AI224: OPERATING SYSTEMS

Lab 6

In information technology security, authentication is a security service that allows a party to prove its identity to another party (one-way authentication). To that end, it may use a password.

In this lab, we assume that a challenge-response-based authentication protocol is being adopted by two processes, a client —Alice — denoted as P_A, and a server process — Bob — denoted as P_B. Process P_A proves to P_B that it is the real P_A by proving the possession of the correct password. To start the authentication protocol, process P_B sends a challenge c (i.e., a random string) to process P_A. Upon the reception of the challenge, the latter concatenates the challenge c with a secret password p (that is known to P_B) to create a response $r = Hash(p||c)$, where $Hash(.)$ is a hash function (i.e., a function that takes as in input a sequence of bytes and outputs a unique value out of that). Then, process P_A sends back the computed response r to process P_s. The process P_B performs the same computation by computing $r' = Hash(p||c)$ and checking whether $r' = r$. In the latter case, process P_B authenticates process P_A. Otherwise, if $r' \neq r$, process P_B aborts the authentication process with P_A.

An attacker — Eve, represented by a process P_E, wants to crack the password to impersonate P_A (Alice) later on. It eavesdrops on the communication between P_A and P_B during their authentication and captures the challenge string c that is sent by process P_B to process P_A as well as the response $r = Hash(p||c)$ sent by process P_A to P_B. Furthermore, Eve, via another source, learns that the password p that was used by Alice is a 7-character word that uses only lower-case alphabetical letters $(a, b, c, \ldots, z)$ and starts with letter a, e, or o. The attacker also learns that the hashing function that was used is `MD5` 128 bits (Message Digest 5 — 128 bits).

The attacker implemented as a multi-threaded process (composed of multiple threads), performs a brute force attack (i.e., tries all possible combinations of letters) to crack the password p. To that end, and to speed up the cracking process, the attacker creates three threads t_1, t_2 and t_3. Each thread has access to the challenge c and the response r. The attacker process (main thread) starts all children threads (t_1, t_2 and t_3) together and assigns to each thread a character to work on, i.e., a, e, or o. In this way, each thread t_i takes the challenge c, fixes the first letter to a, e, or o (depending on the assigned value by the main process), constructs a 7-character password p', computes the response r', and compares the result r' with the real (captured) response r. If they are equal (i.e., $r' = r$), then the thread stops its execution and displays the password p'. At the same time, the other remaining threads, through a shared variable, which they check each time, get notified that one of the threads has cracked the password. This makes them stop as well.

We want to implement this attack using Java threads. The attacker code is given in `ThreadMain.java` and the attacker's threads class definition is provided in `ThreadChild.java`. The attacker process knows the following information: the used challenge (`challenge`), the captured response (`captured`), and the used hash function algorithm (`MD5-128`).

1. *Complete the thread class definition to crack the password. What is the password? How long it took to crack it?*
2. *Demonstrate that the time it takes to crack the password using a multi-threaded process is less than the time it takes for the attacker who uses a single-threaded process if we consider the same order of testing the first characters, i.e., a, e, then o.*

ANNEX

To compute the MD5 hash for a given string *str*, use the following code:

```
md5.update(str.getBytes(), 0, str.length());
String hv = new BigInteger(1, md5.digest()).toString(16);
```