our approach. Instead of computing all hash values on the fly we can write a program that writes all these in a file. Thus, we can prepare a file containing all possible hash values which we can then use to determine the PIN. It will take quite some time to compute the hash values for the file; however, once we have the file we can just read hash values out of a file without the need of costly hash calculations.

Write a program to create a hash table for a 4 digit PIN. Then, modify your program such that all hash values are read from the file instead of computing them on the fly.

Measure the time it takes to generate the file with hash table. Measure the time it takes to break a 4 digit code using the generated file. Compare these timings to the timing in PART4. Look at the file size of the hash table file.

PART6: Preventing Dictionary Attacks

To prevent the attack in PART5 we can use a salt. Modify the login procedure such that the user is identified using username, salt and hash. Let's assume the salt is as well a 4 digit number.

To now run an attack using precomputed hash values we need a precomputed table for every salt value that there is. What would this mean for the required time to compute all tables and for the storage requirements?

PART7: Using a Library

The library *hashlib* provides the function *pbkdf2_hmac()*. Investigate how this function works and how it could be used to replace the authentication procedure you have implemented in PART6. You will see that the parameter provided by this function relate to the issues we looked at in PART1 to PART6.

CS4615 Continuous Assessment - PART 1

Please submit an answer to the following question with your CS4615 Continuous Assessment. Your answer should not be longer than half a page (You can use figures or code pieces to illustrate your answer).

Question P1 [2 MARKS]: PIN Guessing

Assume the test of one combination takes 1ms. How long does it take to test all possible combinations for a 3 digit PIN, a 4 digit PIN, 5 digit PIN and 6 digit PIN. Plot a graph to show the relationship between time and number of digits. Can you give a formula describing the time to test all combinations in dependence of the number of digits N.