|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| We declare that we have completed this assignment in accordance with the UAB Academic Integrity Code and the UAB CS Honor Code. We have read the UAB Academic Integrity Code and understand that any breach of the Code may result in severe penalties.  We also declare that the following percentage distribution faithfully represents individual group members’ contributions to the completion of the assignment | | | | |
| Name | Overall Contribution (%) | Major work items completed by me | Signature / initials | Date |
| Tiara King | 33 | Part 1, Part 2, Report | TK | 3/26/2023 |
| Harshith Honnaganahalli Kadaiah | 33 | Part 1(Merkle Hash Tree & Report) | HK | 3/26/2023 |
| Vinay Maddiralla | 33 | Report, Debug code | VM | 3/26/2023 |

GA3: Immutable Records

March 26, 2023

Group 14

Table of Contents

Overview……………………………………………………………………………...…………..1

How to run………………………………………………………………………………………...1

Design and Implementation……………………………………………………………………….1

Screenshots……………………………………………………………………………………...1-2

Immutability……………………………………………………………………………………….2

Comparison………………………………………………………………………………………..2

Vulnerabilities……………………………………………………………………………………..2

Pros and Cons of shortened time intervals……………………………………………………...2-3

Future Improvements……………………………………………………………………………..3

Overview

The goal of this assignment was to implement a system in which you can use immutable records and track data on a public platform immutably. The purpose was to build a track of records that could not be altered, and in the case that they were altered then you would be able to easily detect that some alterations have been made.

How to Run

1. Add files to the main project folder.
2. Name the folders with the name starting in Day (this is something that we would like to improve in further implementations)
3. Run the ReadFiles.py file.
4. Output of hash values will be displayed in the terminal.

Design and Implementation

* The assignment is implemented using three files HashData.py, MerkleHash.py and ReadFiles.py
* HashData.py takes each file as an input and performs SHA256 using hashlib module and prints the 256 bit hash.
* MerkleHash.py uses the hash outputs from HashData.py and performs recursive hashing for all the file hashes in the folder and gives one hash as Merkle Root Hash for entire snapshot.
* ReadFiles.py imports the scripts HashData.py and MerkleHash.py to generate hash values for each given file and finally to apply the Merkle Tree algorithm on the generated list of hash values. This file also signs the hash with the time to make it one output to improve immutability.

Screenshots –

Text

Description automatically generated

This screenshot is the output of each file being hashed and the hash for the day. The last value is the chain of hash values which form the immutable record.

Text

Description automatically generated

Continuation of above screen shot.

Text

Description automatically generated

Screenshot of data with signing



Continuation of above screenshot

Immutability

The system that we have built is immutable because if you modify any of the files that are in the directories then you will no longer receive the same hash values. If the hash values are not the same, then you will know that contents have been altered. With the addition of signing the data, it adds another layer of protection from the data since it attaches a timestamp along with the signature.

Comparison

The usage of time stamps along with the hash to sign in the second part using RSA algorithm helps against any modifications of the data. In the first part since we are only using SHA256, it would be easier for an attacker to modify the data, especially since those hash values were published in a public place. With the addition of the RSA signing, along with timestamps, it gives an attacker less of an advantage, because changing the time stamp will change the hash and it will be more easily detected.

Vulnerabilities

If the attacker can make changes to the system and the hashes as well which cannot be detected, the system will be vulnerable. There is a high possibility of brute-force attacks if the public key used is weak or not stored properly. It's also vulnerable to replay attacks, where the attacker can intercept the time-signature pair and republish it, making it appear the data hasn't changed.

Pros and Cons of Shortened Time Interval

Implementing the system with shortened time intervals between each snapshot benefits the system in detecting even the slightest of changes made, therefore increasing the overall security of the system, and providing a more granular and up-to-date information. The drawback being high usage of system resources such as time, memory and storage and publishing the same time-signature pair multiple times can increase the risk of replay attacks, since there's a higher chance of intercepting and reusing.

Future Improvements

1. Be able to accept files no matter the name
2. Be able to accept nested directories
3. Update the merklehashfile to be a tree rather than a list