Computer Assignment 4

Deadline: Mar 11, 2024, 11:59 PM

Winter 2024

(Upload it to Gradescope.)

Project Description

The goal of this assignment is to learn the basics of software-based remote attestation. You have to implement a simple remote attestation protocol, an attack scenario, and introduce noisy communication to study the impact of network and processing noise.

Design

Create a Jupyter notebook and do the following. You need to create a verifier function and a prover function.

1. **Verifier:** Write a verifier function that creates a nonce, sends the nonce to the prover function, and invokes it (details later). Once the response is received, it locally computes the hash itself and compares it with the response. It passes the test if the two hashes are equal.
2. **Prover:** It takes the nonce, reads the memory line-by-line, and computes the hash. It returns the final hash value.

Notes:

a. The nonce should be a random number between 0-256. Make sure that it doesn't repeat if you are doing multiple attestations (verifier's responsibility).
b. Use hashlib sha256 function to compute the hash (for both verifier and prover). You need to only send the final hash thus use the hash of the previous line as the input of the new line along with the new content of the memory – i.e., newhash = sha256(previous hash, new line). Repeat this for all lines.
c. You don't need to read the actual content of the memory instead, use these files (link) to mimic the content (use only "random_numbers_16" file). Assume that your memory has 1600 lines and each line is a number between 0-255.

Steps

Once you have created and tested the two functions, do the following:

1. Measure the time it takes to complete an attestation. Use a timer and start it after creating the nonce and stop it once the prover function returns its final hash (i.e., measure how long it takes to compute the hash in prover). To find the average time, repeat this process 10 times and report the average.
2. Create an attack as follows:

   a. Create a pre-compute function that does the following: it combines the “zeros_8” and “attack_8” files by copying the non-zero parts of the former into the zero parts of the latter.
   b. Use the new file, write a malicous_prover function that takes the nonce and computes the correct hash for the zeros_8 file but only using the attack_8 file (note that you can assume that the second half of the file should have been zero, and use that to forward the correct content and compute the hash).
3. Once you confirm that the attack passes the test successfully, measure the time for the malicous_prover function. Report the average (10 runs).
4. Assuming that the times are slightly different, modify the verifier function to check the response time and only accept the response only if the hash is correct and the response time is less than a threshold. Explain how you find that threshold.
5. Repeat this process 10 times, 5 with the correct verifier and 5 with the malicious, and report the true and false positive rates.
6. Add a random noise to the timer (to mimic the communication and computation randomness). Report the amount of noise required to reduce the true positive rate to 50%.
7. Explain how one can improve the true positive rate in the presence of such a noise.

What to submit:

Submit your notebook in PDF format. Your PDF file should show your well-commented code and the result for each step. Add a description at the end, briefly describing your algorithm, and answer the question in step 7.

Good luck!