Expanding Universal Second Factor (U2F) to Non-Browser Applications

Anonymous

Abstract—There is a critical need for improving the authentication standards because of breaches being caused by weak usernames and passwords. With more than 60% rise in credential thefts and violations occurring across global major corporate organizations last year, second factor authentication is a must. To address these concerning issues, the FIDO alliance launched a new protocol in 2015 called the Universal Second Factor (U2F). The main objective behind launching this protocol was to provide an open, easy to use and secure experience to everyday users and simplify the two-step authentication mechanism. U2F minimizes the reliance on passwords by providing resistance against phishing, man-in-the-middle (MITM), replay threats through end-user hardware and one-tap convenience. The U2F devices are user-friendly to carry and provide utmost security through public key cryptography at a faster speed.

Currently, U2F is available on Google Chrome, Mozilla and Opera browsers. This project focuses on expanding Universal Second Factor (U2F) to non-browser applications. There is ongoing research on further extending U2F to mobile devices so that a user can use second factor authentication for daily applications such as Gmail, Facebook and PayPal. The most likely options to be considered are Bluetooth and Near Field Communications (NFC) devices. This project implements an application on a mobile operating system such as Android that simulates the U2F protocol by communicating it with a Bluetooth Low Energy (BLE) device.

Index Terms—second factor authentication; U2F; public/private key, elliptic curve cryptography; Bluetooth; mobile; Android

I. INTRODUCTION

There is a significant increase in the number of security breaches occurring in this modern world of Online authentication. The two most common phenomena observed in these credentials are that they can be very easily guessed or users replicate the same combination for multiple accounts. Several organizations like FIDO, SQRL, OAuth, Identity 2.0 and others are taking efforts to overcome this serious issue. FIDO (Fast IDentity Online) is an industry non-profit organization that was founded in July 2012 to provide strong mechanisms for online authentications [2]. The alliance proposed solutions for some common problems that everyday users face such as creating multiple usernames and remembering multiple passwords. FIDO has developed technical specifications that support more secure, extensible and pragmatic techniques that reduce the dependency on passwords to provide authentication for users. The FIDO authentication solutions are adopted and recognized by the top IT firms and service providers such as Google, Microsoft, Bank of America, PayPal, SalesForce, Samsung. FIDO has designed and developed target specifications that will support a wide range of technologies. A user can confirm his authentication through face or voice recognition, biometrics like iris scanners and fingerprint detection, other communication standards like USB security tokens, Smart Cards, Near Field Communications (NFC), Bluetooth. Currently, the FIDO Alliance provides two categories of user experiences:

A. Universal Authentication Framework (UAF)

The FIDO UAF Protocol provides a passwordless experience in which a user can utilize an online service by offering their unique biometric as a substitute for password [4]. The user will have to carry a device that has the UAF stack installed in it. In this approach, the user first registers the device by providing an authentication using face recognition or iris scanning by looking at the camera, voice recognition by speaking into the mic, fingerprint scanning by swiping a finger. After the successful registration process, the user can repeat the same authentication technique provided in the first step whenever they want to authenticate the service. Also, UAF can provide a combination of multiple authentication methods such as entering a PIN + voice.

B. Universal Second Factor (U2F)

The U2F offers one device to many services as well as one service to many devices. It is very easy to use this device by simply inserting it in the USB port of the computer and pressing or tapping a button. The U2F device is safe and provides security resistance against phishing, man-in-the-middle and replay attacks. The core idea behind this protocol is standard public key cryptography in which the user’s device will mint a new key pair and will share the public key to the server [1]. The server will then send a request to the user’s device to verify the credentials and sign the data. Consumers are providing a positive feedback since its launch
Fig. 2. U2F Registration

and availability in the online market. The privacy of this protocol is guaranteed as the keys generated are site specific and there is no unique ID per device. This means that even if a user accidentally looses the device, it cannot be used to login to the original user’s account by another person. Companies that manufacture U2F devices are FIDO certified and can be trusted. The most beneficial advantage experienced by the user from this protocol is that the devices produced are affordable today. The cost of hardware is minimal and will continue to reduce in future. U2F provides speed to the user as it works on elliptic curve cryptography. User can register and authenticate the device in a matter of seconds. It can be imagined as a Smartcard that is revamped to address the modern consumer web.

The U2F protocol supplements the security of the online service by adding an extra strong second factor on top of the login username password mechanism. The device that has a built in U2F stack and support available for the web browsers to use it. In this approach, the user logs in to the service with existing username and password. Then, the service will allow user to present his second factor for authentication after successful login. The second factor is presented in the form of an action by tapping the USB security device. Now, the user can repeat the process and access the online service across all platforms that provide U2F support. Since U2F is a challenge-response protocol, every transaction will begin with the server issuing a new unique challenge. Communication from the browser over the U2F device happens through the JavaScript library that is supported by the FIDO Client [3].

II. U2F DEVICE

A. U2F Registration

The aim of this process is to register a U2F device with a user by assigning and creating a binding between the user-account and the device. It begins by the relying party (server) issuing a challenge which is in the form of a random number. This challenge combined with the Application ID (app_id) are fed into the browser. The browser on receiving the app_id and the challenge of the web-page will verify it to make sure that the app_id registered against the user is correct. After the successful verification, the browser then starts to collect information about the challenge it just received, the origin of the web-page, the U2F action being performed i.e Registration or Authentication and state about the SSL connection. It constructs a client data object from the above gathered information. This client data object together with the app_id are passed from the browser to the U2F device. The device then constructs a new user credential for the corresponding app_id and waits for the user. The user provides his acknowledgement by touching the device and a new credential is created in the form of a public key, private key and a key handle. This key handle generated is used to refer the new credential combination of K_{pub} and K_{priv}. The private key generated from the device secret key will always remain inside the device. [3] The handle and K_{pub} are returned back to the browser with the ECDSA signature on the client data. This signature ensures that the client data is not modified and tampered. After verifying this, the browser returns it back to the server. The server on receiving the data verifies it again to confirm that the challenge returned is the same that it issued.
in the beginning. Also, the correctness of the origin is verified by the SSL state that guarantees prevention against Man in the Middle (MITM) attack. The final step of the registration process concludes with the server storing the public key and the key handle that it received from the browser.

B. U2F Authentication

The aim of this process is to authenticate the U2F device for the user and prove that he is authorized to use it while logging into a service. After verifying the username and password, the server retrieves the public key and handle registered against this user credentials from the registration process. The server needs to ensure that the same user has the U2F device and it does it by matching the public key and the key handle. After successful matching, the server sends the app_id with a newly issued challenge and the registered key handle to the browser. The browser verifies the app_id same as it does in the registration process and constructs the client data object over the challenge, origin, channel id and the action performed at this point i.e "Authentication". On verification of the app_id, it is passed on to the U2F device with the client data and the key handle. The U2F device on receiving this information validates the key handle against the specific app_id. The private key is then retrieved in the U2F device by matching the app_id with the key handle. Thus, multiple credentials can be locked to a specific app_id and if the device matches them, it will use the private key to create a signature over the client data. The ECDSA signature constructed over the client data and app_id is passed to the browser with a counter that increments on every successful authentication and prevents from replay attacks [5]. The browser on receiving this information passes it back to the server. The server for the last time will again verify all the information it received from the browser and will validate the signature over the client data.

C. U2F Device Specifications

There is a full elliptic curve implementation on x & y points and the curve is frozen on the tamper-resistant chip integrated inside the U2F key. These chips are generally used for processing sensitive information that can be transmitted with the help of the private key. The U2F device generates a new ECC key-pair whenever it tries to register the device for a new service. The public key and the key handle are actually points generated on the elliptic curve secp256r1 as per the Certicom-NIST approved standards [10]. The U2F token will then use the same public key & the key handle generated from the previous step for authentication. When the device receives a request, it generates a nonce which is an arbitrary random number from RNG and can be used only once for processing it. The nonce and the app_id are then passed through a Hash Message Authentication (HMAC) in combination with the device secret key which is minted on the device by the manufacturer. The hashing algorithm used is SHA-256 and the output of this hashing function will result in the private key. This private key combined with the RNG nonce and the MAC will produce the public key and the key handle that will be eventually stored on the server. The MAC will ensure that the same key handle was registered for the intended service with the help of the private key during every successive authentication process. The security of the secret key inside the device is ensured as it can never be reset and there is no other means to communicate with the device.
III. Contribution

The U2F protocol can be simulated over a web application which requires a browser and an extension that consists of JavaScript API stack. This extension is available with pre-installed libraries and allows the U2F device to communicate using the register and sign functions. Also, the extension can be operated in developer mode enabling us to modify or extend methods, change request input parameters and verify intermediate values. While simulating the protocol over a mobile application, the browser extension gets replaced by APDU requests and responses. There are no libraries available for U2F that have pre-defined APDU methods needed for sending requests and receiving responses. As a part of the implementation, this project provides contributions in creating APDU methods that invoke requests and record responses. These methods are implemented in Java and deliver the same browser extension functionality. The APDU functions are crucial as they provide a medium of communication with the U2F device and it is now possible to register and authenticate the device using bluetooth.

IV. Implementation

The U2F protocol is implemented over the Transport layer and the APDU (Application Protocol Data Unit) layer [5]. The transport layer is responsible for breaking the packets that are transferred over the channel into specific frames. The sending and receiving of the frames between the U2F device and computer happens across the APDU layer either through the bluetooth channel or a HID (Human Interface Device). The transport layer does not support the transfer of frames over NFC (Near-Field Communication). A sample U2F packet is shown in Fig. 5.

Every U2F client communicates with the device over a logical channel and each application thus makes use of the unique channel identifier (CID) [6]. It is a 32-bit identifier that is used for routing and is discarded for bluetooth transmission. The 8-bit command (CMD) has the specific MSG type with its first bit always set and also contains the MSG number that should be invoked. If the size of the packet that needs to be sent is greater than the maximum transmission unit (MTU), it is usually divided into the maximum size limit frames. Or else if it is smaller than the MTU, it will be transferred directly without further dividing it. After the first frame is transferred...
that simply contains the truncated packet fit to the size of the frame, the rest of the remaining subsequent frames are simply the additional packets. They generally begin with a 8-bit prefix SEQ and has a zero value which increments on every successful frame transfered. The first bit of this SEQ is always set to 0 for allowing consecutive frames to be distinguished easily. A sample U2F Frame is shown below in Fig. 6.

There are five standard transport layer commands specified in the U2F specifications that are used to pass requests. The APDU request and response are included in the MSG command while the PING command simply returns the unmodified packet back to the sender. The ERROR command is used to record errors that occur while transferring data in the transport layer. The error command is a 8-bit error-code and is always sent as a response back to the issuer. The KEEPALIVE command is used for sending request only from bluetooth U2F devices and its primary aim is to identify different requests that are currently processed or wait until it gets a user confirmation. The data inside this command includes a 8-bit field which specifies the cause of the delayed response.

The INIT command contains a request of 64-bit nonce which is primarily used for allowing the U2F device to allocate a unique channel identifier and return general information from the hardware device. This command is only valid for HID devices and hence not accountable for BLE key. The response data generated from INIT is displayed in Fig. 8.

When the device receives the MSG command packet, it includes an APDU request and the host expecting an APDU reply. The APDU layer commands are similar to the transport layer commands but contain in addition a parameter via the CLS = 0X00 bytes and a miscellaneous payload. The VERSION command is represented as the UTF-8 encoding and its value is U2F_V2 and does not contain a NULL terminator. This command outputs the version supported by the U2F Raw Message Format and takes no input data or parameters. The REGISTER command does not require an input parameter but the AUTHENTICATE command takes in P1 as an input parameter.
verifying the newly created key-pair and ensuring the trust of the U2F device from the remote servers[11]. The server will validate the attestation signature from the certificate to store the combination of public key and key handle.

```javascript
u2f.register(
  'challenge': 'OdyBQ49PaqmXhrrAY5',
  'app_id': 'https://accounts.google.com/ServiceLogin?Pfue385622n'
), callback);

callback = function(response) {
  sendToServer(
    response['clientData'],
    response['tokenData']
  );
};

{ "typ": "register",
  "challenge": 'OdyBQ49PaqmXhrrAY5',
  "cid_pubkey": {
    "kty": "EC",
    "crv": "P-256",
    "x": "HzQwIFX7XQ455MtcCnZUNbw3RMZPo9tOyWjBqRl4UB",
    "y": "XyguGFLj1X1Kg3WnQf8dnb75hi4_.7-8xhMljw42Ht4"
  },
  "origin": 'https://accounts.google.com'
},

call = sendToServer(
  response['clientData'],
  response['tokenData']
);
```

Fig. 10. U2F Register Request Response

Inside the AUTHENTICATE command, the input data field requires the 256-bit challenge combined with the app_id same as the REGISTER request field. But in addition, it also requires the 0-255 bytes key handle generated from the REGISTER command. The AUTHENTICATE request operates in three different modes. P1 = 0X07 is a check only mode, P1 = 0X03 is enforce user presence and sign mode, P1 = 0X08 is do not enforce user presence and sign mode. After verifying the input data fields, the U2F then signs the payload and generates a response that contains a 32-bit counter used to identify key theft detection with a 8-bit user presence flag.

V. EXPERIMENTATION

A. U2F Web Application

The web application was developed on Eclipse Oxygen IDE in Java Servlet that generates registration and authentication signature statements similar to how a physical U2F device would generate. This application was built on a Google Engine and Apache Tomcat server that will communicate with the device through a web page developed in HTML and JavaScript. The project includes all the packages for utilities, crypto, etc and includes a self contained web server. To run the application, the U2FHttpServer.java file needs to be executed. On successful execution, the application will now be ready to listen on port 8080. Launching Google Chrome on localhost 8888 will display the demo web page. The homepage has buttons provided for registering and authenticating the U2F device. Ensure that the U2F extension for chrome is installed and configured by navigating to chrome://extensions and enabling the developer mode option. Check to see if the u2f.EXTENSION_ID in the chrome extension matches the one specified with the u2f-api.js file in the project. On clicking the “Register” button, the web page will prompt to touch the U2F device. Tapping the U2F device will perform registration and display the key handle with public key generated from the device. To authenticate these credentials, click on the “Authenticate” button and again tap the device. On successful authentication, the card that holds the key handle & the public key turns green and fades out after three seconds. Also, there is functionality provided to re-register the key that will generate different sets of public key and the key handle combinations.

B. U2F Android Application

The application was developed in Android Studio 3.0.1 with JRE 1.8.0 platform on a Windows 10 operating system.
machine. It can be deployed on any Android device that supports connectivity for Bluetooth having Android version 6.0 (Marshmallow) and above. The `activity_main.xml` is the user interface that contains the buttons and text. The actions invoked on pressing the buttons are implemented in `MainActivity.java` that makes function calls to the Register and Authenticate interfaces. Every request and response that is sent and received respectively from the device is transferred through byte array streams. To make a connection with the Bluetooth device, android Bluetooth libraries such as `BluetoothDevice` and `BluetoothAdapter` were imported [7]. The register and authenticate signatures were verified using the X509 attestation certificate and SHA256 libraries available in Java. For creating the elliptic curve parameters over secp256r1, X9EC library from `org.spongycastle` was imported. This library provides functions for generating the base point, cofactor H to the order G and N to the order G for the specified curve. All the curve related parameters remain unchanged and are used as constants. Validation and error handling was implemented to catch connection timeout exceptions.

The U2F device used for the android application is Feitian Bluetooth Low Energy (BLE) key. It can be purchased online on Amazon.com and ordered for a low cost of around 20-25 dollars depending on the vendor. Fig. 12 shows the front and back side of what the device looks like. In the front side, there are two indicators given to display if the BLE device is operating in Bluetooth or normal authenticator mode. There is a mechanical button provided that confirms user presence when pressed and acknowledges the register or authenticate action. The charging indicator displays the level of battery remaining in the device and blinks when plugged in for charging. Also, there is a micro USB port provided that can connect this device to a computer. At the back side, there is a Bluetooth device ID given similar to other Bluetooth devices that allows any user to identify or search it on the network. Once a connection has been established successfully with the BLE key, it needs to be paired by the pass code provided. After pairing the BLE key, it is now available to perform registration and authentication.

VI. RESULTS

The Android application created for the simulation of the U2F protocol has a screen that shows six buttons. The “Scan Devices” button will ask the user to turn on his mobile Bluetooth settings and will show all the devices available to connect in the nearby network. The information displayed includes date, time, MAC address and name of the Bluetooth device. The “Scan Device by Name” button provides functionality for scanning the device by name. So if a user knows the name of the device, simply typing the name and pressing OK will search the device and display if it was found or not. The “Register” button will Register the device that was selected and first validate if it is a U2F device or not. If the device selected is not a U2F device, it will simply respond with a message “device is not a U2F authenticator”. If it is, then the request for registration will be passed with the required challenge & application parameters. The device on receiving the request will generate the public key along with the key handle and display the certificate. This completes the registration step and the user can then use the fourth button.
to authenticate the device against the public key and the key handle that it just registered. After pressing the “Authenticate” button, the response is displayed that shows the user presence byte set to 1, the counter number incremented for successful authentication requests and the signature data. It also displays a signature verification message in the end. The “Disconnect” button is used for disconnecting the device and the Bluetooth connection. The “Clear Logs” button is used to erase/clear all the log information and data displayed on the screen.

VII. CONCLUSION AND FUTURE WORK

The increase in the number of attacks, security threats and phishing attempts is the primary reason for the need of having a strong second factor authentication. Other second factor authentications like OTP (One-Time password) and Google Authenticator are still vulnerable and inconvenient to use for certain users because of network or device unavailability [9]. The U2F protocol overcomes these disadvantages by offering devices that provide security at maximum level combined with a simple and easy to operate user experience. More and more organizations are in favor of adopting U2F since the cost of implementing and configuring the devices is very low. U2F is currently working on expanding its extension to Edge, Safari, Internet Explorer and other browsers. U2F’s extension from the web to mobile devices added more dynamics and attracted larger audience as it is now possible to add second factor authentication over channels such as Bluetooth or Near field Communications [8]. Apart from the channels, it supports Windows, Linux, macOS, Android and iOS operating systems over million devices on computer, laptop, tablets, iPads and mobiles. U2F plans to cross reference all the above mentioned browsers, operating systems and channels across each other in future for more feasible use and easy access.
Fig. 15. U2F Android Application Register Response

REFERENCES


