# An Authentication System for Student and Faculty Projects

**Aubrey Barnard**
**Computer Science**
**St. Olaf College**
**barnard@stolaf.edu**

**Richard Brown**
**Computer Science**
**St. Olaf College**
**rab@stolaf.edu**

**Theodore Johnson**
**Computer Science**
**St. Olaf College**
**johnsotm@stolaf.edu**

## Abstract

This paper describes the objectives and features of an authentication system that was developed at St. Olaf College for use with student and other programming projects. The system authenticates a user, a client, and a server that are part of client-server software. The system uses SSL, authentication tokens, and features advantages such as reusability, ease of use and integration, and complete security. Since the user login is a separate component, the system does not need to address the technical and ethical issues of authenticating users. This is one of the many features of the system that make it appropriate to student and faculty projects.

## Introduction

Security is a large concern at the forefront of today's computing world, one that grows larger every year. In a computing world that constantly deals with private information such as financial transactions and important correspondence, security becomes essential. Unfortunately, implementing and administering good security is difficult, and consequently people address the issue poorly or not at all. In today's world, success in security calls for thorough consideration from the very beginning of any project.

As the first step in a class programming project for a course in client-server applications, we developed an authentication system to support the network security needs of the project. Such a system was important to build because it does more than satisfy the needs of our project; it paves the way for incorporating quality authentication systems into any project. Our system integrates with the standard web authentication system on campus. This provides benefits such as: a user only needs one password to access all campus services, including those services created as student projects; the system requires only minimal maintenance by the campus computing services administrators; and project programmers never see or have access to passwords. A description and explanation of this authentication system follows.

The authentication system exists as a Java package that produces a secure and authenticated network connection. That package essentially adds a layer of security over existing Java sockets, providing an encrypted network connection with mutual authentication at each end of that connection. This authentication system can easily expand to support languages other than Java, and its design accommodates its reuse as part of many projects.

To get us started together, here are some definitions. Computer security is protecting the information in a computer. There are three main aspects to protecting computerized information (Russel and Gangemi, 1991, pp. 8-9). The most widely considered aspect is secrecy or confidentiality. This is keeping your information to yourself. There is accuracy or integrity, which means that your information is true and stays that way. Finally, there is availability or reliability, which means that your computer can do the work you give it when you want it done, without interference. Authentication is proving your identity. Authentication provides confidentiality and accuracy, and can help availability by preventing unauthenticated users from using (attacking) your computer. Because it addresses these three aspects, authentication is one of the most important goals in computer security.

## Motivating Context

As the programming project for a course in client-server applications at St. Olaf College during fall semester of 2003, a team of students designed and built a piece of software that enables students to plan and track their progress in programs of study, such as majors. This software processes sensitive personal data like registration information, student identification numbers, and grades, thus raising privacy concerns and creating the need for an authentication system. Such a system would need to identify and authenticate students, advisers, program directors, and any other users of the system before granting them access to any private information. It also needs to ensure the security and integrity of the network connection for the duration of the communication. The system can then transmit data reliably, without intentional or unintentional interference. Therefore, we developed an authentication system to satisfy this security need.

Our system can satisfy the need for secure network connections in other software on campus. Since one can reuse our system for other projects, it has value beyond the scope of its original project. This was important to us as we designed the system, and motivated a design that would make the system independent and reusable.

As programming projects become more practical and realistic, or are intended for actual use (as this one was), they will need to seriously address security needs. Real-world projects are great educational experiences, but they also need real-world solutions. The concern for security in today's computing world motivates the development of a reliable and reusable system for creating and maintaining secure, authenticated network connections. Ours is such a system.

## Objectives of the Authentication System

The general objective of the authentication system is providing a user with access to private information in a piece of client-server software. The methods used to access and manipulate the information must protect the privacy of the information. This implies, among other things, that the information remains private while the software transmits it over the network, that users can only access information the privacy policy allows them to access, and that users prove their identity upon log in. One can break down these large-scale objectives into specific security requirements, which I explain below.

The authentication system should create and implement an authentication strategy that satisfies the following security requirements and goals.

The authentication system must be secure beyond a reasonable doubt. Naturally, this relies on assumptions about the security of underlying systems. (I cover these assumptions later.) Therefore, it should be implemented on top of protocols and strategies that are widely used and considered to be quite secure (e.g. SSL, RSA, HTTPS). Eliminating the doubt of underlying systems, by investigating them, reimplementing them, or otherwise, would certainly be impractical, and probably impossible, so we will have to make assumptions about them. Actually, the crux of this issue is how much security we will be satisfied to have for the cost we are willing to pay.

For the system to be secure beyond a reasonable doubt, we must deem it secure when we think carefully about the security of the system. That is, we cannot think of any security holes in the system. To support this, the system should have a clean conceptual model, one that makes it easy to think about the security of the system. Ideally, the system should be provably secure. In part, this depends on being able to formalize the function of the system (simply and concisely enough) so that we can logically and mathematically prove it secure. This would eliminate any doubt about the security of the system. Any other doubt would then concern the layers on top of which the authentication system is implemented.

An important goal of the authentication system implementation is that it be reusable. One should be able to easily integrate the system into any project that needs secure network communications. This reusability should be available without administrative intervention or assistance after the initial setup, which means that the parts of the system requiring administration should be independent of the parts that projects will reuse. Student and non-student projects on (and possibly off) campus should be able to reuse the system without modifications. Reusable software is not just a convenience, it actually helps improve security. Reuse of code is a fundamental software paradigm because it avoids the effort and hazards of reinventing and reimplementing existing systems. A single reusable, reliable, easy-to-use, trusted way of authenticating reduces the effort of everybody implementing their own security, and more importantly lessens the chance of varied and independent security implementations each having their own security flaws. With one piece of reusable software the effort and bugs are consolidated. Also, everybody who might otherwise be working on competing solutions, can work together to develop one solution, and consequently do it really well, much better than anyone could do it individually. This approach will help make the authentication project successful.

This authentication system must authenticate three entities, the user, the client, and the server. Each actor in the system must be identified and authenticated, because one cannot blindly trust a network connection. Threats can easily sniff or otherwise manipulate network traffic, possibly leading to security vulnerabilities. Consequently, the system must secure network connections before it transmits valuable, private data. The system must ensure that threats cannot impersonate a legitimate actor by securing the network connections and authenticating each actor. This is the essence of the authentication system, and the basis for secure network communications.

A practical objective of the system is that it be appropriate to student programming. If students will use the system as part of their projects, then it should support student programming as much as possible. There are two main parts to this objective. First, the system should have an API, documentation, and an implementation that supports student use. This is especially important for students who are learning about networking and computer security. Second, there are ethical issues to consider when students deal with the passwords that authenticate the users. Basically, students should never have access to a password. Students probably could be trusted with passwords when in a contained situation. However, the misuse of passwords is not a risk that we want to take. Likewise, students should not program any code that handles passwords. Code that handles passwords would be very crucial, and would require extensive verification by the faculty member in charge of the project. This unnecessarily consumes the time of the students and professor. More importantly, having students program code that handles passwords gives the students the opportunity to insert code mechanisms to steal passwords or otherwise subvert the system. Therefore, keeping student code and involvement entirely separate from the password verification process is a decision that benefits security, validity, and reliability.

Another objective of the system is that the its authentication should rely on a single, trusted authenticator. That is, a system with a good security history that already performs authentication should perform the authentication for our system. Reliance on a trusted authenticator is partly a consequence of the decision to make passwords inaccessible to students, since someone other than the students has to process the passwords. This sounds like it would be an inconvenience, but it actually makes the system much simpler and more secure. Using a trusted authenticator is an advantage because it means that there will be only one, central source of passwords. Just as students can reuse the security package for different projects, everybody can reuse their passwords as well. There is also the advantage of using existing mechanisms and policies that are the responsibility of the trusted authenticator. These include password services such as password authentication, screening new passwords to ensure their security, password expiration, and other aspects of password management, like changing your password. Essentially, having one entity that processes all the passwords follows the programming paradigm of software reuse, and has the corresponding benefits.

The last objective of the authentication system is that it should use the actual UNIX passwords that the students and faculty use on campus. This ties in with the benefits and use of a single, trusted authenticator. It also provides compatibility with other college services such as e-mail and student records, thus providing a convenience to students and faculty. It avoids the ethical and technical issues of having new passwords for each campus service. For example, having many passwords for many services places undue burden on the user to remember all the passwords. In turn, the users are then more likely to repeat or forget passwords, which decreases security. Another ethical and technical issue is how to store and process passwords securely for each new system. Only the trusted authenticator needs to address these issues if each system uses the campus UNIX passwords and relies on the trusted authenticator for password services.

Besides these technical requirements, there are implementation goals, namely implementing the authentication system at St. Olaf College or any other college. One must communicate and cooperate with the computing services of the college, if they are to be the trusted authenticator. There are also the issues of publicity, availability, distribution, and maintenance. The system must be publicized so that people will know about it, and can choose to use it with projects. Then, we must make it available through a system of distribution so that the interested people can use it. Also, there must be people willing to work once in a while to maintain the system. These issues may seem more like logistic concerns than objectives, but they are necessary to the success of the system, nonetheless.

## The Authentication System

### *Assumptions Underlying the Authentication System*

In order for the authentication system to be secure beyond a reasonable doubt, we must assume certain things about the security of the the underlying systems on which the authentication system relies. If these security assumptions do not hold, the security of our system is in jeopardy.

- First, the integrity of the hardware is important. There must be physical security of the computing and networking hardware. The computers and network must also be reliable.
- The operating system must have integrity, particularly to ensure that a threat cannot compromise the administrative accounts (e.g. root), and gain control of the operating system.
- We assume that a threat cannot hijack a socket. This relies on the integrity of the operating system. This assumption allows us to trust a single socket connection for the duration of the client-server interaction without reverifying the connection at any point.
- We assume that SSL provides adequate protection against someone snooping or hijacking the network connection.
- We assume that we can indeed trust the trusted authenticator. It is especially important that nobody can compromise the trusted authenticator, since our authentication relies on the integrity of the system introducing the client and server.
- We trust that the underlying protocols (TCP/IP, Ethernet, etc.) and their implementations are secure, at least secure enough for our purposes. We encrypt the network connection, but it is still possible to sniff packets and learn about the behaviors of the network even if the contents of the packets cannot be read. However, every network is vulnerable to varieties of low-level packet interference, and so we assume that this will not be a problem, since the higher-level protocols ensure proper packet transmission. (Cheswick, Bellovin, and Rubin, 2003, pp. 19-21)
- We also trust that using a user name and password to identify and authenticate a user is a sufficiently secure method of authentication.

With these assumptions in place, we are confident that our system is secure beyond a reasonable doubt, and that we have considered all the possible security flaws.

***Description***

The following is a description of the authentication system we developed to solve our authentication needs. There are basically three parts to the system: the trusted authenticator; the client; and the server. For the general idea of how the system works, consider introductions at a party. The host(ess) introduces two new guests to each other, and since both of the guests trust the host(ess), they trust that they have been properly introduced. Figure 1 shows the parts of the system.

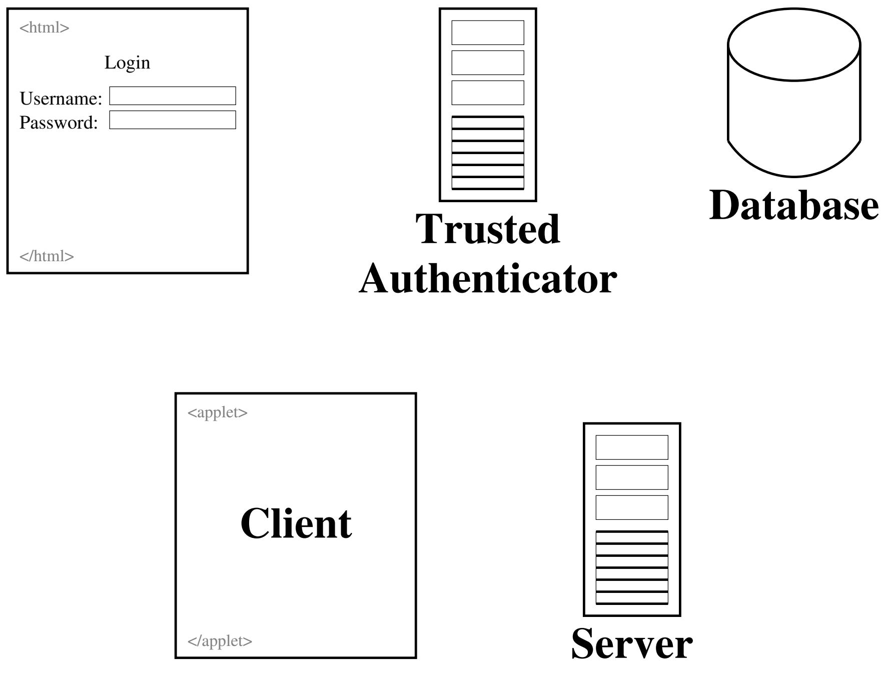

Figure 1: The authentication system

The first thing a user will encounter when using the system is a web page that introduces them to the system and asks for a user name and password. All further authentication relies on this login, and assumes that the user has logged in successfully. The web page will securely transmit, using HTTPS, the user name and password to the trusted authenticator.

The main responsibility of the trusted authenticator is to process passwords, and authenticate users when they log in. It also must generate and store authentication tokens and provide the client to the user. To do this, it will run a CGI script that receives and processes the information from the user login page. The script authenticates the user, and, if that is successful, then generates a session identification number (ID), and two long random numbers, authentication tokens for the client and server. It stores these numbers in a database, so that the server can access them. The trusted authenticator accesses the database over a secure connection, and uses a transaction to ensure the database operation was successful. Then, it returns a secure web page with the client embedded as an applet. The returned web page contains the session ID and authentication tokens as arguments to the applet. The client starts in the user's browser, beginning the process of client and server verification.

Once the user logs in, and the system confirms their identity, the client and server must verify each other's identity. The client starts this process by making a secure network connection to the server over an SSL socket. This socket connection is the single connection created between the client and the server for the duration of the client process. This reuse of a single socket connection eliminates the need to verify future network connections because there is only ever one connection. Once the network connection is in place, the client sends the session ID and its authentication token to the server. The server uses the session ID to retrieve authentication tokens of itself and the client from the database, using a secure connection. The server immediately erases this information from the database so that it can only be used once. The server conducts the retrieval and deletion of the tokens as a transaction so that the operation is failsafe. At this point, both the client and the server know what each other should present as their authentication token. The server verifies the token the client sent against the one it retrieved from the database. If this is not successful, the server terminates the connection. Otherwise, the server sends its token to the client. The client verifies the server's token against the one it received as an argument. If this fails, the client terminates the connection. Otherwise, the client sends an acknowledgment that all verification is successful and complete, and that they (the client and server) can start communicating to accomplish the intentions of the program.

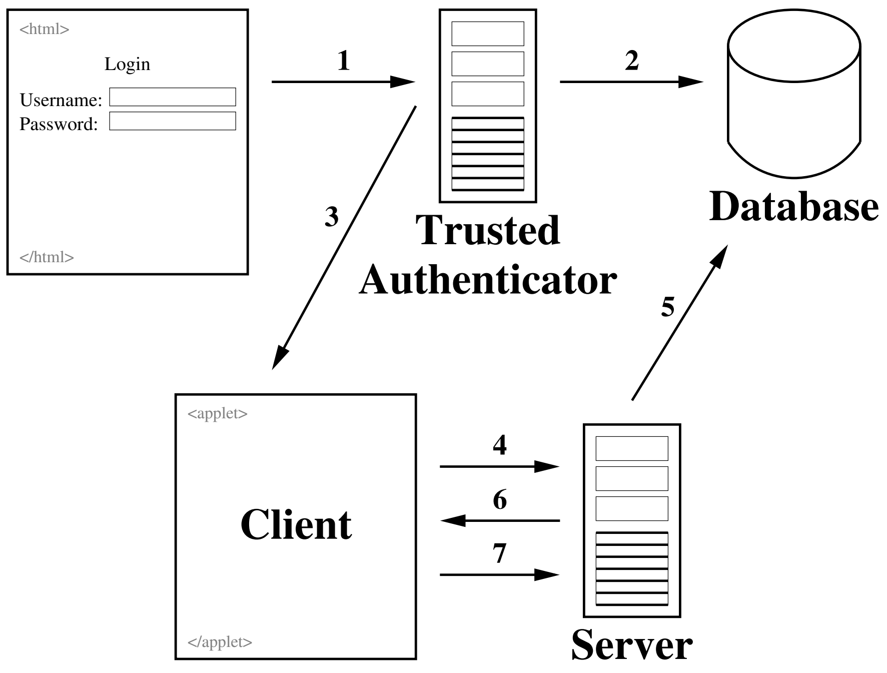

Figure 2: The steps of the authentication process

Consider the summary of the authentication process shown in Figure 2. In step 1, a user logs in using the login web page. In step 2, the trusted authenticator generates and stores the authentication tokens. In step 3, the trusted authenticator sends the user the web page that starts the client in the user's browser. In step 4, the client makes a connection to the server, sending its token. In step 5, the server retrieves the authentication tokens from the database. In step 6, the server verifies the client's token and sends its own token to the client. In step 7, the client verifies the server's token, and sends a message to the server saying that they are ready to communicate.

This system satisfies all the security objectives mentioned above. It is reusable because the package provides networking just like many other APIs. This networking hides the client-server verification process. Also, in the case of St. Olaf College, the web login interface already exists for other services, and the login processing script fits seamlessly into the existing user authentication process. The system clearly authenticates the user, the client, and the server, and does so in a way that cannot be compromised. The package is appropriate to student programming and future projects because it provides a standard networking API, and has good documentation. The system itself does not process any passwords, leaving this responsibility to the trusted authenticator. The system uses actual UNIX passwords, and shares the benefits of the other services that the trusted authenticator provides.

## The Software Package

The authentication system consists of a software package written in Java that is a substitute for regular Java networking packages. The name of the package is SecureVerifiedSocket, and it contains the classes SVClientSocket, SVServerSocket, and two exceptions that signify verification failure on the server side and client side. These classes provide the programmer with comprehensive network functionality, and can be used just like any other Java sockets. To use the package, simply replace the existing client and server sockets with the versions from the package. There is also a Perl script that the trusted authenticator runs to generate and store the session ID and the client and server authentication tokens. This script also starts the client applet. Although the system requires some other components in order to work as described, these are the core components needed to use our authentication system as part of a project.

There are a few components we do not include in our package, but are necessary to using our authentication system as part of a project. There needs to be a web page, or some other mechanism, that allows a user to log in. The package does not include this because such a piece is specific to the trusted authenticator and the system in which it will be used. In our case, the campus computing services provide such a web page and login service. The package does not include a database or database driver, as these components would be highly impractical to include. However, one can easily modify the package to use an SQL database of your choice, simply by specifying the appropriate driver. Although it would be beneficial to student programming, the package does not include prototypes for the client or server. We assume that a student using our system will already be familiar with networking.

## The Future of the System

This system is in its infancy. In the future, we hope to provide support for languages other than Java as well as support for other databases. We would like to be able to support all the varying types of encryption so that a programmer could choose which method suits their project best.

We would also like to create a formal analysis of the authentication system, and try to prove it is correct and secure. We feel that this would not only be a beneficial academic exercise to further understand security and our system, but it would be an asset to the success our system.

## Conclusion

I have described to you an authentication system for use with campus student projects, a system that has value within and beyond the campus due to its quality and reusability. Whether you consider appropriateness for student projects, integration with existing campus authentication and services, or the simple yet solid conceptual model, our authentication system is secure and easy to use. This is because it was designed from the beginning with security in mind, a necessary approach for it to be successful. While many modern systems have poor security patched on as an afterthought, we are confident that our system is a step in the right direction in a world of ever increasing security concerns.

## References

Cheswick, W. R., Bellovin, S. M., & Rubin, A. D. (2003). *Firewalls and Internet Security* (2nd ed.). Boston: Addison-Wesley.

Russell, D., & Gangemi, G. T., Sr. (1991). *Computer Security Basics*. Sebastopol, CA: O'Reilly and Associates, Inc.

## Acknowledgments

I would like to acknowledge the help and cooperation of the Information and Instructional Technologies at St. Olaf College, especially Robert Breid. They helped us understand certain security issues and agreed to act as our trusted authenticator.