CS 1653: Applied Cryptography and Network Security
Fall 2023

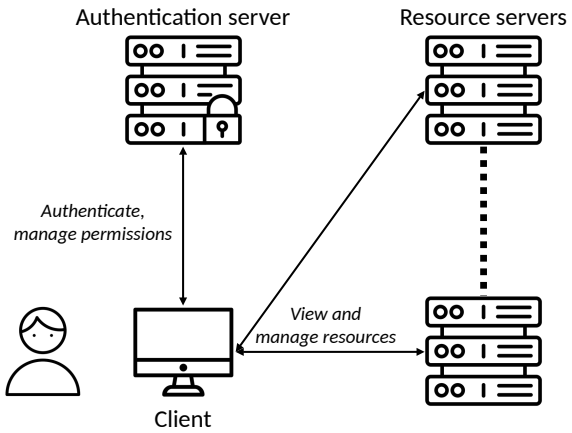
Term Project, Phase 2

Assigned: Tue, Sep 26 **Due:** Tue, Oct 17, 11:59 PM

---

1 Background

Over the course of this semester, you will apply the security concepts that are covered in lecture by developing a distributed (networked) system of your own design and securing it against a number of different types of security threats. In this phase of the project, you will implement a bare-bones system that provides the necessary functionality but with *very few* security features. Recall that the system will consist of three main components: a single authentication server (AS), a collection of resource servers (RSs), and some number of client applications.



The *authentication server* will manage the users in the system and keep track of any metadata about each user. Any number of *resource servers* can be deployed throughout the network (without prior approval from the authentication server), and will rely on the AS to provide each legitimate user with an authentication and authorization *token* that answers the question, *"Who are you, and what are you permitted to do?"* Users within the system make use of a networked *client application* to log in to the system, manage their user profile (via the AS), as well as upload/post, download/view, modify, delete, etc., resources that are stored in the system (via one or more RSs).

Your project submissions will be made using the git distributed version control system. You are encouraged to use this system to help coordinate your group work, and follow the git best practice: "commit early and often."

2 Trust Model

In this phase of the project, you should focus on implementing the *core functionality* of the service, and will defer the implementation of most *security features* to later phases of the project. The reason for this is twofold. From a pragmatic perspective, you have not yet learned all of the security "tricks and tools" that you will need to properly secure this type of application. Furthermore, having a fully-functional system in hand may simplify the later phases of the project, as you will only be debugging security features, not security features *and* core functionality. (Note that our high-level system model already takes into account the security goals you will try to achieve later. Thus, this does not fall into the category of "retrofitting security as an afterthought" that we have discussed in lecture, at least not to the same degree.) When implementing this phase of the project, you may make the following assumptions about the participants in the system.

- **Authentication Server** The AS is entirely trustworthy. In this phase of the project, this means that the AS will always behave *exactly* as you specify in the technical specification.
- **Resource Servers** The RSs are all entirely trusted. In this phase of the project, this means that each RS will behave *exactly* as you specify in the technical specification. For example, this means that you need not worry about a RS modifying a user's token, making unauthorized use of a user's token (e.g., using the token at another RS or disclosing the token to another client), or leaking the resources to non-members.
- **Clients** All clients will behave in a trustworthy manner but may be *curious*. This means that clients will not share their tokens with one another, nor will they make any attempt to modify their tokens or login as other users. Clients may attempt to access resources that are not available to them, but they will not lie about their identity.
- **Other Principals** You may assume that the only principals in the system are the AS, RSs, and clients. In particular, this means that you do not (yet) need to worry about the threats of passive attackers stealing information that is transmitted between the client and servers, or active attackers altering the data that is transmitted between the client and servers.

In short, you may assume that your code will be run by trustworthy principals in a closed environment. This does not mean that you should ignore good error handling and input validation, but you can assume that all entities in the system are trusted not to be malicious.

3 What do I need to do?

First, you will need to fully flesh out (and document) the design of your service. You will be asked to submit documentation that serves as both the full "user's manual" as well as the "technician's guide" for your system. Along the way, I recommend that you prioritize the following components.

- Fully specify all interfaces and protocols. This includes the following questions and many more: What functionality does the AS and RS provide? What messages do they respond to? How are messages encoded and sent over the network? Does the client stay connected through multiple requests and responses, or does it disconnect and reconnect for each request? What elements are included in the token provided by the AS?

  Do this early, so that when you distribute the work of building each part, you know how they are meant to interact through these interfaces. This parallelism will be very important to completing your project on time.

- Specify the access policy. What users can access what resources? What data is stored on the AS/RS that allow them to make the determination? If it's stored on the AS, then it needs to be declared in the access token, and you should describe how the RS checks it from there. If it's stored on the RS, then different RSs have independent (and perhaps conflicting) data, and you should describe how it is managed by each separate RS.

- Determine how the AS and RSs are run, and how to use the client. Are servers configured in any particular way? What server settings can be changed, if any, and how? How and where does the user specify which server(s) they are connecting to? Do they connect to one RS at a time, or multiple? What things should users expect may be different across different RSs?

After specifying any remaining details that you did not discuss in Phase 1, you will be building the software for the AS, RS, and corresponding client application. You are encouraged to use external resources to study examples of code that establishes network connections and transmits data across those connections. If you are writing your programs in Java, you might want to check out the following repository, which contains a simple chat client/server application that could provide some helpful hints about the networking components specifically.

`https://github.com/2241-cs1653/server-sample`

3.1 The Authentication Server

The primary purpose of the AS is to manage user metadata, which may include users' group membership, their assigned roles, credentials they have earned, and/or attributes that they are labeled with. This AS should serve as the central point at which (i) users are created and deleted; (ii) user metadata is updated and stored; and (iii) a user can obtain a *token*, *ticket*, or *certificate* that can be used to prove their identity and metadata to RSs. Once a user obtains a token from the AS, they can log into one or more RSs to view, upload, edit, and delete resources (according to your system's access policies).

Note that the token plays a hugely important role in your distributed system, as it represents the binding between a user and metadata that determines their accesses. Given the centrality of this component, it would be prudent for your project group to design your implementation of this interface *before* tackling any other coding tasks.

The AS should use multi-threading to serve multiple clients at a time, and should save its data in full when shutting down, so it can be re-launched without losing user data.

3.2 The Resource Server

In our scenario, there will only be one AS, but there may be many RSs. After obtaining a token from the AS, a user can make use of *any* RS (or RSs). As such, you **should not** hardcode specific server addresses or ports—in your demo, your TA may specify the machines on which you should launch the server(s), and thus that you should connect to. This also means you should test your code multiple times using different machines playing the roles of the servers.

Access to resources should be limited according to user metadata present in the token from the AS. That is, users should only be able to access resources that would be granted according to your documented access policy, under the assumptions stated in Section 2. In other words, honest users should not be given access to resources that they should not have. The obvious loophole is for users to lie about their identity, but that behavior is not (yet) part of your threat model!

As with the AS, the RS should use multi-threading to serve multiple clients at a time, and should save its data in full when shutting down.

4 Testing Environment

You will be asked to demonstrate your project's functionality using the last commit pushed to the default branch before the deadline, executing on the CS Linux Cluster. Your code will be evaluated based on functionality that can be demonstrated on these machines with communications across the network. As such, you must test your code in this environment well before the deadline to identify any bugs that may not be apparent when running in other environments.

The addresses for these machines are:

- `ritchie.cs.pitt.edu`
- `kernighan.cs.pitt.edu`
- `thompson.cs.pitt.edu`

You can connect to these machines via SSH using your Pitt username and password.[1] In order to connect to these machines from off-campus, you will first need to connect to the Pitt VPN. Instructions for GlobalProtect can be found on the Pitt IT page.

We recommend running the AS on one machine, the RS on another, and clients on a third.[2] Note that home folders are automatically synchronized between these machines,

[1] If you are running Windows, you may use the PuTTY SSH client. In the vast majority of cases, Mac, Linux, and other Unix-like operating systems will have command-line SSH clients pre-installed. Consult the documentation for your distribution if you need help.

[2] If your client is graphical, you can run the client locally using an SSH tunnel. Since your personal machine will not be able to communicate freely with the cluster servers, this will enable you to send data first to one of the cluster machines, making it appear to the server as if the incoming connection is from within the cluster. Read examples online on how to use an SSH tunnel, and ask on Discord if you need help.

which can mean files created by (e.g.) the AS can interfere with (or incorrectly pass information to) the RS. To be completely sure that communications are restricted to those implemented via your networking code, **each process should be run either by a different group member or from a different subfolder**.

5 What (and how) do I submit?

A repository will be created for your group via GitHub Classroom. You are encouraged to use this version control repository regularly while collaborating on this project, as each individual's contribution to the group's work will be judged in part by the version control logs.

Within your repository, include the following files and directories (this basic structure is already created for you—be sure to preserve it).

- `desc/` In this directory, we provided this project description. No changes are necessary.
- `doc/` In this directory, include any documentation for your project. As mentioned in Section 3, this should include a full user's manual and technician's guide (separately or combined) for your system. Submit your documentation in PDF or HTML format.
- `src/` In this directory, include all of your source code that is needed to compile your programs. You may create subdirectories (packages) within this folder if you'd like to better organize the code. Please do not commit any compiled code (e.g., JAR or class files). It is common version-control etiquette not to commit files that can be re-derived from the included source. Also, please do not commit any publicly available libraries that you make use of—include instructions (or, even better, a script) for acquiring those libraries.

  Note that we will be evaluating your submission using the CS Linux Cluster as stated in Section 4, so you must test your code in that environment prior to the submission deadline!

Each individual's contribution to the group's work will be judged in part by the version control logs. In addition, *each student in your group* will be asked to evaluate the group's performance, including how well everyone is working together and supporting one another.

Your project is due at the precise date and time stated above. We will clone your repository immediately after the due date, so you will be graded on whatever changes have been committed **and pushed** to your repository's main branch by this time. No changes made after this point will be considered in your demo or in grading your project. Make sure your repository is created and you understanding the submission process well in advance!