学術情報処理研究 No.20
2016年9月

既存のサービスシステムの変更を不要とする認証シャッターの提案

A Proposal for design and implementation of an authentication shutter without modification of existing service systems

本村真一 †, 川村尚生 ‡

Shin-ichi Motomura†,Takao Kawamura‡

motomura@tottori-u.ac.jp,kawamura@ike.tottori-u.ac.jp

鳥取大学総合メディア基盤センター †
鳥取大学大学院 工学研究科 情報エレクトロニクス専攻 ‡
Center for Information Infrastructure & Multimedia, Tottori University†
Department of Information and Electronics,
Graduate School of Engineering, Tottori University‡

概要

情報システムの多くは、ログイン認証としてユーザ ID と固定パスワードを採用している。しかしながら、固定パスワードによる認証はフィッシング等によるアカウント情報の漏洩により不正アクセスの被害を受けやすい。この対策として、ワンタイムパスワードや 2 要素認証など認証システムを変更する強化策が提案されている。一方近年、固定パスワードによるログイン認証を強化する方法として認証シャッターが提案されている。これは、ユーザが認証シャッターの開閉操作を行い、認証シャッターが開放されている時だけ認証を実施することで不正アクセスを防止するための仕組みである。本論文では、これまで提案されているものよりもユーザの利便性を高め、また既存のシステムの変更を不要とする認証シャッターとその実装を提案する。

キーワード

認証シャッター, 2 要素認証，アクセス制御

1 はじめに

情報システムの多くは、ログイン認証としてユーザ ID と固定パスワードを採用している。しかしながら、固定パスワードによる認証はフィッシング等によるアカウント情報の漏洩により不正アクセスの被害を受けやすい。FIDO Alliance[1] の UAF[2] など、固定パスワードを用いない認証方法も提案されており、Windows10 において実装が予定されているがまだ普及しているとは言い難い。固定パスワードによるログイン認証の強化策としては 2 要素認証などが提案されており、本学においても乱数表を用いた 2 要素認証（以下、「マトリクス認証」という。）を採用している。これはファルコンシステムコンサルティング株式会社の Wisepoint Shibboleth[3] を用いて実現している。そのため、moodle や Active!mail[4] など Shibboleth に対応したシステムについては 2 要素認証により認証の強化を実現しているが、Shibboleth に対応していない Web システムやメール等の Web 以外のシステムにおいては固定パスワードによる認証のみである。そこで学外からこれらのシステムを利用する際は、

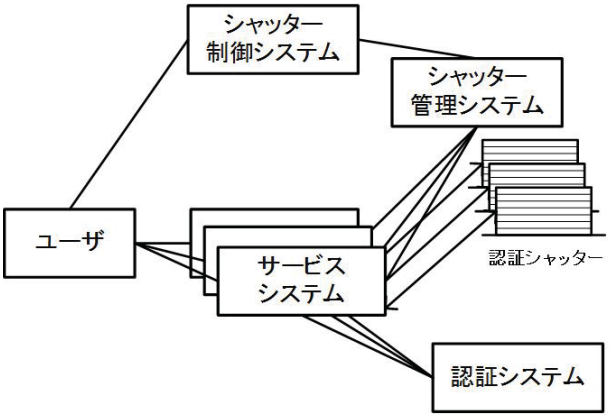

図- 1: 認証シャッターの概略図

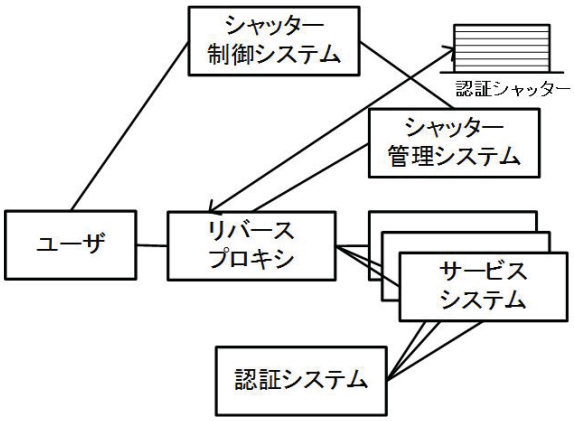

図- 2: 提案する認証シャッターの概略図

Shibboleth 認証に対応した VPN 装置を用いて 2 要素認証を実現しているが、ユーザにとっては VPN ソフトウェアのインストールが負担であることや VPN ソフトウェアの利用が煩雑である等の問題を抱えていた。特にメールについてはメールソフトにこだわりがあるユーザも多く、VPN に代わる方法が求められていた。一方、近年このような固定パスワードの認証強化策として認証シャッター [5, 6] が提案されている。これは、認証シャッターが開いているときのみログインを可能とし、認証シャッターが閉じているときはログインを拒否することで不正ログインの防止を図る仕組みである。図 1 に認証シャッターの概略を示す。ユーザは目的のサービスシステムを利用する前に、シャッター制御システムと通信して認証シャッターの開放操作を行う。シャッター制御システムはユーザの認証シャッターの開放情報をシャッター管理システムに登録する。その後、ユーザがサービスシステムに対してユーザ ID と固定パスワードによるログイン操作を行うと、サービスシステムはシャッター管理システムと通信を行い、認証シャッターが開放されている時だけログインを許可する。不正ログイン防止のために、シャッター制御システムへの通信においても何らかの認証が必要である。この認証に固定パスワード以外のものを用いることで、サービスシステム利用時の 2 要素認証も実現できる。

提案されている認証シャッター [5, 6] では、サービスシステムもしくは認証システムを変更することで実装できる旨述べられているが、これらのシステムの変更が困難な場合もある。多くの商用ソフトウェアでは改変することができず、またオープンソースソフトウェアにおいても変更箇所のメンテナンスという負担が発生する可能性がある。認証システムにおいては、LDAP サーバのような統合認証システムを導入していることが多く、変更作業の影響を考慮するとその変更は容易ではないと考えられる。また、サービスシステム毎に認証シャッターを設置することになっているが、ユーザにとっては操作が煩雑になると考えられる。また、[6] では Web システムを対象として提案されているが、メールシステムは利用者が多く、セキュリティを考慮すると認証シャッターを適用すべきだと考える。

本論文では、既存のサービスシステムや認証システムを変更することなく導入できる認証シャッターとその実装を提案し、サービスシステムへの適用例を用いてその有効性を示す。

2　新しい認証シャッターの提案と実装

本論文の認証シャッターでは、既存のサービスシステムや認証システムを変更することなく認証シャッターが導入できるよう、図 2 に示すようにサービスシステムに対するリバースプロキシを配置した。このリバースプロキシがサービスシステムに代わりシャッター管理システムと通信を行うことで、目的のサービスシステムへの通信可否を制御する。なお、図ではリバースプロキシ毎に認証シャッターを設置しているように見えるが、実際には複数のリバースプロキシがあっても認証シャッターはひとつで良い。本実装では Web システムではないメールシステムについても対象としており、その際検討しなければならないのは認証シャッターの開放時間である。認証シャッターを開放している間はサービスシステムへのログインが許可されているため、長時間開放することは不正アクセス対策の観点から好ましくない。これまでの提案 [5, 6] では、認証シャッターの開放時間が短くなるよう、手動での認証シャッターの閉鎖やログイン後自動的に閉鎖する運用について述べられている。しかしながら、多くのメールソフトでは POP3 や IMAP4 通信のために頻繁にログイン処理を行っている。また、メールシステムの利用は長時間におよぶことがあり、認証シャッターの頻繁な開閉操作はユーザの負担となる。そこで本実装では、認証シャッターの開放時間を比較的長く保つとともに、認証シャッターの開放情報に併せて

接続元 IP アドレスを保持することで、ユーザ単位でのアクセス制御付認証シャッターとしている。接続元 IP アドレスを保持したことと、認証シャッターを全サービスシステムで共有していることから、本実装における認証シャッターの状態を得るためのキーは、これまでの提案 [5, 6] の「ユーザ ID とシステムサービス」ではなく「ユーザ ID と IP アドレス」のペアへ変更している。

図 2 のシャッター管理システム、シャッター制御システム、リバースプロキシの詳細を以下に示す。

2.1 シャッター管理システム

認証シャッターの状態は、ユーザ ID と IP アドレスのペア（以下、「Skey」と言う。）をキーとして、状態を示す値を用いて表す。これはちょうどキーバリューストアとして実装できる。キーバリューストアには高速なインメモリデータストアの実装がいくつかあるが [7, 8]、サーバの停止などに対応するため永続化機能を持った redis[9] を採用している。認証シャッターの開放状態の登録は、redis へ Skey をキーとして数字の 1 を登録する。認証シャッターの状態を確認する場合、redis にキーとして Skey を与え、 1 が取得できれば認証シャッターが開放されている、値が取得できず FALSE が返されると閉じていると判断する。なお、redis に Skey を登録する際は、認証シャッターが自動的に閉じるよう 2 時間をキーのタイムアウト時間としている。この時間は、不用意に認証シャッターが閉じないよう、Outlook や Thunderbird など各種メールソフトにて設定できるフェッチ時間を確認して定めた。これらのソフトは任意の時間が設定できたが、iOS のメール App では設定できる最長時間が 1 時間であったことから、これより長い時間として採用している。

2.2 シャッター制御システム

シャッター制御システムは PHP による Web システムとして実装しており、ユーザからのリクエストに応じて redis に Skey を登録する。不正アクセスを防止するためには、シャッターの開放操作についても何らかの認証が必要であり、これまでの提案 [5, 6] においてもそれぞれの認証方法が提案されている。本実装ではクライアント証明書等による認証を想定しており、本学では既に提供していたマトリクス認証、クライアント証明書、Time-Based One-Time Password（以下、「OTP」という。）認証の 3 つの方法を提供している。Web サーバとて apache を利用しているため、いずれの認証方法においても apache の環境変数から接続元 IP アドレスが取得できるが、ユーザ ID の取得方法は異なる。マトリクス認証は、Shibboleth SP を設定することで apache の環境変数からユーザ ID が取得できる。クライアント証明書の場合は、CN に登録しているユーザ ID を利用する。OTP 認証では、認証時にユーザ ID と OTP を入力するためこれを利用する。

2.3 リバースプロキシ

リバースプロキシには nginx[10] を採用している。nginx はオープンソースの Web サーバソフトウェアであるが、HTTP だけでなく様々なプロトコルのプロキシ機能も実装されている。また、リバースプロキシサーバとして機能させる際、Web サーバと通信を行うことで独自の認証を実装することができる。ここでは、redis を参照する PHP プログラムを作成することで独自の認証を実装し、シャッター開放時のみサービスシステムへの接続を可能としている。

3 サービスシステムへの認証シャッターの適用

メールシステムと Web システムに対して認証シャッターを適用した例を示す。ここでは、本学で運用しているメールシステム及び ownCloud[11] とサイボウズガルーン（以下、「ガルーン」という。）[12] を取り上げる。なお、ownCloud は有償の Enterprise 版においては Shibboleth に対応しており、Shibboleth を用いた 2 要素認証が実現できるが、本学では無償で利用できる Community 版を使用しているため 2 要素認証によりログイン機能を強化することができない。3rd party apps を用いて OpenAM と連携することで Wisepoint Shibboleth の 2 要素認証機能を使うことができると考えられるが、ownCloud のデスクトップ用同期ソフトウェアである Desktop Client[13] が利用できないため利便性を損なう。ガルーンはセッション認証機能を用いることで Shibboleth により認証することもできるが、スマートデバイス用アプリであるサイボウズ KUNAI[14] が利用できず、こちらも利便性の低下を招く。

3.1 メールシステム

postfix と dovecot を用いて SMTP、POP3、IMAP4 に対応したメールシステムを提供している。nginx では SMTP、POP、IMAP へのログイン時に、次のように auth_http ディレクティブを設定することで、指定した Web サイト（以下、「認証用 Web」という。）にて独自の認証を行うことができる。

```
mail {
    servername xxx.tottori-u.ac.jp;
    auth_http localhost:9000/cgi-bin/mailauth.php
}
```

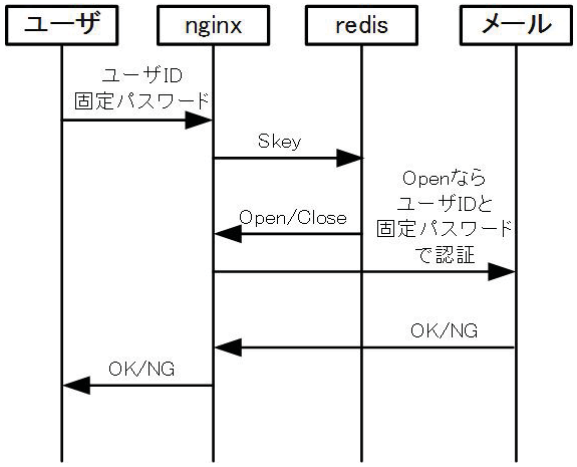

図- 3: 認証シャッターを利用したメールシステムへのログインフロー

図 3 に認証シャッターを用いたメールシステムへのログインフローを示す。このログインフローについて、以下に設定例を含めて述べる。下記は imap ログイン時に nginx が認証用 Web サイトへ送信する HTTP ヘッダの例である。auth_http ディレクティブに設定された mailauth.php プログラムは、HTTP ヘッダの Auth-User からユーザ ID、Client-IP から接続元 IP アドレスを受け取り Skey として redis から認証シャッターの開放状態を取得する。

```
GET /cgi-bin/mailauth.php HTTP/1.0
Host: localhost
Auth-Method: plain
Auth-User: user
Auth-Pass: password
Auth-Protocol: imap
Auth-Login-Attempt: 1
Client-IP:  xx.xx.xx.xx
```

シャッターが開放状態であった場合、mailauth.php は Auth-Protocol を元に接続先メールサーバを Auth-Server、ポート番号を Auth-Port に設定し、Auth-Stats を OK として次のような HTTP ヘッダを nginx へ返す。その結果、nginx により先の HTTP ヘッダに設定したメールサーバへプロキシ接続される。なお、各メールプロトコルにおける認証はメールサーバにより実行される。

```
HTTP/1.0 200 OK
Auth-Status: OK
Auth-Server: imap.yy.tottori-u.ac.jp
Auth-Port: 143
```

また、認証シャッターが閉じているときは Auth-Status に OK 以外を設定して応答すると、nginx はメールソフトに対して認証エラーを応答する。この時、ユーザにはアカウント認証のエラーなのか認証シャッターが閉じているためのエラーなのか区別ができないため、注意事項として周知する必要がある。

```
HTTP/1.0 200 OK
Auth-Status: Invalid login
```

メールソフトは比較的長時間利用することがあるため、mailauth.php は認証シャッターの開放状態の確認時に Skey が redis に登録されていた場合、Skey のタイムアウト値を 2 時間後に再設定している。このことにより、ユーザは認証シャッターの開放作業を頻繁に行う必要がない。移動時など断続的にメールソフトを利用する際も、IP アドレスが変わらなければ認証シャッターの開放操作が必要ないため利便性が良い。

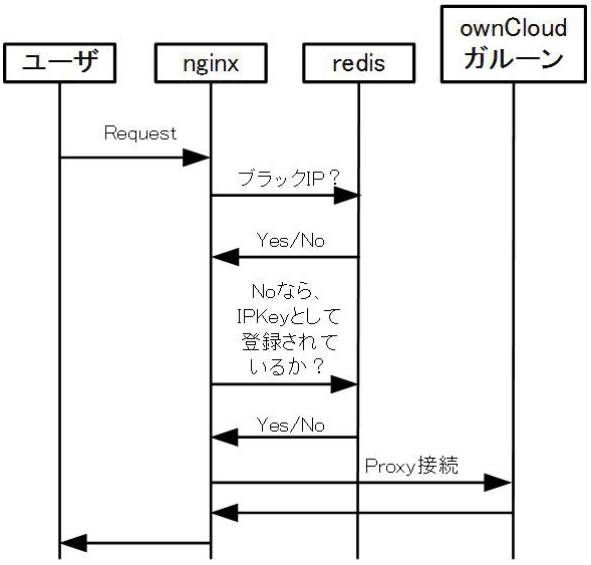

図- 4: 認証シャッターを利用した ownCloud 及びガルーンへの接続フロー

3.2 Web システム：ownCloud とガルーン

メールシステムにおいては、各プロトコルにログイン処理が含まれているためリバースプロキシにてログイン時に追加の認証を行うことができるが、多くの Web システムでは独自にログイン処理を実装しているため、リバースプロキシにて Web システムのログイン処理を拡張することは困難である。多くの Web システムと同じく、ownCloud 及びガルーンにおいても、HTTP の POST メソッドによりユーザ ID と固定パスワードを送信することでログイン処理が行われる。リバースプロキシでは、ログイン処理に利用されるユーザ ID と Skey に含まれるユーザ ID が同一であるかを検証することができず、このままでは認証シャッターを用いることができない。また、ログイン処理が完了していることを示すセッション情報を cookie に保存する Web システムもあるが、セッション情報は Web システムが独自に管理している情報である。検証用の API が提供されていない Web システムでは、リバースプロキシにて cookie のセッション情報の正当性を検証することは困難であり、認証シャッターの情報として採用することは対象が限定されることとなるため難しい。 そのため Web システムにおいてはサービスシステムへのログインの可否ではなく、サービスシステムへの接続制限を行なっている。これは認証シャッターにおいて、ユーザ ID を用いず IP

アドレスだけで制限することになるため、そのままではメールシステムに対する認証シャッターと比べて不正アクセスが成功する可能性の増加をまねく。そこで、サービスシステム上で認証ログをチェックするプログラムを稼働させておき、ログインが成功したユーザ ID と接続元 IP アドレスを Skey として redis から認証シャッターの開放状態を取得する。redis に Skey の登録がなければ不正接続と判断し、接続を拒否すべき IP アドレス（以下、「ブラック IP」と言う。）として当該 IP アドレスを「black-IP アドレス」という形式で redis にタイムアウト値を 20 分として登録する。この方法への対応のため、シャッター管理システムは redis に Skey を登録する際、IP アドレスをキーとしての数字の 1 を登録している（キーとして登録する IP アドレスを以下では「IPKey」という。）。図 4 の接続フローに示すように、サービスシステムへの接続を判断するプログラムでは、接続元 IP アドレスが redis にブラック IP として登録されていれば接続を拒否し、登録がなければ IPKey として登録されているか確認する。IPKey として登録されていれば接続を許可する。この方法では最初のリクエストは許可されるが、続くリクエストを拒否することができる。なお、ブラック IP の登録による接続拒否は Web システムへのログイン成功後に行われるため、ログイン失敗を伴うアカウントリスト攻撃などへの対策としては、ロックアウトや fail2ban など既存の対策方法の併用が考えられる。

nginx を用いたサービスシステムへの接続制限は、ngx_http_auth_request_module モジュール [15] を用いて実現している。このモジュールを用いることで、Web サーバもしくはその一部への接続可否を別の Web リクエストの結果を用いて判断することができる。このモジュールを用いた設定例を以下に示す。

```
location / {
  auth_request /_auth/challenge;
  proxy_set_header    X-Forwarded-For
              $proxy_add_x_forwarded_for;
  proxy_pass https://xxx.tottori-u.ac.jp;
}

location  /_auth/challenge {
  internal;
  proxy_pass_request_body off;
  proxy_set_header Content-Length "";
  proxy_set_header Host $http_host;
  proxy_set_header    X-Forwarded-For
              $proxy_add_x_forwarded_for;
  proxy_pass
      http://auth.xxx.tottori-u.ac.jp/auth.php;
}
```

location /ブロックに設定された auth_request ディレクティブの効果により、Web サイトへ接続されると設定された同サイトの/_auth/challenge へ nginx から新たなリクエストが発行される。/_auth/challenge へのリクエストは、proxy_pass ディレクティブに設定された Web サイトへ渡され auth.php が実行される。auth.php が HTTP ステータスコードとして 200 番台を返すとクライアントからの接続は許可され、401 もしくは 403 を返すと接続は拒否される。接続が許可されると、location /ブロックに設定された proxy_pass ディレクティブにより目的のサービスシステムへプロキシ接続される。auth.php は HTTP ヘッダ X-Forwarded-For から接続元 IP アドレスを受け取り、redis の登録情報から接続可否を判断する。なお、HTTP ヘッダはクライアントにより送付できるため、nginx へリクエストが届いた段階で X-Forwarded-For が付与されている可能性がある。その場合、X-Forwarded-For ヘッダの値は追加されることになる。そこで、クライアントからの偽装を避けるため、最後に追加された値を利用している。

3.2.1 認証ログチェックプログラム

認証ログをチェックするプログラムは、サービスシステムが異なってもログフォーマットを調整する以外は基本的に同じである。考慮すべきこととして、リバースプロキシを用いているため接続元 IP アドレスの取得について工夫が必要である。上述の auth.php のように自作のプログラムであれば、接続元 IP アドレスを HTTP ヘッダ X-Forwarded-For から取得することもできるが、サービスシステムは Web サーバの環境変数から接続元 IP アドレスを取得している。そのため、Web サーバにおいては HTTP ヘッダから接続元 IP アドレスを取得するように設定する必要がある。本学の ownCloud 及びガルーンは apache を利用しており、バージョン 2.4 以降の apache では mod_remoteip モジュールを使用し、次のように RemoteIPHeader ディレクティブを設定することで実現できる。

```
RemoteIPHeader X-Forwarded-for
RemoteIPInternalProxy xx.xx.xx.xx
```

ownCloud では、ログイン失敗のログは出力されるがログイン成功のログは出力されない。サービスシステムに対する変更は行わない方針ではあるが、ログイン成功時にログを出力する変更だけ行った。ownCloud パッケージの lib/private/user/manager.php には checkPassword という関数が定義されており、この関数中の認証が成功した処理部分に次のログ出力用プログラムを追加している。なお、manager.php の checkPassword 関数は、確認した範囲ではバージョン 7.0.2 から最新の 9.0.4 まで変更は発生していなかった。

```
\OC::$server->getLogger()->warning
  ('Login success: \''. $loginname .
   '\' (Remote IP: \''.
   \OC::$server->getRequest()->
   getRemoteAddress(). '\')', ['app' => 'core']);
```

また、ownCloud に WebDAV 接続している場合は、Web サーバのログに BASIC 認証のユーザ ID を出力することができる。接続成功時のログに含まれるユーザ ID と IP アドレスも認証ログチェックに利用できるため、ownCloud においては ownCloud が出力するログと Web サーバのログの両方を用いてチェックを行っている。なお、ガルーンについてはガルーンが出力するログ

表- 1: 仮想化環境のスペック

ソフトウェア	VMware 社製 vSphere5.5
サーバ	HP 社製 Proliant DL360 Gen9
CPU	Xeon E5-2680 v3 2.50GHz 2 個
メモリ	196GB
ネットワーク	10GbE
ストレージ	NetApp 社製 FAS8020 (NFS 接続)

表- 2: nginx 及び redis サーバのスペック

OS	CentOS 7.2
CPU 数	1
メモリ	2GB
ネットワーク	10GbE

表- 3: 学外からの接続環境

OS	OS X 10.11.5
CPU	Intel Core i5 1.6GHz
メモリ	4GB
Wi-Fi ネットワーク	IEEE 802.11n 2.4GHz
通信回線	Mineo D プラン

イン成功ログに含まれるユーザ ID と IP アドレスを用いている。

3.3 不正アクセス実験

認証ログチェックプログラムは perl の File::Tail 拡張を用いて実装している。ログをチェックする間隔を 1 秒間としており、不正アクセスがあった場合、各サーバの負荷状況により異なるが、redis にブラック IP が登録されるまで 1-2 秒程度要する。Web ブラウザによる接続の場合は、発行されるリクエストが多いことやユーザインターフェースの都合上待ち時間が多いため不正な利用はできないと考えて良いが、API などの接続方法においては最小のリクエストを連続して送付することができるため、ある程度不正アクセスが成功すると考えられる。そこで確認のため次の実験を行った。redis に Skey は登録せず、接続元 IP アドレスを IPKey として登録し、学外のネットワークから ownCloud 及びガルーンのそれぞれへ接続してどの程度リクエストが受け付けられるか計測した。nginx、redis、ownCloud 及びガルーンは全て仮想マシンとして作成している。これら仮想マシンと仮想化環境のスペックを表 1 と 2 に示す。なお、ownCloud とガルーンのスペックはメモリ以外表 2 と同じであり、それぞれに割り当てているメモリは 8GB、4GB である。また、実運用している ownCloud については [16] に詳細を記述している。学外からの接続環境については表 3 に示す。

ownCloud は WebDAV により接続できることから、curl コマンドにより 1KB のファイルの取得を 4 回連続して実行した。これを 5 回繰り返したところ、平均して 1.6 のリクエストが受け付けられた。

ガルーンでは Garoon API[17] により接続できるため、ログイン後に連続して 5 回スケジュールを取得するプログラムを PHP で作成し実行した。スケジュールの取得は、ひとつだけ予定が登録されている日の 0 時 0 分から 23 時 59 分を指定している。これを 5 回繰り返したところ、平均して 3.2 のリクエストが受け付けられた（最初のログイン用リクエストを含む）。なお、ガルーンにおいてファイルではなくスケジュールを取得しているのは、学外からガルーンに対してファイル管理へ接続できないよう設定しているためである。

nginx のログより、リクエストが拒否されるのはいずれの実験結果においても最初のリクエストを受け付けたあと 1 ないしは 2 秒後であったが、ownCloud の WebDAV 接続とガルーンの API 接続では 1 秒間に処理できるリクエスト数が異なるためこのような違いが生じている。実験結果から、接続元 IP アドレスが許可されいる、固定パスワードが既知である、取得対象の情報が分かっている、という 3 つの条件を満たせば情報を不正に取得される可能性があることが分かる。これらの条件を満たす不正アクセスについて考察すると、攻撃者は固定パスワードと取得対象の情報が必要であるが、不正アクセス時にこれらの情報を入手するのは難しい。固定パスワードの場合は、規定回数のログイン失敗によるロックアウトなどがあり、取得対象の情報の調査については上記実験のような数回のリクエストでは入手が困難である。攻撃者は事前にこれらの情報を入手したうえで不正アクセス行うことになるため、想定される攻撃の多くは標的型になると考えられる。不正アクセスは認証シャッターが開放されたネットワークから行う必要があり、可能性の高い環境としては店舗、空港、イベント会場など多人数が利用する NAT 環境が挙げられるが、攻撃者は標的の活動状況を把握するか偶然に頼ることになるため、実際に攻撃を成功させるのは困難であると考えられる。

4 まとめ

本論文では、認証シャッターを実装しメールシステムや Web システムに適用することで、既存のサービスシステムや認証システムにほとんど変更を加えることなく不正アクセスへの対策が図れることを示した。本学では、メールソフトを用いてメールサーバへ接続するため

に VPN 接続を提供しているが、認証シャッター提供後約 1ヶ月で 25%程度のユーザが移行している。移行数は増加傾向にあり、今後の周知活動に伴いさら移行が進むと考えている。また、これまで提供していた VPN ソフトウェアではインストールや起動が必要なため利用を敬遠していたが、認証シャッターの方が簡易に利用できることから学外からの接続を行うようになったユーザもおり、利便性が向上していることが分かる。

認証シャッターによるメールシステムへのアクセス制限は、ユーザ ID と IP アドレスのペアになっているため、大規模な NAT 環境などグローバル IP アドレスを複数人で共有している環境では、固定パスワードが既知の攻撃者からは不正アクセスを受ける可能性がある。そこで追加の対策として、メールソフトにより使用する暗号スイートが異なることを利用し、ユーザが普段使用しているメールソフトの暗号スイートと異っていた場合は接続を制限する対策も検討している。

Web システムにおいては、IP アドレスによる接続制限後に認証シャッターの開放状態を確認しているため、グローバル IP アドレスを複数人で共有している環境ではメールシステムよりも不正アクセスを受ける可能性が高くなる。利用を Web ブラウザに限定すれば、認証シャッター開放時にブラウザとシャッター管理システムに認証トークンを保存しておき、リバースプロキシによる接続可否の判断時に認証トークンを利用することで不正アクセスを防止することができる。そこで、Desktop Client やサイボウズ KUNAI の利用よりもセキュリティを優先するユーザに対して、認証シャッターの動作を選択できるようにすることも検討している。

参考文献

[1] FIDO Alliance. https://fidoalliance.org/,(参照 2016-07-19).

[2] UAF. https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-protocol-v1.0-ps-20141208.html, (参照 2016-07-19).

[3] Wisepoint shibboleth. http://wisepoint.jp/wpshibboleth.html,(参照 2016-07-19).

[4] Active!mal. http://www.qualitia.co.jp/product/,(参照 2016-07-19).

[5] 高田哲司. Authentication shutter:個人認証に対する攻撃を遮断可能する対策の提案. コンピュータセキュリティシンポジウム 2014 論文集, 第 2014 巻, pp. 883–890, oct 2014.

[6] 多田充. パスワード認証の強化策. 学術情報処理研究, No. 19, pp. 40–49, Sep 2015.

[7] memcached. https://memcached.org/, (参照 2016-07-19).

[8] Kyoto Tycoon. http://fallabs.com/kyototycoon/, (参照 2016-07-19).

[9] redis. http://redis.io/,(参照 2016-07-19).

[10] nginx. https://nginx.org/en/,(参照 2016-07-19).

[11] ownCloud. http://owncloud.org/, (参照 2016-07-19).

[12] サイボウズ ガルーン. https://garoon.cybozu.co.jp/,(参照 2016-07-19).

[13] Desktop client. https://owncloud.com/products/desktop-clients/, (参照 2016-07-19).

[14] サイボウズ KUNAI. http://products.cybozu.co.jp/kunai/, (参照 2016-07-19).

[15] ngx_http_auth_request_module. http://nginx.org/en/docs/http/ngx_http_auth_request_module.html,(参照 2016-07-19).

[16] 本村真一, 川戸聡也, 木本雅也. 教育・研究用情報システムにおけるオブジェクトストレージの活用. 学術情報処理研究, No. 19, pp. 26–34, Sep 2015.

[17] Garoon API. https://cybozudev.zendesk.com/hc/ja/categories/200157760-Garoon-API,(参照 2016-07-19).