Università degli studi di Modena e Reggio Emilia

Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Laurea in Ingegneria Informatica - Sede di Mantova

Titolo: prima riga
Seconda riga
Terza riga
Quarta riga

Relatore:
Prof. Luca Ferretti

Candidato:
Giulio Barabino

Correlatore:
Ing. Federico Magnanini

Anno Accademico 2021/2022

Indice

1 Introduzione **1**

2 Conoscenze di Base **2**

2.1 Crittografia asimmetrica 2

2.2 Single Sign-On 2

2.3 Survivability 3

2.4 Passwordless 4

2.5 FIDO 5

2.5.1 Rilevamento tentativi di clonazione 5

3 Modellazione del sistema **7**

3.1 Attori 7

3.2 Flusso operativo 9

3.2.1 Fase di registrazione 9

3.2.2 Fase di autenticazione 10

4 Dettagli **12**

4.1 Modifica della libreria FIDO2 13

4.2 Modifica al codice Solo 13

Capitolo 1

Introduzione

Capitolo 2

Conoscenze di Base

In questa sezione verranno introdotti i concetti necessari per proseguire con la lettura dei capitoli successivi.

2.1 Crittografia asimmetrica

La crittografia asimmetrica è un tipo di crittografia in cui ogni attore possiede una coppia di chiavi: una **pubblica** e una **privata**. Le chiavi vengono create in funzione di un primo scambio di dati che avviene tramite crittografia simmetrica. Gli usi della crittografia asimmetrica sono due:

- Cifrare le comunicazioni: tramite la chiave pubblica del destinatario è possibile cifrare un messaggio che solo il destinatario può decifrare usando la propria chiave privata.
- Firmare digitalmente: tramite la propria chiave privata un attore può apporre su un messaggio, autenticandolo, la firma, la quale può essere verificata tramite la chiave pubblica del firmatario.

2.2 Single Sign-On

Il *Single Sign-On* è un protocollo molto diffuso utilizzato per autenticarsi a servizi Web. Il protocollo è eseguito da uno User Agent, un Identity Provider e un Service

Provider. L'User Agent è un software usato dall'utente che permette l'interazione con un contenuto Web. L'Identity Provider è un attore terzo che si occupa di fornire un sistema di autenticazione sicuro, creando e gestendo le credenziali degli utente, a servizi Web che lo richiedano. L'Identity Provider possiede degli Identity Server presso cui gli utenti devono eseguire le operazioni di autenticazione. Il Service Provider è un fornitore di servizio Web che non implementa sistemi di autenticazione propri ma utilizza quelli messi a disposizione dall'Identity Provider. Qualora l'utente si autentichi con successo presso l'Identity Server gli viene fornito un token di autenticazione firmato dell'Identity Server, che deve presentare al Service Provider per poter fruire di quel servizio. Una volta presentato il token al Service Provider sarà autenticato a tutti quei servizi messi a disposizione dal Service Provider che fanno uso dello stesso Identity Provider, da qui l'accezione *single* di *sign-on*.

2.3 Survivability

La centralizzazione dello schema SSO lascia spazio ad attacchi in cui un malintenzionato prenda il controllo dell'Identity Server e utilizzi la chiave privata dello stesso per firmare token di autenticazione forgiati arbitrariamente, così da poter poi impersonare qualunque utente egli voglia. Vengono in aiuto gli schemi cosiddetti *survivable SSO* che possono limitare tali criticità sfruttando più Identity Server. Un singolo Identity Provider gestisce quindi più Identity Server e l'utente deve autenticarsi presso un sottoinsieme di questi, i quali rilasciano poi un **token** firmato collettivamente.

La componente survivable risiede nel fatto che viene tollerato un certo numero di Identity Server violati e, di conseguenza, viene richiesto un token in funzione di questo numero. Con una soglia di tolleranza di server maligni sufficiente si riesce a garantire l'integrità del meccanismo di autenticazione e un overhead, dovuto alla reiterazione dei passaggi, trascurabile.

2.4 Passwordless

L'autenticazione passwordless è un metodo di autenticazione che permette ad un utente di effettuare il login ad un servizio senza la necessità di conoscere una password o più genericamente una conoscenza considerata segreta. Tipicamente utilizza una coppia di chiavi crittografiche, una privata e una pubblica: la prima è generata e immagazzinata sul dispositivo dell'utente, mentre la seconda è inviata al server così che esso possa verificare l'autenticità dei messaggi ricevuti. La chiave privata, o segreta, non lascia mai il dispositivo su cui è stata creata e per accedervi è necessaria l'autorizzazione ottenuta tramite **mediazione** da parte dell'utente. Un'azione è detta mediata da un utente qualora sia necessario il suo esplicito consenso. Il consenso può avvenire, ad esempio, premendo il bottone sul dispositivo fisico.

La registrazione passwordless e, conseguentemente, l'autenticazione vengono svolte seguendo un meccanismo *challenge-response*: al pervenire di una richiesta di registrazione il server invia una cosidetta *challenge*. L'utente che ha iniziato l'operazione ha il compito di apporre, tramite propria chiave privata, una firma crittografica sulla challenge e di fornire in risposta al server la challenge firmata accompagnata dalla chiave pubblica. Così facendo il server verifica l'autenticità della firma tramite la chiave appena ricevuta e in caso di esito positivo la immagazzina. La fase di autenticazione è svolta in modo analogo con la differenza che il server è già in possesso della chiave pubblica e non è quindi necessario inviarla.

Come si può vedere non viene scambiato alcun segreto e l'unica interazione richiesta all'utente è quella in fase di firma della challenge. Anche allora l'utilizzatore non deve inserire codici o password ma semplicemente mediare l'operazione tramite uno dei metodi sopra elencati. Sfruttando l'autenticazione passwordless è possibile sopperire alle criticità tipiche dei segreti a bassa entropia come le password, quali phishing, brute forcing etc.

2.5 FIDO

FIDO Alliance è un'associazione nata nel 2013 con lo scopo di migliorare i sistemi di autenticazione. Sono gli autori di *FIDO*, un set di specifiche che include gli standard **CTAP** e **WebAuthn**. Nel corso degli anni vi sono stati un susseguirsi di iterazioni degli standard, prima conosciuta come Universal Authentication Factor per poi diventare Universal 2nd Factor e giungere infine alla versione corrente FIDO 2.0.

Il protocollo CTAP definisce le specifiche tramite cui vengono programmati gli autenticatori crittografici per fare in modo che possano operare con un client. Il protocollo WebAuthn si occupa invece di definire le specifiche tramite cui standardizzare l'autenticazione a servizi web sfruttando le chiavi crittografiche.

In particolare definiscono tutto il necessario per programmare un autenticatore e un server come: strutture dati, metodi, requisiti di funzionamento, encoding dei dati etc.

2.5.1 Rilevamento tentativi di clonazione

Compito dell'Identity Server è anche quello di rilevare eventuali tentativi di duplicazione dell'autenticatore fisico. Per fare ciò lo standard FIDO prevede un **contatore**, sia esso globale o multiplo, aggiornato dall'autenticatore ad ogni operazione avvenuta con successo. Il contatore prende il nome di *signature counter*. Il server mantiene in memoria l'ultimo valore ricevuto e, all'interazione successiva, controlla che non vi siano discrepanze. Ad esempio, ipotizzando di avere allo stesso tempo:

- Un autenticatore originale con contatore pari a m
- Un autenticatore clone dell'originale con contatore pari a m

Se l'autenticatore originale si autentica presso un servizio, questi aggiorna il proprio contatore a $m + 1$. Il clone, tentando di autenticarsi allo stesso servizio, fornisce un valore del contatore pari a m, dunque minore di quello salvato in memoria dal server al momento dell'ultima interazione con l'autenticatore originale. In questo modo, il server riconosce il clone in quanto tale.

Ne consegue che il rilevamento del tentativo di clonazioni basato sul contatore risulta:

- Inefficace finché il clone non procede ad autenticarsi
- Fallace se il clone procede ad autenticarsi prima dell'originale: quest'ultimo viene di fatto invalidato nonostante sia legittimo

Capitolo 3

Modellazione del sistema

In questo capitolo verranno inizialmente descritti gli attori che concorrono alle operazioni di registrazione/autenticazione per poi definire il flusso delle operazioni stesse.

3.1 Attori

Lo schema di seguito rappresenta lo stato attuale dello standard FIDO in accordo all'implementazione descritta successivamente.

Figura 3.1: PLACEHOLDER.

Il protocollo include i seguenti attori: l'utente, Service Provider, Identity Provider, un numero n di Identity Server, l'autenticatore hardware e il FIDO Client.

Il **Service Provider** è un fornitore di un generico servizio a cui l'utente è interessato ad accedere. Può essere un qualunque servizio di streaming, banking, shopping etc. il quale fa uso di un intermediario per l'autenticazione dei propri utenti. Questo può avvenire per varie ragioni sia economiche che legate alla sicurezza. Il Service Provider definisce il **security level** per indicare il livello di *survivability* desiderato, cioè il numero di Identity Server necessari a completare le operazioni di autenti-

cazione/registrazione. Tale valore è un intero positivo e viene stabilito in funzione della confidenzialità del servizio erogato.

L'**Identity Provider** è un ente terzo che si occupa di fornire a un Service Provider il servizio di autenticazione. A tale scopo si avvale di n Identity Server. Compito dell'Identity Provider è anche quello di fornire il token di autenticazione al client da presentare al Service Provider per fare in modo che l'utente possa accedere al servizio scelto. I vari Identity Server memorizzano le credenziali degli utenti ed una chiave segreta con cui autenticare attestazioni di identità.

L'**autenticatore** hardware è un dispositivo che, tramite l'interazione con l'utente, permette l'accesso al servizio web richiesto. L'autenticatore si occupa di memorizzare le credenziali. Ogni credenziale contiene una chiave privata, metadati e un **array associativo** di contatori arr_C. L'arr_C associa ad un determinato *security level* il valore di un *contatore* intero per tenere traccia delle operazioni effettuate con successo. L'autenticatore ricorre alla chiave privata per firmare le challenge che gli sono sottoposte e invia la chiave pubblica al server così che esso possa verificare l'autenticità delle firme. Per la fase di creazione e le successive di autenticazione viene richiesto all'utente di compiere un'azione: essa può essere la pressione di un pulsante sulla chiavetta stessa, un collegamento NFC oppure ancora l'identificazione tramite impronta digitale. Così facendo l'operazione in corso viene autorizzata.

Il FIDO **Client** è un dispositivo che sfrutta un **User Agent** conforme ad implmentare le specifiche FIDO per il dialogo con l'Identity Server e con l'autenticatore, in collaborazione con l'hardware sottostante su cui è installato l'User Agent, tipicamente un sistema operativo. Il client è quindi interposto tra il FIDO Server e l'autenticatore fisico, agendo da intermediario. Il suo compito è duplice:

- Comunicare con il server al fine di iniziare, e successivamente terminare, le operazioni di autenticazione e di creazione delle credenziali
- Comunicare con l'autenticatore allo scopo di creare le chiavi crittografiche e firmare le challenge ricevute dal server

La comunicazione è bidirezionale e segue i protocolli definiti dagli standard: CTAP per l'interazione con l'autenticatore e WebAuthn per la comunicazione con il FIDO

Server. Nel caso particolare dell'estensione survivable il client si occupa di replicare le operazioni su n FIDO Server distinti, computando l'hash delle challenge ricevute e fornendolo all'autenticatore come un digest unico su cui apportare la firma.

3.2 Flusso operativo

Il flusso operativo si compone di due operazioni distinte: una fase di creazione delle credenziali e una fase di autenticazione dell'utente. Tali operazioni vengono svolte, rispettivamente, durante la registrazione al servizio del Service Provider e a tutti le autenticazioni successive.

3.2.1 Fase di registrazione

Alla fase di registrazione partecipano: l'utente, il FIDO Client, lo User Agent, n Identity Server, l'autenticatore.

La fase di registrazione si origina a partire dalla richiesta dell'utente, utilizzando un User Agent, di registrarsi ad un servizio, offerto da un Service Provider, che supporti l'autenticazione passworldess, tramite degli Identity Provider. Il Service Provider fornisce all'utente il livello di sicurezza n necessario per completare l'operazione. La fase di creazione dovrà essere replicata dal client su tutti gli n Identity Server presenti. Il processo messo in atto è il seguente:

1. Ogni Identity Server crea il proprio stato interno e la challenge
2. Ogni Identity Server invia al Client una serie di requisiti secondo cui deve essere svolta la cerimonia di registrazione, e la challenge generata
3. Il Client salva tutte le challenge ricevute in un vettore e computa l'hash dello stesso
4. Il Client effettua una chiamata al metodo opportuno dell'autenticatore, fornendo i requisiti di creazione richiesti dall'Identity Server e il digest computato come challenge

5. L'autenticatore procede a generare la coppia di chiavi crittografiche seguendo le imposizioni del server e invia al client la challenge firmata accompagnata dalla chiave pubblica e il contatore specifico dell'arr_C inizializzato a uno.

6. Il Client invia ad ogni Identity Server il vettore con le challenge, l'hash dello stesso, la firma, la chiave pubblica e il signature counter ricevuto

7. Ogni Identity Server controlla che la challenge da lui generata sia presente all'interno del vettore e controlla, computando lui stesso l'hash del vettore, l'integrità di quanto ricevuto. Infine, verifica tramite la chiave pubblica fornitagli l'autenticità della firma.

8. Qualora il processo sia andato a buon fine, gli Identity Server salveranno le informazioni ricevute (contatore, chiave pubblica, identificatore del client) al proprio interno

3.2.2 Fase di autenticazione

La fase di autenticazione ricalca i passaggi di quella di registrazione con la differenza che gli Identity Server sono già in possesso della chiave pubblica con cui verificare l'autenticità della firma apportata alle challenge.

In questa fase viene comunicato un security level pari a $|Q|$ da parte del Service Provider, cioè la cardinalità del sottoinsieme di Identity Server $Q \leq n$ presso cui è necessario autenticarsi. Questo valore prende in considerazione la tollerabilità alle intrusione che ha il Service Provider. In particolare: $|Q| \in [(2k+1), (3k+1)]$, dove k rappresenta il numero di Identity Server di cui si può tollerare la compromissione.

1. Ogni Identity Server crea il proprio stato interno e la challenge

2. Ogni Identity Server invia al Client una serie di requisiti, secondo cui deve essere svolta la cerimonia di autenticazione, e la challenge generata

3. Il Client salva tutte le challenge ricevute in un vettore e computa l'hash dello stesso

4. Il Client effettua una chiamata al metodo opportuno dell'autenticatore, fornendo i requisiti di autenticazione richiesti dall'Identity Server e il digest computato come challenge

5. L'autenticatore incrementa il contatore dell'arr_C specifico. Invia poi al Client la challenge firmata e il contatore aggiornato

6. Il Client invia ad ogni Identity Server il vettore con le challenge, l'hash dello stesso, la firma ricevuta e il signature counter

7. Ogni Identity Server controlla che la challenge da lui generata sia presente all'interno del vettore e controlla, computando lui stesso l'hash del vettore, l'integrità di quanto ricevuto. Infine, verifica tramite la chiave pubblica, memorizzata precedentemente, l'autenticità della firma

8. Ogni Identiy Server controlla che il *signature counter* ricevuto sia maggiore di quello memorizzato in precedenza e in tal caso aggiorna quest'ultimo con il valore appena ricevuto

9. Se i passaggi precedenti sono avvenuti con successo rilasciano al Client l'attestazione tramite cui può completare l'autenticazione presso il Service Provider

Capitolo 4

Dettagli

In questo capitolo verranno trattati i dettagli implementativi relativi alle modifiche operate a:

- Una variante della libreria FIDO2 realizzata da Yubico modificata in una tesi precedente per accogliere il meccanismo survivable [5]
- Il codice sorgente dell'autenticatore Solokeys [4] per integrare il *security level*

Nonostante la libreria Yubico fosse già stata modificata in precedenza per adottare la struttura survivable, è stato comunque necessario operare cambiamenti. La libreria FIDO2 si occupa di simulare l'interazione tra un Client FIDO2 e un Server FIDO2 per emulare la registrazione e la successiva autenticazione. Grazie alla modifica apportata precedentemente è possibile simulare un numero arbitrario di Server con cui stabilire la comunicazione e svolgere tali operazioni. Viene correttamente gestito tutto il funzionamento descritto nel capitolo precedente meno la parte di invio del security level.

Lato autenticatore invece si è reso necessario implementare diverse funzionalità: dal parsing del security level nel messaggio inviato dall'User Agent all'autenticatore fino ad arrivare a un contatore globale vero e proprio. Infatti, da standard FIDO2 il signature counter utilizzato per il controllo della clonazione dell'autenticatore può essere anche globale e non specifico per credenziale [3]. Ciò è dovuto alla natura *constraint*, cioè con limitazioni di memoria importanti, degli autenticatori hardware.

4.1 Modifica della libreria FIDO2

La libreria FIDO2 presenta il file `client_multichallenge.py` in cui viene definita la classe `Fido2ClientMultichallenge`, figlia della classe `Fido2Client` nel relativo `client.py`, che permette l'autenticazione WebAuthn simulando un Client FIDO. Presenta due metodi, `make_credential` per realizzare l'operazione di creazione delle credenziali e `get_assertion` per compiere l'operazione di autenticazione.

Rispetto alla condizione di partenza sono stati aggiunti ai metodi sopracitati il passaggio del parametro `security_level`. Questo valore viene poi passato alla funzione `send_cbor` chiamata in concatenazione. Quest'ultima funzione si occupa di creare la sequenza di byte codificata in *CBOR*, inviarla all'autenticatore e attenderne la risposta.

4.2 Modifica al codice Solo

Il codice dell'autenticatore Solokeys, come detto in precedenza, presentava un solo contatore globale per tenere traccia di tutte le operazioni di creazione/autenticazione svoltesi con successo. Il primo passo è stato, quindi, quello di implementare una struttura dati per la memorizzazione di un contatore per credenziale. Tale struttura dati, definita `signCounter` è una tipo di dato composto da due valori.

- Una istanza `id` della *struct* `CredentialId`
- Un intero senza segno di 32 bit definito come `signCount`

La struttura dati `CredentialId` definisce come vengono memorizzate le credenziali all'interno dell'autenticatore seguendo lo standard FIDO [2]. In particolare, invece che avere una sequenza di 16 bytes come da standard, presenta valori come

```
tag, nonce, padding, metadata, rpIdHash
```

La definizione della struttura `signCounter` è effettuata all'interno del file `ctap.h` e sempre nello stesso file è inizializzato anche `signCounterArray`, cioè un array di strutture dati `signCounter`. Tramite questo array è possibile memorizzare un contatore per credenziale.

L'array viene popolato all'interno nel momento in cui viene chiamata la funzione `ctap_make_credential` utilizzata dall'autenticatore per compiere le operazioni di creazione delle credenziali. In particolare:

- tramite un intero non segnato `globalCounter` viene tenuta traccia dell'ultima posizione dell'array occupata
- viene istanziata una struttura dati `signCounter` con `id` pari a quello calcolato e `signCount` = 1
- viene aggiunta la struttura dati `signCounter` all'array `signCounterArray` alla posizione `globalCounter`

Nel momento in cui viene chiamata la funzione `ctap_get_assertion` viene cercata iterativamente la struttura dati il cui `id` corrisponde a quello dell'operazione corrente e viene incrementato il contatore di tale struct.

Per verificare che l'`id` corrisponda a quello generato in fase di `get_assertion` è stato necessario scrivere una funzione di comparazione: il C, infatti, non supporta nativamente la comparazione di due struct. A tal scopo è stata scritta la funzione `count_cmp_func` che compara attributo dopo attributo tutti quelli presenti all'interno della struttura per verificarne l'uguaglianza. Il contatore, invece, viene aggiornato tramite la funzione `update_sign_counter` che semplicemente prende il valore `signCount` passato in chiamata e restituisce $signCount + 1$.

Il passo successivo è stato quello di modificare la struttura dati `signCounter` per fare in modo che accogliesse un contatore per *security level*. Per fare ciò l'attributo `signCount` è stato cambiato da intero senza segno di 32 bit ad array di interi senza segno di 32 bit. Analogamente al processo di incremento seguito prima, il contatore corrispondente al security level ricevuto verrà aggiornato nel seguente modo:

```
update_sign_counter(signCounter.signCount[n]])
```

Così facendo vengono mantenuti e incrementati n signature counter differenti per ogni credenziale.

L'ultimo passaggio è stato quello di ricezione del *security level*. Lo standard CTAP2 [1] definisce i codici dei comandi a cui deve essere associato il lancio di alcune funzioni. Ogni comando è strutturato con il proprio codice di comando e i codici per i propri parametri. I codici per per i comandi di interesse sono i seguenti:

```
0x01 authenticatorMakeCredential
0x02 authenticatorGetAssertion
```

L'autenticatore Solo sfrutta il polling per controllare l'arrivo di messaggi da parte del Client. Al giungere di uno di questi viene controllato il codice del comando presente nei primi byte e viene invocata la funzione indicata dal codice e di cui è stato mostrato un esempio sopra. Tale funzione a sua volta effettuerà il parsing, cioè l'analisi del contenuto, per ottenere i parametri della funzione invocata. Oltre ai codici per i parametri già esistenti dei comandi `authenticatorMakeCredential` e `authenticatorGetAssertion` è stato necessario aggiungere nel file `ctap.h`:

```
0x0A MC_securityLevel
0x08 GA_securityLevel
```

Per definire i codici con cui codificare i livelli di sicurezza da usare, rispettivamente, in fase di creazione credenziali (**M**ake**C**redential) e autenticazione (**G**et**A**ssertion).

Solo utilizza delle funzioni definite nel file `ctap_parse.c` per fare il parsing del flusso di dati CBOR ricevuto dal Client. In particolare sono presenti due funzioni:

- `ctap_parse_make_credential`
- `ctap_parse_get_assertion`

che si occupano di controllare il flusso di dati per ottenere i parametri necessari alle funzioni `ctap_make_credential` e `ctap_parse_credential` per poter compiere le operazioni di creazione/autenticazione. In questo caso viene aggiunto allo `switch` l'identificazione dei codici definiti sopra per il parametro *security level*.

Di seguito il funzionamento semplificato all'interno della funzione `ctap_parse_make_credential`:

```
switch(cmd)
{
    ...
    case MC_securityLevel:
        cbor_value_get_int(MC->securityLevel)
    ...
}
```

Al riconoscere del codice relativo al parametro security level definito sopra, viene salvato il valore del security level all'interno della struttura dati `MC`, ovvero il risultato costruito con i parametri ottenuti dal flusso di dati ricevuto dal Client. Analogo quanto avviene all'interno della funzione `ctap_parse_get_assertion`.

In questo modo all'interno delle funzioni `ctap_make_credential` e `ctap_parse_credential`, grazie al parsing effettuato, è disponibile il valore `securityLevel` per poter incrementare il contatore adeguato.

Bibliografia

[1] FIDO Alliance. CTAP2 Commands, 27 February 2018.

[2] FIDO Alliance. Credential Id definition, 8 April 2021.

[3] FIDO Alliance. Signature counter considerations, 8 April 2021.

[4] Solokeys. Solokeys/solo1: Solo 1 firmware in C.

[5] Yubico. Yubico's FIDO2 Python Implementation.