Project: Dropbox

Design (Part I) Due: Friday, April 15, 2022 @ 11:59 pm ET
Implementation (Part II) Due: Tuesday, May 3, 2022 @ 11:59 pm ET

Reminder: Late days cannot be used on Part I: Design.

0 Introduction

After you broke into almost every single technical system at the Crewmate Academy, the Information Technology office figured that the only way to prevent vulnerabilities in their newest project (a file sharing service for inter-departmental collaboration) was to hire you as the lead developer.

In this project, you will implement a secure file storage system. Most importantly, you will implement it *securely*. This project will give you experience not only with writing secure software, but equally importantly, experience with *designing*, *planning*, and *critically analyzing* secure software—carefully thinking, threat modeling, and critiquing your own design before you write any code.

This project consists of two components: Part I, where teams will write a detailed design plan for their system and Part II, where teams will implement their Dropbox systems in Python.

Because of the nature of this project, there are some specific guidelines for approaching TA at Project Hours about Dropbox. Because the project is meant to be evaluative of your own security analysis skills (and since TAs only have a limited amount of time to help at hours), please know the following:

- When possible, teams should attend Project Hours together. If one member of a team is seen during a block of hours, the other member will be removed from the queue (if applicable).
- During Project Hours, we won't be able to do large-scale debugging of your code, though we can answer general questions about Python and the stencil code. If you have small-scoped debugging issues, we recommend that you post them on Ed, where the TAs have more time to do research into your question.
- During Project Hours *or on Ed*, we won't be able to *validate* your entire design or think through all possible cases that aspects of your design should cover. Instead, we ask that you show up with specific questions and scenarios (For instance, how can I test for specific scenario X? Or, does technique Y prevent attack Z?) Part of the goals of this project are to demonstrate the complexity and challenge of analyzing secure systems, even for the TAs—as such, the TAs may not be able to give you definitive answers as to the security of your whole system, etc. (We'll certainly try to help you think through these issues, though!)

1 Specification

Heads up! The section below is a quick introduction to the project, but the majority of the specification can be found at the CS166 Dropbox Wiki at `http://cs.brown.edu/courses/csci1660/dropbox-wiki/`. The online documentation is the canonical source of information on this project, so make sure you read the wiki to learn about the main requirements.

1.1 Motivation

When you're developing software, you naturally trust the systems that you are working on—whether it's your computer, the VM that runs your CS166 machine, the Computer Science department machines, etc—to not act *maliciously*. For instance, when you save files on your local computer or upload a file to, say, Google Drive, you trust that there aren't any attackers on your machine, that your hard drive won't start flipping random bits, and so on.

In the real world, these kinds of trust assumptions may not be reasonable or valid. In industry, where companies delegate computation to third-party resources all the time, trusting in one of those companies implies trust in the third-party resources. However, if you're dealing with particularly sensitive information, it may be unreasonable to keep expanding your circle of trust.

One solution is to simply avoid outsourcing any resources and maintain physical and technical control over every system you use. However, this may not be cost-efficient or practical. For example, it may not be feasible for a company to maintain physical control over a secured data center, and so that company might outsource their cloud storage to a third-party provider. However, it may be feasible for that company to access a small, secure amount of trusted storage space, and somehow devise a way to combine both types of trusted and untrusted storage to create a overall secure system. In this project, you will architect a secure system that does exactly this.

1.2 Assignment

You will write a client for a file storage service. The client must implement eight operations—`CreateUser`, `AuthenticateUser`, `UploadFile`, `DownloadFile`, `AppendFile`, `ShareFile`, `ReceiveFile`, `RevokeFile`.

Users of the client will provide their username and password to authenticate themselves to the service. Once authenticated, users will use the client to upload new files (and identify them with a filename chosen by the user) and download previously uploaded files from the server. Users may also modify the contents of their files by uploading a file with the same name as another file previously uploaded. Users will also use the client to *share* files with other users (as well as revoke permissions from previously shared files) and download files shared to them by other users.

To implement this functionality, your client will have access to two servers: `dataserver`, which is an *untrusted* data storage server which can store arbitrary data; and `keyserver`, which is a *trusted* public key server.

Using only these two servers, you will implement the functions above in such a way that your client ensures *confidentiality* and *integrity* of files that are stored on the server.

Finally, users are not guaranteed to be online between invocations of calls (nor are they guaranteed to be using the same machine!), and thus your client must be *stateless*. This means that your client may not rely on local storage or global variables to provide its security guarantees—if the client is restarted, it must be able to pick up where it left off given only a username and password.

1.2.1 CS162 Requirement

CS162 students must satisfy an additional performance requirement on `UploadFile` which allows for *efficient updates*. This requirement is detailed in the wiki.

Part I

Design

2 Assignment

Critical parts of creating real-world, secure software are careful, thoughtful design and crisp, precise, written analysis of the security of your system. This first phase reinforces this notion—in this phase, you write a detailed design document for your client implementation.

There are intentionally many elements of the client whose implementation is highly-dependent on your overall design. This project is designed to give you a chance to do some critical thinking about security design in a broad sense—we're not just trying to test your ability to pick a strong hash function or avoid path escaping vulnerabilities. We encourage you to do lots of brainstorming with your partner, and consider many possible designs.

If you do the design phase right, you will likely find writing your design document the most difficult aspect of this project—implementation might take longer, but it will be much easier if you're confident in your outlined design.

2.1 Deliverables

Your design document must include the following sections:

- *System overview.* Summarize the design of your client (maximum 4 pages, plus an optional 2 pages of diagrams).
  - Any major design choices should be explained in such a way that a fellow CS166 student could implement a client that achieves a grade similar to your client just by reading your design document. (Clear, concise, precise writing is the goal here.) At a minimum, you should address:
    - How users are "authenticated"
    - How files are stored on the `dataserver`
    - How your design allows `AppendFile` to meet its efficiency requirements
    - *(CS162 only)* How your design allows `UploadFile` to meet its efficiency requirements
    - How files are shared with other users (that is, how `ShareFile` and `ReceiveFile` work)
    - How previously shared files are revoked, including revocation by the root user of direct subtrees in the sharing tree
  - We encourage you to include system diagrams if it makes sense for your design.
- *Security analysis.* Present *at least four* and *at most five* concrete attacks that an adversary may conduct against the system and explain how your specific design protects against each attack. You will be graded on the four analyses which provide you the most credit—and a particularly good set of *five* analyses may earn a small amount of extra credit at the final deadline.
  - You should make sure your attacks cover different aspects of your system design. (That is, don't provide four attacks that all concern file storage, but no attacks involving sharing or revocation.)
  - Do not include any of the following kinds of attacks:
    - Breach of confidentiality of *unencrypted* data
    - Breach of integrity of *unauthenticated* data
    - Attacks involving the leakage of the length of a filename

- *Example:* "After user A shares a file with user B and later revokes access, user B may try to call `ReceiveFile` again to regain access to the file. To prevent a revoked user from regaining access this way, our design..." This analysis is satisfactory because it describes a concrete attack that can be derived from the provided security definitions and function specification. *(To be clear, your analysis may not include this example described here.)*
- The attacks you describe must have a security consequence and cannot simply be a bug. That is, your proposed attacks must result in an attacker breaking confidentiality or integrity guarantees or executing an unprivileged action.

You do not need to describe how you will implement any networking-related components (such as how file data is transferred from client-to-server), since those are outside the scope of the project and are abstracted away in the support code.

Additionally, your design document does not need to be formal (that is, you may use bullet points when describing your service's design).

2.2 Grading

You will submit your design document twice. The initial draft of your design document is due at the Design due date (Friday, April 15, 2022 @ 11:59 pm ET) and will be graded on *completion* of each of the sections. (While you will not receive any written feedback at the completion checkpoint, the completion checkpoint is worth 5% of the final grade on this project.)

By the Implementation due date (Tuesday, May 3, 2022 @ 11:59 pm ET), you must update your Design document to reflect any changes in your implementation design and submit your document again—this version of the document will be much more closely graded, and is worth an *estimated* 15% of the final grade on this project. (This weight is subject to change.)

2.3 Handing In

You should hand in your design document as a PDF on Gradescope. Only **one** partner should hand in the PDF—you must use the team selection dropdown in Gradescope to select your partner's name when you hand in. (A deduction may be applied if more than one partner from a team hands in or the team is not correctly selected on Gradescope.)

At the top of all of the pages of your document (i.e. in the header), you should *clearly mark whether or not the team includes at least one CS162 student* (which means you must complete the CS162 requirements for the design document).

Part II

Implementation

3 Assignment

To translate your theoretical design work to practice, you will implement your client design in Python. The CS166 Dropbox Wiki details most of the technical requirements for this part of the project, but here we detail some logistics about the implementation.

3.1 Evaluating Your Implementation

We will test your client application with a series of functionality and security tests to determine your code score. Due to the nature of computer security, most of the tests are hidden. Some test results are available before the deadline, and others will only be available after the deadline.

Each failed code test will incur a multiplicative penalty on your score. This means that each subsequent failed test has less impact on your grade. This emulates an environment where the existence of a vulnerability is more important than the exact scope of the vulnerability.

The autograder will run multiple fixed random seeds for each test. Because your code must be robust to failures, we will take the lowest score across these multiple runs as the score for your code. Thus, any nondeterminism introduced by your code will be penalized.

3.2 Test Coverage

You must write tests for your client application in `test.py` in the stencil code. Your tests should verify correct functionality of the client, correct handling of erroneous inputs, and any security problems. Each test should be defined in a separate function.

Several basic functionality tests are already defined in `test.py`. Make sure that your implementation passes these basic functionality tests. An implementation that fails these basic tests will receive zero credit for the testing portion of the total grade.

Part of your testing score is determined by code coverage (the amount of the program that is exercised in your tests). Your tests will be run against the staff implementation of the client application. Many lines in the staff implementation are instrumented so that its execution will be a point towards your test score. A comprehensive test suite will have better coverage and will increase your score.

3.3 Grading

Your final Python client implementation is worth approximately 65% of the final grade for this project. Your test coverage is worth approximately 15% of the final grade for this project. (These weights are subject to change.)

3.4 Handing In

You will hand in your implementation code and testing code on Gradescope. Only **one** partner should hand in—you must use the team selection dropdown in Gradescope to select your partner's name when you hand in. (A deduction may be applied if more than one partner from a team hands in or the team is not correctly selected on Gradescope.)

Instructions on how to hand in your code are documented on the wiki.