識別器を用いた生体認証の強靭化

大崎 康太[1]　八槇 博史[1]

概要：現代の生体認証は鍵などベースの認証に必要なシークレットを取得するクライアントで完結するケースが現れている．しかし，サーバはクライアント側の認証過程を把握することができない．認証デバイスは統計モデルや関数を作成するためのパラメータを保存する．今後クライアントで完結する認証の提案が増えることを想定すると、パラメータを差し替えることで正当なデバイスの所有者にもかかわらず，なりすましが行われる可能性がある.本研究は，ソフトウェアベースの機械学習モデルを識別器と想定し，同じ識別器が同じデータを入力した際同じ値を出力する性質に着目した.クライアントで解決する生体認証の問題を防ぐため,鍵を取り出すために用いる識別器を認証要素に加えたチャレンジレスポンス方式の認証基盤を提唱する．

キーワード：認証　認証プロトコル FIDO チャレンジレスポンス　識別器

Enhancing Biometric Authentication By Classifier Verification

Kota Osaki[1]　Hirofimi Yamaki[1]

Abstract: There is a case where modern biometrics authentication is completed by a client that obtains a secret required for key-based authentication. However, the server cannot grasp the authentication process on the client side. Assuming that the number of authentication proposals that will be completed by the client will increase in the future, there is a possibility that spoofing will be performed by exchanging the parameters despite the legitimate device owner. In this study, we assumed a software-based machine learning model as a classifier and focused on the property that the same classifier outputs the same value when the same data is input. We propose a challenge-response type authentication infrastructure in which a discriminator used to retrieve is added as an authentication factor.

Keywords: Authentication　AuthenticationProtocol　FIDO　ChallengeResponse　Classifier

1. はじめに

現代の生体認証は鍵などベースの認証に必要なシークレットを取得するクライアントで完結するケースが現れている．しかし，サーバはクライアント側の認証過程を把握することができない．認証デバイスは統計モデルや関数を作成するためのパラメータを保存する．今後クライアントで完結する認証の提案が増えることを想定すると、パラメータを差し替えることで正当なデバイスの所有者にもかかわらず，なりすましが行われる可能性がある.本研究は，ソフトウェアベースの機械学習モデルを識別器と想定し，同じ識別器が同じデータを入力した際同じ値を出力する性質に着目した.クライアントで解決する生体認証の問題を防ぐため,鍵を取り出すために用いる識別器を認証要素に加えたチャレンジレスポンス方式の認証基盤を提唱する．

2. 背景

2.1 従来の生体認証

従来の生体認証は，生体情報の特徴パターンをデータベースに保存し，専用のハードウェアで入力された特徴と照合することで認証を行っている．

X_I ユーザーに対応した特徴ベクトル
X_Q 入力ベクトル
I　アイデンティティ
S　X_QとX_Iを入力し類似度を返す関数
t 事前に定義された閾値

$$(I, X_q) \in \begin{cases} w1 \; if \; S(X_Q, X_I) \geq t \\ w2 \; otherwise \end{cases}$$

生体認証システムは下記の四つのモジュールによって設計される.[1]しかし，認証時に使用する生体情報を直接扱う形になるため，奪取された際の危険性が大きい．

1. 指紋などの生体情報を取得するモジュール
2. 生体情報から静的情報を抽出するモジュール
3. あらかじめデータベースに保存されたテンプレートと

1 東京電機大学　情報環境学研究科
Graduate School of Information Environment, Tokyo Denki University

抽出したベクトルを照合するモジュール
4. ユーザーの生体情報やアイデンティティを登録するデータベース

2.2 現代の生体認証

現代の生体認証では，2018 年の Apple の Face ID[2]等モバイル端末を使った生体認証技術が増えつつある[2]．持ち運びやコスト，普及率などの利点が大きく今後焦点が当てられると予想される． FIDO UAF[3]では，クライアント内で生体認証が完結し，サーバのやり取りには従来の公開鍵でのチャレンジレスポンスが使われており，ベースとなる認証のシークレットを取り出すために使われているケースもある．

3. 課題

前述の背景を踏まえて，今後起きる可能性のある課題をまとめる．FIDO[3]の様にクライアントで生体認証が完結するような生体認証が増えていくことを想定する．そして，クライアントの端末内に生体認証を行う識別器とクラウドなどのサービスとやりとりや識別器への命令を送るプログラムがあるとする．本研究では識別器を入力ベクトルからパラメータを更新した関数と定義する．例えば，Tensorflow[3]等の機械学習モデルや確率密度関数などである．

次に攻撃者の想定をする．攻撃者はユーザーの端末にアクセスできる．スマートフォンのロックを解除できる．偽のクライアントを用意する等が考えられる．そして，プログラム自体に攻撃しない．なぜなら，プログラムはハードウェアベースで書き込まれている可能性があるからである．ここで攻撃するのはメモリ上に格納された識別器を構成するパラメータである．あらかじめ，攻撃者用のデータセットを用意する．又は，パラメータ自体を用意し差し替える．パラメータを差し替えることで攻撃者は，差し替えた端末から本来の利用者のアカウントやサービスにアクセスできてしまう．サーバはクライアント間で完結した認証の結果をもとに認可を行うからである．

4. 関連技術

本研究での背景，実装に関連する認証技術について説明する．

4.1 FIDO

FIDO 認証[3]は，端末内の FIDO Authenticator 内の秘密鍵と Relying Party 内の公開鍵を使ってチャレンジレスポンス形式の認証である．図は，FIDO 認証の principal と一連の処理の概要である．メッセージのやり取りは UAF protocol[4]を使いユーザーのデバイスと Relying Party 間で行う．FIDO client は UAF protocol[4]から受信したメッセージによって認証や登録，アカウントの削除の処理を行う．また，client は，UAF protocol[4]に乗せるメッセージの構成などを行う．FIDO metadata service では，送られてきたメッセージ内の変数をもとに鍵の正当性を示す．この正当性を示す機能をアテステーションと呼び， ユーザーの正当性に対する緩衝策を行っている．

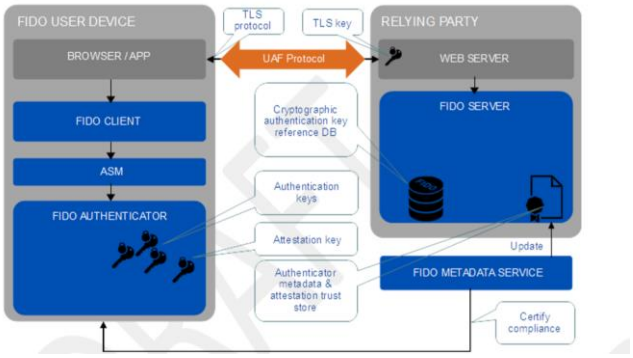

図 1 FIDOUAF の処理 出典[3]

アテステーション[5]について説明する． アテステーションは， 認証時ではなく登録時に行われる． 初めに，デバイスの製造時に固有の秘密鍵を用意する． 例えば，galaxy8[4]のデバイスがあるとする． それらはすべて同一の鍵と証明書を持つ．ユーザーがアカウントを作成した際に生成される公開鍵に対してアテステーション用の秘密鍵を使って署名をする．アテステーションの証明書は従来の電子署名と同様にルート証明書と紐づけることによって信頼関係を形成する．また， 登録時にデバイスの固有の番号を公開鍵と一緒にサービスに送信をする． この番号を AAID と呼び， 認証時などに MDS と呼ばれるメタデータを管理するサーバとの照合で用いる．

2 FaceID https://support.apple.com/ja-jp/HT208109
3 Tensorflow https://www.tensorflow.org/lite?hl=ja
4 https://www.galaxymobile.jp/galaxy-note20/

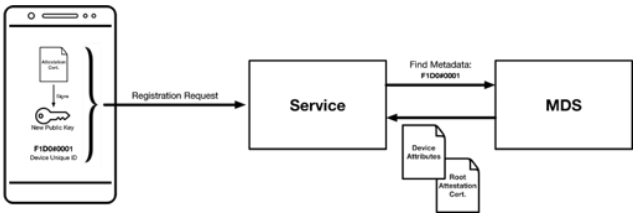

図 2　アテステーションの処理　出典[5]

4.2　Kerberos[6]

当時，暗号鍵をいかに効率よく利用するかという課題があった．なぜなら，利用者が様々なソフトウェアを利用するようになったからである．鍵を取り出したままにするのは，安全面での問題がある．しかし，都度パスワードを要求し，取り出していては，ユーザーにとって負担になる．そのため，KDC という第三者機関を導入したプロトコルが提案された．これが Kerberos である．Kerberos はセッション鍵，第三者機関，nonce 等を使ってユーザーの認証を行うことでマスター鍵の取り出しを減らし，攻撃者に権限を奪われた際の被害を最小限にとどめた．KDC は鍵の管理，間接認証を行う．Kerberos は，Needham-Schroeder プロトコルを採用することでチケットを使った認証を可能にし，鍵利用の効率化を行った．チケットは一時的な共有鍵とシークレットのデータ構造である．また KDC 間でのチケット認証を行うことによって複数の組織の連携を可能にし，OAuth の考えの基盤となった．

5.　識別器を加えた認証の提案

前述の課題を解決するため，本研究では識別器を使った認証を提案する．以下，ハッシュ値での照合方法とチャレンジレスポンス形式での照合方法について述べる．

5.1 識別器の定義と性質

本研究では識別器をソフトウェアベースの機械学習モデルを識別器と想定する[7]．具体的には Tensorflow[3] 等の学習モデルファイルと定義する．識別器には，学習したデータとネットワークの組み合わせ等から固有のアイデンティティとしてできると考える．また，学習モデルは，同じデータが入力された時，それに対応するデータを出力する性質を持つ．この性質や特徴から，ソフトウェアベースでの生体認証を行う際，認証の要素として加えることができると考えた．

5.2 前提

ハッシュの形式，チャレンジレスポンス形式での環境の前提について述べる．クライアントとサーバ間で識別器を共有していることを前提とする．

5.3 ハッシュ値での照合方法[7]

ハッシュ値での検証の想定である．利用者は PC に搭載している Web カメラなどを使い顔画像のデータを識別器に入力し分類を行う．その後，識別器又は識別器のハッシュ値をサーバに送信する．サーバは保管している識別器をハッシュ値に変換し照合しレスポンスを送る．

X_i クライアントの識別器
X_s サーバに保管された識別器
H ハッシュ関数

$$(X_i, I) \in \begin{cases} 1 & H(X_i) = H(X_s) \\ 0 & else \end{cases}$$

5.4 チャレンジレスポンス形式[8]

チャレンジレスポンス形式での想定である．サーバはリクエストを受け取った後，ランダムな数値のバイナリデータをチャレンジとして返す．クライアントは受け取ったデータを ndarray に変換し，識別器に入力する．出力値をレスポンスとしてサーバに送信する．サーバは識別器にチャレンジとして生成したデータを入力し，レスポンスと照合することで認証を行う．

X_i チャレンジ
C_U　ユーザーの識別器
C_S　サーバの識別器

$$S\big(C_U(X_i), C_S(X_i)\big) \in \begin{cases} 1 \; if \; C_U(X_i) = C_S(X_i) \\ 0 \; else \end{cases}$$

5.5　実装で使用する方式

ハッシュ値とチャレンジレスポンス形式のそれぞれについて述べた．実装で使用する方式について検討する．本研究ではチャレンジレスポンス形式での実装を選んだ．ハッシュ値でのやり取りの場合，ハッシュ値をペイロードにのせているため，ハッシュ値そのものを奪取される可能性を考えられるからだ．

5 Github のコードを参考に拡張した
https://github.com/oauthinaction/oauth-in-action-code

6. 実装

課題に対するアプローチが実現で切ることを示すため，実装を行った．有益性を強く示すことを考慮し，既存の認証プログラムを拡張した．以下，Kerberos[6]の拡張とOAuth[9]の拡張について述べる．

6.1 OAuth での拡張[8][9]

OAuth のコード[5]を使った拡張を行った．以下，データやプログラムの詳細，構築環境，拡張の処理を示す．

6.11 データとプログラム

実装で使用した，拡張元のプログラム，拡張プログラムについての解説をする．また，プログラム間で渡すデータも示す．

表 1　拡張元プログラム

プロセス名	説明
Client.js	ユーザーがブラウザ上で操作を行い，クライアント側の情報を AuthorizationServer.js に認証に必要な情報を送信する
AuthorizationServer.js	Client.js から送られたメッセージをもとに認証を行い，アクセストークンを発行する．
ProtectedResource.js	送られてきたアクセストークンをもとにサービスを提供する

表 2　拡張プログラム

プロセス名	説明
AuthClient.py	ユーザーが識別器の登録か認証かを選択し，それぞれに対応した機能を実行する．
Test_sesrver.py	識別器の作成や登録，学習や検証に必要なデータの作成を行う．

表 3　user 変数

変数名	説明
Id	ユーザーに対応した固有の識別子
Code	識別器の登録か認証かを判断するための文字列 “account” 識別器の登録 “Auth”　認証

表 3 は，AuthClient.py から client.js に送信される user 変数である．識別器の登録や認証のどちらを行うか判断するためのメッセージとなる．表 4 は client 変数内の変数の詳細である．

```
Var user = {
        String : id
        String : code
}
```

表 4 client 変数

変数名	説明
client_id	ユーザーに対応した固有の識別子 AuthorizationServer.js と認証を行う際に使う変数
client_secret	ユーザーに対応したシークレット 識別器の出力結果によって更新される変数
redirect_uris	AuthorizationServer.js が開放している IP アドレスにリダイレクトするための URI

表 5　データセット

変数名	説明
Data	Numpy 型の変数のリスト Test_server.py で作成され識別器の登録や検証時に入力される
Label	data に対応した 2 値のリスト

dataset 変数は Test_server.py が作成する識別器の学習，検証用のデータセットである．クライアント側でも同様のデータを入力するために Auth_Client.py に送信する変数である．

```
dataset = {
        List: data,
        List: label
}
```

表 6 clients 変数

変数名	説明
client_id	client.js から送信された client_id と照合を行うための変数
client_secret	client.js から送信された client_sercret と照合を行うための変数 識別器の出力結果によって更新される変数でもある.
redirect_uris	認証を行った後, client.js にコールバックするための URI
Scope	認証後,サービスの認可を行うための変数

6.12 構築環境

構築では，拡張用の認証プログラムと拡張元の JavaScript のコードを使って拡張を行った．以下それぞれの環境を記載する．

認証サーバ

使用言語
　Python 3.6.5
ライブラリ
　requests2.184　メッセージの送信
　numpy1.143　チャレンジの生成
　Keras2.24　識別器の作成や検証

認可サーバ（拡張元のサーバ）
使用言語
　JavaScript
　Node.js

6.13　登録時の処理

AuthClient.py が識別器登録のコマンド入力を受け取った際, ID の入力を促す．その後入力された ID を Client.js に送信し, id とメッセージを json 形式で Test_server.py に送信する．Test_Server.py はデータと対応するラベルを作成し，学習処理を行い,識別器の作成と保管を行う．その後，AuthClient.py と識別器を共有する．

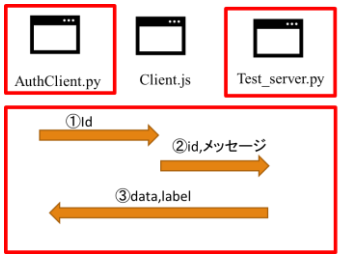

図 3　登録時の処理　出典[8]

6.14　認証時の処理

Authclient.py が ID を Test_Server.py に送信する．送信後 ID に対応した識別器を取り出し，ランダムな数値列のデータとラベルを生成し，入力を行う．その後損失値を出力し Client.js と AutohorizationServer.js のシークレットを損失値の値に更新する．次に data と label を AuthClient.py に送信する．同様に AuthClient.py 側でも識別器を取り出しデータを入力する．出力結果を AuthorizationServer.js に送信し，照合を行うことで認証を行う．その後は従来の OAuth[9]の方式と同様であり，アクセストークンを取得し，取得したアクセストークンをもとに ProtectedResource.js から必要なデータやサービスを取得する．

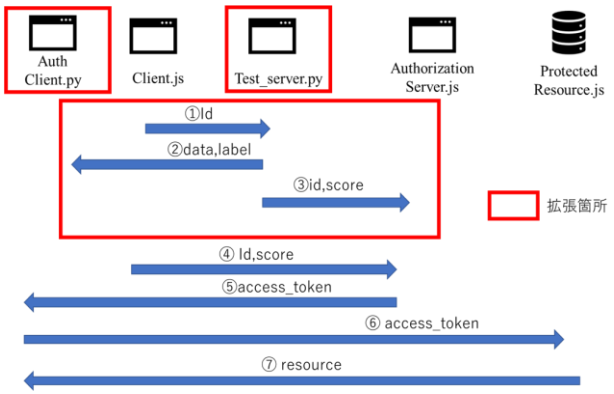

図 4　認証時の処理　出典[8]

6.15 処理時間

実装したプログラムを動かし， 認証のコマンド入力後からシークレットの更新までの時間を計測した．計測には time 関数を使用した．認証にかかる平均時間が 600ms 程度である．

6.2　Kerberos での拡張[6]

Kerberos[6]サーバを構築し，そのサーバの一部分を拡張する計画を行った．以下，実装理由，サーバの構築環境，拡張想定図を示す．

6.21 実装理由

これまでは OAuth[9]のサンプルコード[5]を使い実装を行ってきた．これによりアプリケーションの範囲の対応が可能になった．今後，ユーザーのログイン，データベース等のミドルウェアの領域の対応を考えた．法人向けに使われている Kerberos[6]サーバを構築することでさらに有益性を示すため実装に至った．

6.22 サーバ構築環境

拡張元の Kerberos[6]サーバを構築する必要がある．構築は以下の環境で行った．

OS CentOS7[6]
サーバ　srv5
クライアント　srv7
プロセス　krb5kdc kadmin sshd

srv5 はチケットを配布するための KDC とユーザーを登録するための kadmin を構築し，設定する．srv7 では krb5.conf のアクセス先を srv5 に設定する．次に srv5 の kadmin にアクセスしユーザーの登録, keytab の生成し，パスワードを登録する．また, srv7 のチケットを使って srv5 がアクセスすることを想定している．そのため，srv7 の sshd.conf の GSSAPIAuthentication の設定を変更する．これにより srv5 から srv7 へのシングルサインオンを実装した．

6.23 拡張想定

6.22 の環境をもとに拡張していく．そのための図を示す．図は拡張の想定である．srv5 を kdc とクライアントサーバーが存在する．srv7 をコンテンツと定義する．２のレスポンスのペイロードに TGT チケットに加えてバイナリの数値データをチャレンジとして加える．その後クライアントサーバーではバイナリデータを ndarray データに変換する．その後，識別器を取り出し，変換したデータを入力し検証を行う．出力された損失関数を③のリクエストに加える．kdc サーバでは損失関数のデータを照合し，合致する場合 TGS レスポンスを返すよう処理を拡張する．

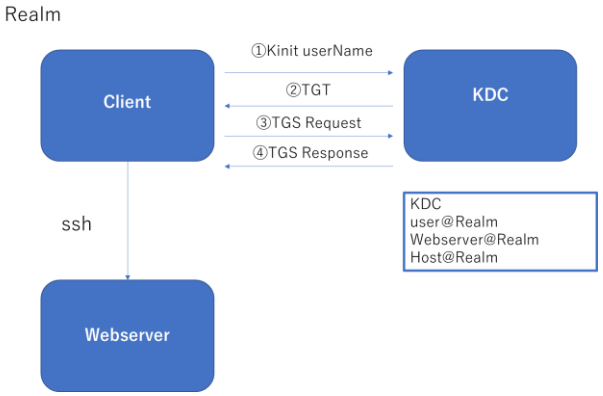

図 5　Kerberos サーバの拡張想定図

7. 拡張方式

拡張する際に考慮する要素がある．既存のメッセージやコードを拡張してデータ野受け渡しを処理する方法がある．また，新しいメッセージを用意し，拡張する方法がある．本研究では，アプリケーション層プロトコルの拡張に関する論文をもとにそれぞれの欠点を述べる．阿部[10]の論文では，プロトコルの拡張を行うと，拡張前の差分への対応から生じる保守性の維持の難しさについて述べている．また拡張した際メッセージのやり取りや状態遷移，プロトコルの挙動が分かりにくくなり処理部の安全性やコードの可読性の低下が挙げられる．河野[11]の論文では，拡張時のエラーチェックや例外処理の記述の手間や既存のメッセージからの拡張に比べて同じようなプログラムの記述をしてしまうといった点が挙げられる．

8. おわりに

本研究は，生体認証の将来可能性のある攻撃として、デバイス内の識別器のパラメータのすり替えによるなりすましを想定した．これに対応するため，識別器をモデルファイルとして認証要素に加える認証基盤を提案した．そして，チャレンジレスポンス形式を採用し，OAuth[9]と Kerberos[6]ベースでの拡張方法についてそれぞれ述べた．今後は Kerberos[6]ベースの実装を行うため，拡張するコードを見極める．そのため，Kerberos[6]サーバとクライアント感でのパケットのやり取りを WireShark[7]等でパケット分析を行い，拡張箇所を特定し実装を行っていく．

謝辞

本研究を進めるに当たり，指導教員の八槇博史教授から

6 https://www.centos.org/

7 https://www.wireshark.org/download.html

多くの助言を賜りました．厚く御礼申し上げます．

参考文献

[1] J. K. Anil, R. Arun , P. Salil, “An Introduction to Biometric Recognition,” IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 14, NO. 1, 2004.
[2] Abhijit Das, Chiara Galdi, Hu Han, RaghavendraRamachandra, Jean-Luc Dugelay , AntitzaDantcheva. (2018). Recent Advances in Biometric Technology for Mobile Devices. IEEE.
[3] FIDO Alliance. (2017 年 11 月 28 日). FIDO UAF Architectural Overview. 参照日: 2020 年 8 月 14 日, 参照先: https://fidoalliance.org/specs/fido-uaf-v1.2-rd-20171128/fido-uaf-overview-v1.2-rd-20171128.html
[4] FIDO Alliance. (2017 年 11 月 28 日). FIDO fido-uaf-protocol-v1.2. 参照日: 2020 年 8 月 14 日, 参照先: https://fidoalliance.org/specifications/download/
[5] FIDO Alliance. (2018 年 7 月 29 日). FIDO TechNotes: The Truth about Attestation. 参照日: 2020 年 8 月 14 日, 参照先: https://fidoalliance.org/fido-technotes-the-truth-about-attestation/?lang=ja
[6] J.Kohl, C.Neuman. (1993 年 9 月). Request for Comments : 1510 The Kerberos Network Authentication Service (V5). 参照日: 2020 年 8 月 14 日, 参照先: https://www.ipa.go.jp/security/rfc/RFC1510-00JA.html
[7] 大崎康太, 八槇博史. (2019). 生体認証における識別器の検証機構. 情報処理学会全国大会第 81 回.
[8] 大崎康太, 八槇博史. (2020). 識別器検証による生体認証の強靭化. 電子情報通信学会 2020 年総合大会.
[9] D. HardtEd. (2012 年 10 月). Request for Comments: 6749 The OAuth 2.0 Authorization Framework. 参照日: 2020 年 8 月 14 日, 参照先: https://openid-foundation-japan.github.io/rfc6749.ja.html
[10] 阿部 勝幸 , 岩崎 英哉 , 河野 健二 . (2005). アプリケーション層プロトコルの記述に基づく 拡張性に優れたプロトコル処理コード生成系. 日本ソフトウェア科学会第 22 回大会（2005 年度）論文集.
[11] 河野健二. (2003). アプリケーション層プロトコルの実現を容易にするフレームワーク. 情報処理学会論文誌, Vol. 44, No. SIG2, pp.25‐36.