Implementation of Two-Factor Authentication with U2F for Gravitational Teleport

CPEN 442 Partial Project Report

December 11, 2016

Alex Charles (36700128), Jay Dahiya (31135130), Jason Jang (35408129), Bibek Kaur (15093123)
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

***Abstract* - Two-factor authentication is becoming commonplace in our society for users to securely login. This has arisen through the utilization of mobile applications such as Google Authenticator. However, Google Authenticator has various issues, including usability drawbacks and vulnerabilities to phishing and man-in-the-middle attacks. These problems are solved by the use of Universal Second Factor (U2F). We introduced 2FA with U2F to a security product, Teleport, allowing users to authenticate with it in place of Google Authenticator. We refactored their code to handle different forms of 2FA, and extensively tested our implementation using their current testing frameworks. Through U2F support, the overall security and usability of Teleport has increased. The pull request acceptance is currently pending, but the U2F implementation has been reviewed and approved by Gravitational's CTO. We expect it to be accepted by the end of 2016.**

I. Introduction

Two-factor authentication (2FA) is utilized throughout the world as a means of confirming a user's identity. Two-factor refers to the individual authenticating themselves through two out of three of the following methods: secrets, possessions, and biometrics. These are commonly known as something you know, something you have, and something you are. By requiring an attacker to authenticate through multiple ways, 2FA can reduce the chance of identity theft or other forms of online fraud significantly. It is clear that a 2FA system carries many benefits, and can reduce the chance of a successful attack [2].

Many systems utilizing 2FA do so at the cost of limited usability. For example, a user authenticating a credit card payment may have to sign the receipt, or provide a PIN. The popular 2FA app Google Authenticator requires users to enter in a generated 6 digit code, in addition to their username and password. This reduction of usability, commonly associated with 2FA, discourages adoption of the increase in security it provides. The aim to promote ease of use, security, and standardization among strong authentication devices, led to the creation of Universal Second Factor (U2F) by Google and Yubico. U2F is a 2FA system based on hardware tokens and a challenge-response protocol. The success of the project has been confirmed by Google, who published a study which found U2F to be more usable and more secure than one-time codes.

Currently the FIDO Alliance, a collection of over two hundred technology companies, promotes and hosts U2F as their primary means for 2FA. U2F is gradually becoming commonplace, and today it is supported by a host of applications, including Dropbox and Github, all major operating systems, and Google Chrome. Other web browsers are planning on supporting it in the future, and it is rapidly being adopted across security-critical areas [3].

We implemented two-factor authentication using the U2F standard for Gravitational Teleport, an open source, managed SSH solution for server clusters. Currently, Teleport only supports two-factor authentication with the Google Authenticator mobile app. We refactored Teleport's web client, command-line client, and server code to allow for different forms of 2FA. We added U2F as an option for 2FA, and included simple, usable instructions for the user.

Through extensive unit and end-to-end testing, we are confident in the correctness of our solution. In our development of unit tests, we used the testing frameworks and libraries currently employed by Teleport. On the server side, we utilized the testing library gocheck, and created a mock U2F device to aid in unit testing. For the web application, we wrote tests under the Mocha JavaScript testing framework. We purchased U2F devices, and conducted end-to-end tests with these. Finally, we conducted usability testing to ensure our UI changes were acceptable.

II. Existing System

Teleport allows users to authenticate once and log in to multiple Linux servers within the cluster. The system consists of the web client, the command-line client, and the server, which provides the three services - node, proxy and auth. The web client is written using standard web technologies in JavaScript, and the command-line client and services are both written in the Go language.

Each server instance can be configured to provide one or more of the node, proxy and auth services. The node service runs on the individual servers in the cluster and provides SSH access to them. The proxy service, as the name suggests, accepts connections from the client and routes them to nodes. It also serves the web interface to browsers. The auth service is the access control authority and handles authentication and authorization. Figure 1 illustrates the interactions between the components of the system.

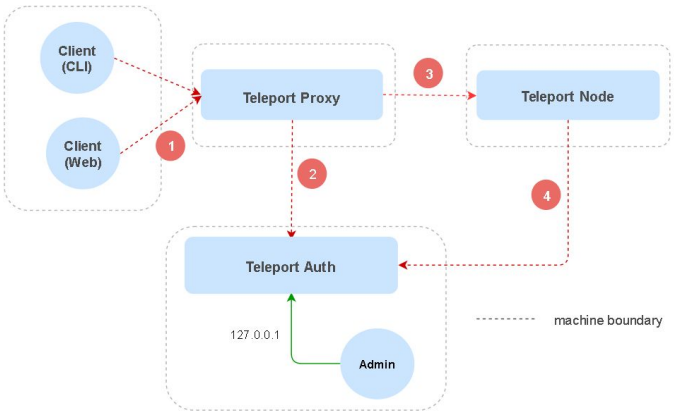

Figure 1: Teleport's primary components

The four interactions shown in the diagram are as follows:

1. The client connects to the proxy and presents its certificate.
2. The proxy checks with the auth service to see if the certificate is valid. If the certificate is invalid or expired, the proxy will require the client to authenticate. Once client sends the correct credentials, the auth service will issue a new temporary certificate to the client, which the client can use to connect with.
3. The proxy connects to the node. Additionally, the proxy also records the session, which it forwards to the auth service for auditing.
4. The node checks with the auth service to see if the certificate is valid and whether the user is authorized to log in.

Figure 1 also shows that the Admin tool, used to configure the auth service, can only be run on the same machine that runs the auth service.

Since our implementation covers authentication, we only modified the clients, the proxy, and auth services.

This system's stakeholders include Gravitational, their customers who use Teleport, and indirectly the customers of the users of the Teleport system who may have their private data stored on servers which are accessible using Teleport.

Adversary Model

The adversary model for the existing system is as follows.

Objectives

- To gain access to the account of a user in order to acquire SSH access to the company's servers.

Initial Capabilities

- Install keyloggers and network traffic loggers on the users' computers
- Visually observe the users while they enter the password and one-time token

Capabilities During the Attack

- Reuse credentials gathered with previously installed keyloggers and network traffic loggers
- Physical access to the users' password protected mobile phones

III. Related Work

Soon after publication of the U2F standard, FIDO released *FIDO U2F Implementation Considerations*, detailing various points of interest and guidelines, in order to correctly integrate the standard into a project's authentication scheme. This paper directly addresses our project's goals. Among other things, it contains information on timing considerations, key generation, and the U2F tokens. By referring to it throughout the development, we can be assured that the best practices laid out by the designers of U2F are followed [4].

Additionally, Google conducted a study on the usability and security of U2F tokens compared to one-time passwords (OTP) delivered via SMS or a mobile app and found that users who use U2F tokens spent on average less time and made fewer mistakes authenticating than users who use OTP via SMS or OTP via mobile app. U2F tokens additionally provide protection against phishing, which OTP is vulnerable to. They also found that while U2F tokens incur a one-time cost, the savings in recurring IT support costs related to two-factor authentication offsets the cost of buying the U2F tokens. This paper provides further justification of the necessity of our enhancement[6].

IV. Adversary Model

The objectives of the adversary are the same as for the existing system - to gain SSH access to the company's servers. The adversary model for the U2F-based system assumes the attacker has the following capabilities in addition to the capabilities assumed by the adversary model of the existing system.

Initial Capabilities

- Send phishing emails that link to a fake login page set up by the attacker

- Send phishing emails that instruct users to install mobile malware
- Exploit mobile security flaws to install mobile malware on the users' mobile phones

Capabilities During the Attack

- Steal one-time token codes or the secret key used to generate the token codes using previously installed mobile malware
- Steal one-time token codes via direct line of sight, and by entering the token before the legitimate user does

V. System Design

We implemented U2F authentication using existing open-source libraries and programs. This is in accordance with the Teleport project's philosophy of using off the shelf security. Not only is using existing security libraries following the Principle of Open Design, it is also an example of the Economy of Mechanism, as this reduces the amount of code we need to write, and therefore the chance of security-critical bugs being present in our implementation [1].

For the server, we used a U2F server library written in Go. For the web client, we used an npm port of the official JavaScript U2F client library, as this integrates more cleanly with the existing web client code. For the command-line client, U2F authentication was implemented by calling the official u2f-host command-line utility, as there is no U2F client library for Go and the u2f-host utility is supported on all operating systems supported by Teleport. Because the u2f-host program is typically installed in locations that can only be written to by the root user, using it does not diminish the security of the system as an attacker who already has root access to the victim's computer can simply steal the user's temporary certificate after the victim logs in.

Teleport uses role-based access control. Because the U2F system needs to allow partially authenticated users who have entered the password correctly to get a U2F authentication challenge and do nothing else, all of the existing roles were too broad. In following the Principle of Least Privilege, a new role was created specifically for partially authenticated users to get the U2F authentication challenge.

U2F requires the server to persist the authentication challenge between client requests. Each challenge is randomly generated by the server and expires after five minutes. We chose to store only the most recently created challenge for each user, instead of storing all challenges until they expire, as one of the storage backends that Teleport can use does not fully support automatically expiring values. In particular, values that have expired cannot be read, but are also not cleaned up until an attempt is made to read them. If an attacker knows the password of a user who utilizes U2F, we have the choice of either allowing the attacker to perform a denial of service attack on the user by repeatedly requesting authentication challenges to overwrite the authentication challenges for the user, or allowing the attacker to perform a denial of service attack on the whole system by filling the authentication server with authentication challenges. We chose to allow the attacker to perform a denial of service on a single user rather than the whole system, as this minimizes the impact of the attack. Furthermore, such denial of service attacks on a single user can be mitigated by the system administrator, by resetting the user's access credentials.

It should be noted that because Teleport is an open source project, our implementation also follows the principle of Open Design.

VI. System Implementation

We first implemented the server, which has a REST interface, as it is the common endpoint for both clients. Testing it does not rely on either client, as it can be done by manually sending requests using a browser tool such as Postman, or the curl command-line tool. Initially, we investigated the flow of data between the various parts of the server to identify the most appropriate places to call the U2F server library, the data that needs to be persisted, and the most appropriate place to store the data. We then implemented data marshalling and persistence. Challenge and Registration are the two primary data structures that needed to be stored. Data marshalling and data persistence in the existing system is done with JSON encoding and a key-value store. The Challenge data structure can be encoded in JSON using Go's built-in JSON support, but the Registration data structure cannot, because it contains a public key structure which utilizes pointers internally. We implemented a marshallable Registration structure which stores the public key in DER-encoded PKIX (Public-Key Infrastructure X.509) format. After data marshalling was implemented, the Challenge and Registration data structures can be stored in the database, along with the existing user data.

Once we had working data persistence, we implemented the four APIs required by U2F - register request, register response, sign request, and sign response - in this order, as each API could only be tested after the previous one had been implemented.

We then implemented the web client, as the web client has more functionality than the command-line client. Implementing the web client early allowed us to find potential design problems in the server API before we write more code that depends on this API. We made modifications to the signup page and the login page. In the signup page, we added an option to allow new users to choose whether they want to use Google Authenticator or U2F as their second factor. If the user chooses Google Authenticator, then the signup process proceeds as before and the user is configured to use Google Authenticator. If the user chooses U2F, then the new U2F-based server APIs are used instead, and the user is configured to use U2F. In the login page, we added a second login button that allows users who use U2F as their second factor to log in. We chose to add a second button instead of an

option to minimize the number of clicks required for the user to log in.

Finally, we implemented the command-line client, which shares some of the server APIs with the web client. We added a new command-line option to allow the users to choose U2F as their second authentication factor. Requiring the second authentication factor to be specified by the user simplifies the design of the system, and prevents attackers from learning which second factor is used by a user whose password they know.

VII. Evaluation Methodology

We evaluated our implementation with manual and automated testing for functionality, and conducted subjective evaluations of the user experience. First, the server was tested by manually calling the APIs using a browser tool or the curl command-line tool. Once the web client was implemented, we tested the web client against the server. The command-line client was also tested against the server after it was implemented.

To make sure that future changes to the code will not break the U2F authentication functionality, we added new tests to the existing test suites for the server and the web client to test the newly added functionality.

The tests for the server are written in Go using the gocheck library, which is used by the existing tests for the server. Because the U2F standard uses a challenge-response protocol, the server APIs cannot be fully tested by playing back previously recorded traffic. Our solution to this issue was to implement a mock U2F device in software, in accordance with the U2F standard, to create the appropriate responses to the challenges from the server.

The tests for the web client are written in Javascript using Mocha and PhantomJS. Since the web client is tested independently of the server using faked server responses, the same issue of having to generate proper U2F responses does not apply, as the server responses for successful or unsuccessful authentication can be created by the test.

Additionally, the team evaluated the design of our user interface by trying it ourselves, and asking our friends for feedback.

Upon conducting end-to-end manual testing for both the new and existing 2FA methods in the Teleport system, we found that they both functioned correctly. The existing test cases and the new test cases for the U2F functionality also passed. This gave us the confidence that our added feature was ready to go under review from the Gravitational team.

VIII. Discussion

This project provided us the opportunity to enable thousands of Teleport users to use U2F, a considerably more secure, more usable method of two-factor authentication than Google Authenticator. Moreover, U2F is considerably faster than Google Authenticator (Figure 2) as it does not require the time to run a mobile application or copy the key to the login prompt [5]. Apart from U2F being simple to use, it is also portable across all operating systems.

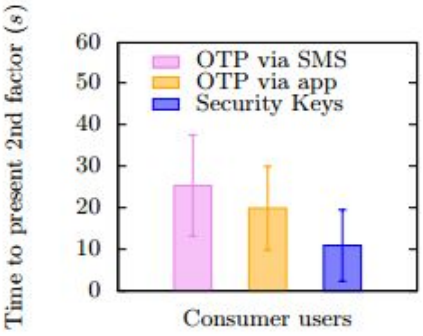

Figure 2: Time Spent Authenticating

In addition to the usability concerns previously discussed, mobile applications bring the distinct disadvantage of sharing Authenticator offers no built-in protection against phishing or the large attack surface of smartphones. Additionally, Google man-in-the-middle attacks. U2F's physical hardware keys, and the protocol it utilizes, avoids this problem. This is because the websites are identified to U2F hardware keys by their domain name, thus increasing the difficulty to perform phishing and man-in-the-middle attacks (Figure 3). This is due to the fact that potential attackers would have to gain control of the domain name and acquire a SSL certificate for the domain name in order to interact with the U2F device. Additionally, U2F detects the cloning of the device through the use of a device counter. This acts as a fail-safe in the event that the U2F device's tamper resistive nature is compromised. An additional benefit of being hardware based and a two-way protocol, is U2F's resistivity to mobile malware and shoulder surfing.

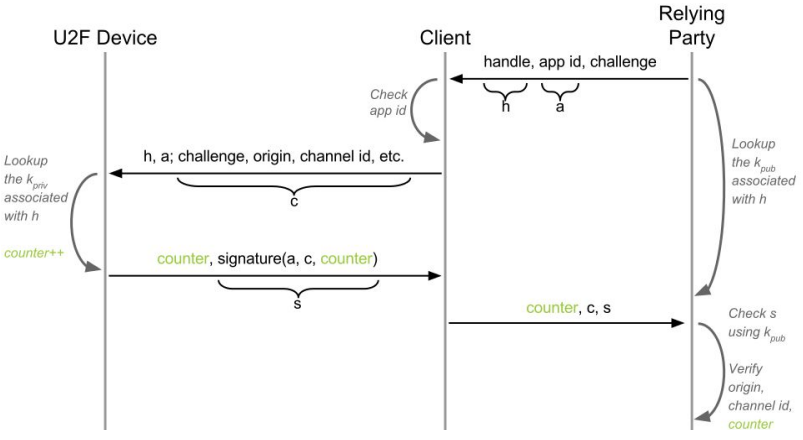

Figure 3: U2F Protocol

Though it can be argued that U2F devices have an entry barrier in the form of a cost (starting at approximately $10) compared to Google Authenticator accessibility as a free application for smartphones. An argument can be made that Google Authenticator assumes a user is in possession of a

smartphone, which might not always be the case due to potential security regulations in companies.

Once we started working on this project, we familiarized ourselves with U2F and its inner workings by reading the *Universal 2nd Factor (U2F) Overview* document. Additionally, we also found open-source libraries which were beneficial to us in accordance with the design phase of this project.

Compared to the existing two-factor authentication system based on Google Authenticator, our U2F-based system has some advantages and disadvantages. Because we chose to store only the last issued challenge of each user to avoid a possible denial-of-service attack on the entire system using the credentials of one compromised user, an attacker who knows a user's password can perform a denial-of-service on that user by repeatedly requesting authentication challenges to overwrite challenges for the legitimate user. In the existing system, this is not possible, but it is much easier to perform an exhaustive search of the keyspace of a 6 digit code, than the keyspace of a digital signature. It can be argued that the behaviour of our U2F-based system is acceptable, if not better, because it follows the principle of fail-safe default.

A notable difference between the two 2FA systems is that in the U2F-based system, an attacker can discover whether a user exists by bruteforcing the username-password combination. In the Google Authenticator-based system, the server would only give a positive response if the username, password and one-time passcode are all correct. This is a disadvantage of our U2F-based implementation and it is difficult to avoid because we would have to issue fake U2F challenges for nonexistent U2F tokens. Generating fake "key handles" for the dummy challenges that are indistinguishable from real key handles is difficult, as different manufactures of U2F tokens can generate key handles differently and we would have to analyze the key handles generated by U2F tokens from different manufacturers. Doing it successfully while keeping the project open source would also add complexity to the fake key handle generation algorithm. We discussed this issue with the Teleport project maintainers and decided to simply return an error upon receiving an incorrect username-password combination.

We submitted a pull request once the implementation and testing phases were completed and we were satisfied with the result. Our implementation of U2F as a 2FA in Teleport is currently under code review. The project maintainers from Gravitational do not have any major concerns about the functionality of our added 2FA option. They have approved the security aspects of our implementation, and they will soon review the user interface changes. U2F 2FA is one of the milestones of Teleport version 1.5 and we expect the pull request to be accepted after code review.

IX. Conclusion

The benefits of U2F to a business like Gravitational are evident. Their customers' confidence in the security of their system is paramount to their business's growth and success. U2F helps protect the confidentiality and integrity of the system by providing a more usable and more secure form of two factor authentication.

A possible extension to this project is to merge the mock U2F device we developed for Teleport into the U2F server library repository in a separate pull request. The mock U2F device may be useful for the testing of similar projects.

References

[1] E. Kontsevoy et al. (2015). Gravitational - the Private SaaS Company [online]. Available: http://gravitational.com/

[2] B. Krebs, "Citibank Phish Spoofs 2-Factor Authentication," *The Washington Post*, 11-Jul-2006.

[3] S. Srinivas, D. Balfanz and E. Tiffany, *Universal 2nd Factor (U2F) Overview*, 1st ed. Fido Alliance, 2014, pp. 4-24.

[4] D. Balfanz, "FIDO U2F Implementation Considerations." FIDO Alliance, 09-Oct-2014.

[5] S. Ehrensvard. (2015). Why Yubikey Wins [online]. Available: https://www.yubico.com/2015/11/why-yubikey-wins/

[6]J. Lang et al., "Security Keys: Practical Cryptographic Second Factors for the Modern Web," Google, tech., 2016.

[7] "U2F - FIDO Universal 2nd Factor Authentication," *Yubico*. [Online]. Available: https://www.yubico.com/about/background/fido/