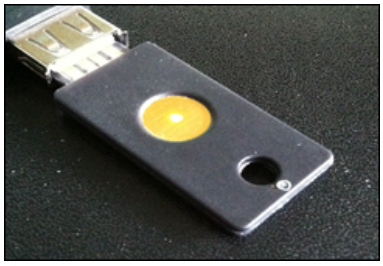
Two-Factor Authentication using Yubikey, YubiCloud and BIG-IP LTM

Jason Rahm, 2013-31-05

Two-factor authentication (hereafter 2FA) has been a staple in enterprise VPN environments for quite some time, but it is really taking off in the web application space now as well with services riding on smart phones like Google Authenticator and YubiCloud, which we've built solutions for before in George Watkin's Google Authenticator with BIG-IP APM and Brett Smith's Yubikey Authentication with BIG-IP APM. This solution borrows from Brett's in that it uses the same 2FA (the Yubikey) service in this two-factor authentication story but it diverges quickly, however, as this solution does not take advantage of the built-in support for authentication that APM provides.

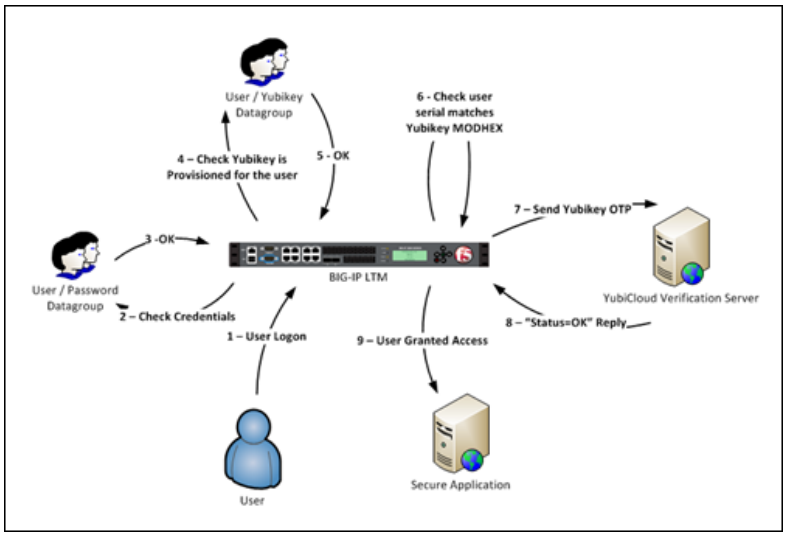
What is a Yubikey?

Directly from Brett's article: "The YubiKey is an innovative USB-key that simplifies the process of logging in with strong two factor authentication. With a simple touch on the device, it generates a One-Time Password (OTP) on any computer and platform without any client software needed. By touching the integrated button, the YubiKey sends a time-variant, secure login code as if it was typed in from a keyboard. The unique passcode is verified by a YubiKey compliant web service or software application" utilizing BIG-IP LTM and the YubiCloud validation service.



Authentication Process

The authentication process is very similar in steps to the BIG-IP APM solution, but the details vary as will be discussed below.

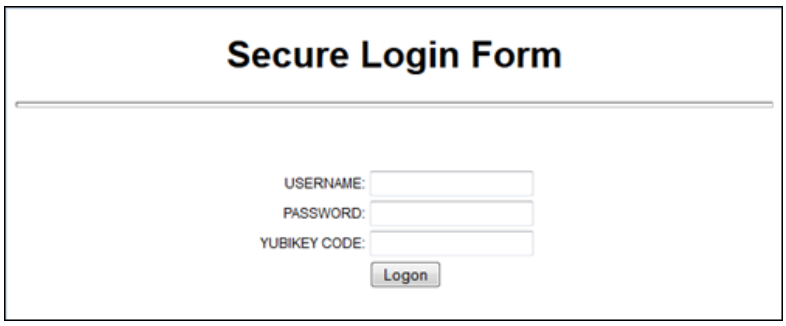


Step 1

The user is presented a form that I have stored and presented from an iFile. The form code and resulting screen cap is below.

```
<html>
<head>
<title>Login</title>
<style type="text/css">
<!--
body,td,th {
    font-family: Geneva, Arial, Helvetica, sans-serif;
}
.style4 {font-size: 12px}
-->
</style>
```

```
</head>
<body><center>
    <h1>Secure Login Form</h1>
    <hr size=5>
    <p> </p>
<form action="" method="post" name="loginForm" id="loginForm">
      <table width="262" border="0">
        <tr>
          <td width="102"><div align="right" class="style4">USERNAME:</div></td>
          <td width="150"><input name="username" type="text" id="username"></td>
        </tr>
        <tr>
          <td><div align="right" class="style4">PASSWORD:</div></td>
          <td><input name="password" type="password" id="password"></td>
        </tr>
        <tr>
          <td><div align="right" class="style4">YUBIKEY CODE:</div></td>
          <td><input name="otp" type="text" id="otp"></td>
        </tr>
        <tr>
          <td> </td>
          <td><input type="submit" name="Submit" value="Logon" style=".style4"></td>
        </tr>
    </table>
        <p> </p>
</form>
</body>
```



Step 2 & 3

At the beginning of this effort, I started by using http basic auth for step one, then presenting a form for the one time password, but that seemed bad form when I could collect it all in one step, so I reformatted the iRule to use the form above. This section of the iRule handles the form presentation (as well as the http collection for processing the data from the form post.) The iFile form contents are returned to the user once the redirect sends them to the login form, and when the post is received, the iRule begins collecting http payload.

```
when HTTP_REQUEST {
  if { not ([HTTP::path] starts_with "/Login_form") } {
    if { [HTTP::cookie exists $ckname] } {
      set cookie_payload [HTTP::cookie value $ckname]
      set decryptedCookie [AES::decrypt $aeskey [b64decode $cookie_payload ]]
      if { not ( $decryptedCookie equals "" ) } {
        # retrieve the auth status from the session table
        set auth_status [session lookup uie $decryptedCookie]
      }
      # If the auth status is 0 then the user is authenticated
      if { $auth_status eq 0 } {
        #Cookie Decrypted & Session Auth valid
        set forceauth 0
      }
    }
    if {$forceauth eq 1} {
      set orig_uri [HTTP::uri]
      HTTP::redirect "/Login_form?req=$orig_uri"
    }
  } else {
    # If the user is re-directed to the login form then serve the login form from
    if { [HTTP::path] starts_with "/Login_form" && [HTTP::method] equals "GET" } {
      # Retrieve the login form from ifile
      HTTP::respond 200 content [ifile get loginformlong] "Content-Type" "text/htm
      return
    } elseif { [HTTP::path] starts_with "/Login_form" && [HTTP::method] equals "PO
      # Process the login form and auth the user
```

```
      HTTP::collect [HTTP::header Content-Length]
    }
  }
}
```

Once the form has been submitted, I extract the data in the HTTP_REQUEST_DATA event and process it.

```
when HTTP_REQUEST_DATA {
  set namevals [split [HTTP::payload] "&"]
  # Break out the POST data for username and password values
  for {set i 0} {$i < [llength $namevals]} {incr i} {
    set params [split [lindex $namevals $i] "="]
    if { [lindex $params 0] equals "username" } {
      set auth_user [lindex $params 1]
    }
    if { [lindex $params 0] equals "password" } {
      set auth_pw [lindex $params 1]
    }
    if { [lindex $params 0] equals "otp" } {
      set auth_otp [lindex $params 1]
    }
  }
  #validate basic authentication (username/password) before trying OTP validation
  binary scan [md5 $auth_pw] H* password
```

Now that we have a usable username and password, the verification is done in the iRule by comparing the user submission with the pre-loaded contents in the data-group. The password is stored as an md5 checksum.

```
[root@ltm1:Active:Standalone] config # echo -n "mypassword" | md5sum
34819d7beeabb9260a5c854bc85b3e44  -

ltm data-group internal authorized_users {
    records {
        jason {
            data 34819d7beeabb9260a5c854bc85b3e44
        }
        jrahm {
            data 34819d7beeabb9260a5c854bc85b3e44
        }
    }
    type string
}
```

This small snippet of the above iRule code handles the first factor of the authentication:

```
binary scan [md5 $auth_pw] H* password
if { $password eq [class lookup $auth_user authorized_users] } {
```

Steps 4 -6

Assuming the above user/password authentication was successful, the process advances to steps 4-6, shown in the code below. A second data-group (yubikey_users) is used for storing username / Yubikey serial numbers. The serial number from the data-group is extracted and converted to modhex and compared to the modhex value submitted as part of the token from the user.

```
ltm data-group internal yubikey_users {
    records {
        jrahm {
            data xxxxxx
        }
    }
    type string
}
when RULE_INIT {

  #for yubico auth
  array set static::modhex_alphabet { 0 c 1 b 2 d 3 e 4 f 5 g 6 h 7 i 8 j 9 k A l

}
set yubikey_modhex 1
if { $auth_user ne "" } {
  set yubikey_serial [string trimleft [class lookup $auth_user yubikey_users] 0]
  if { $yubikey_serial eq "" } {
    HTTP::respond 200 content "<p>No Yubikey on file for username $auth_user. Cont
    return
  }
```

```
    }
    if { [string is integer -strict $yubikey_serial] } {
      set yubikey_serial [split [format %012X $yubikey_serial] ""]
      set yubikey_modhex ""
      foreach index $yubikey_serial {
        append yubikey_modhex $static::modhex_alphabet($index)
      }
    }
  }
  if { $yubikey_modhex equals [string range $auth_otp 0 11] } {
```

Steps 7-9

Now that the user has been authenticated, and the serial number validated for the second authentication attempt, I can build the sideband connection request from the BIG-IP to the YubiCloud service. This can be done with or without signing the request, but they recommend signing if not using https, so that's what I've done in this example. When you establish a yubicloud account, you get an API client id and secret key, and I've stored those in static variables in RULE_INIT.

```
when RULE_INIT {
  set static::yubico_client_id "xxxxx"
  set static::yubico_secret_key "xxxxx"
}
# ... into the HTTP_REQUEST_DATA event
## Build GET request to yubico ##
set params "id=$static::yubico_client_id&nonce=$nonce&otp=$auth_otp"
set signature [string map { "+" "%2B" } [b64encode [CRYPTO::sign -alg hmac-sha1 -k
set yubico_get_request "GET /wsapi/2.0/verify?$params&h=$signature HTTP/1.1\r\n"
append yubico_get_request "Host: api2.yubico.com\r\n"
append yubico_get_request "Accept: */*\r\n\r\n"
## Create connection and send request
set conn [connect -timeout 1000 -idle 30 $yubico_server:80]
send -timeout 1000 -status send_status $conn $yubico_get_request
## Store Response from yubico
set yubico_response [recv -timeout 1000 -status recv_info $conn]
#set hash [getfield [getfield $yubico_response "\r\n" 9] "=" 2]
#set timestamp [getfield [getfield $yubico_response "\r\n" 10] "=" 2]
set otp_r [getfield [getfield $yubico_response "\r\n" 11] "=" 2]
set nonce_r [getfield [getfield $yubico_response "\r\n" 12] "=" 2]
#set sl [getfield [getfield $yubico_response "\r\n" 13] "=" 2]
set status [getfield [getfield $yubico_response "\r\n" 14] "=" 2]
if { not ( ($status eq "OK") && ($auth_otp eq $otp_r) && ($nonce eq $nonce_r)) } {
  HTTP::respond 200 content "<p>Yubico Authentication Failed. Status=$status</p>"
} else {
  session add uie $auth_user 0 1800
  set encrypted_user [b64encode [AES::encrypt $aeskey $auth_user]]
  set authcookie [format "%s=%s; path=/; " $ckname $encrypted_user]
  HTTP::respond 302 Location $orig_uri "Set-Cookie" $authcookie
}
```

You may be wondering where that nonce, the AES key, the YubiCloud server IP, and various other status variables are being set. Well, that occurs earlier in the client accepted event as show below.

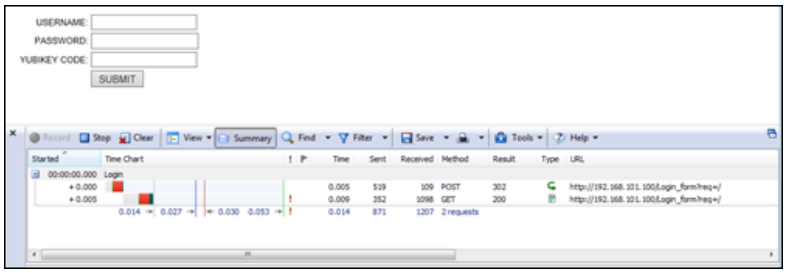
```
when CLIENT_ACCEPTED {
  set attempts 0
  #for form auth
  set forceauth 1
  set auth_status 2
  set aeskey "AES 128 63544a5e7178677b45366b41405f2dab"
  set ckname BIGIP_AUTH
  #for yubico auth
  set yubico_server [RESOLV::lookup @24.217.0.5 -a "api2.yubico.com"]
  set nonce ""
  set chars "0123456789"
  set range [expr {[string length $chars]-1}]
  for {set i 0} {$i < 25} {incr i} {
    set pos [expr {int(rand()*$range)}]
    append nonce [string range $chars $pos $pos]
  }
}
```

Testing

Now that the solution is in place, I can test first and second factor authentication.

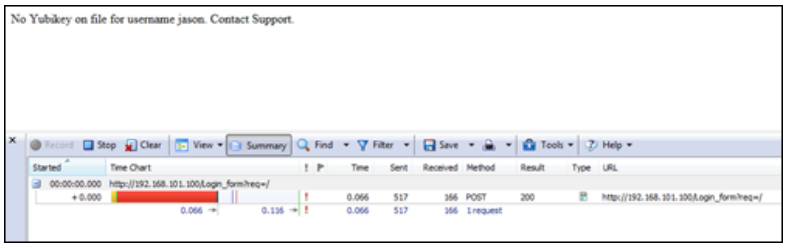
Unsuccessful First Authentication

For unsuccessful attempts on the first authentication, the user will be redirected to the login form, but the user will be cut off after four unsuccessful attempts.



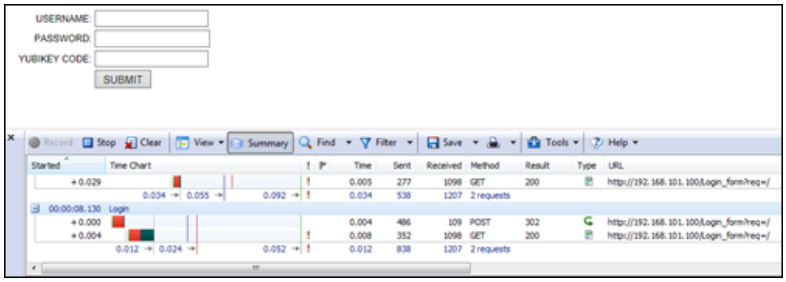
Successful First Authentication for User with No Yubikey

Here the user successfully authenticated, but the system has no record of user Jason having a Yubikey.



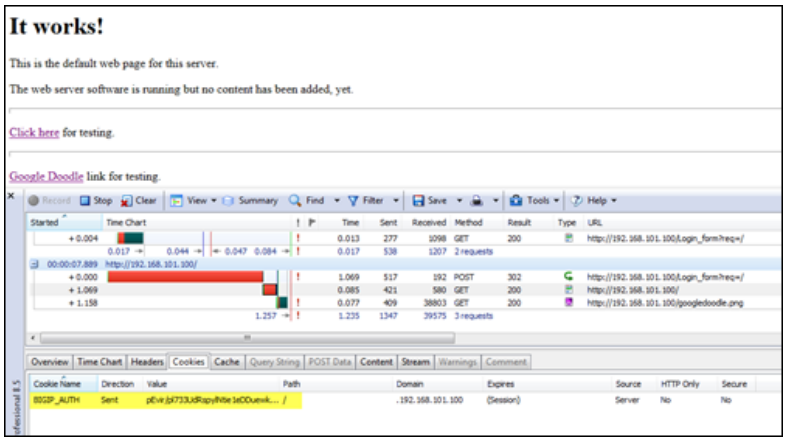
Successful First Auth, Yubikey Holder with Bad Token

This attempt is successful for the first factor, and the user has a configured Yubikey in the system, but the status from the YubiCloud service is not returned OK, so the user is redirected to the login form. Again, the user gets four attempts before being rejected.



Successful Two-Factor Authentication

In this attempt, all steps succeed and the user is directed to the application. Notice the highlighed auth cookie down in the httpwatch capture section.



Conclusions

There are many good solutions here on DevCentral. In the making of this solution, I frankensteined a few of them to build this solution.

- HTTP Basic Auth Tech Tip
- Yubikey BIG-IP APM
- Client Auth Using HTML Forms

The entirety of the this Yubikey solution can be found in this iRules wiki CodeShare entry.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com