Clientless FirePass Login via the command line using client certificate

Norbert Murzsa, 2009-08-11

Clientless FirePass Login via the command line using client certificate

This document is an extension to the article "Clientless FirePass Login via the command line" to present the configuration if the command line client uses client certificate for authentication. It contains some minor changes to the original script.

This configuration is perfect for an embedded UNIX (Linux/*BSD/other) OS based device for automatic authentication using client certificate.

The background and prerequisites of this article are available on the following links:

> Clientless FirePass Login via the command line
> Passwordless Authentication via FirePass using client certificate

Initial information to the configuration

About the extended script

The extended script is still a Perl script but is not restricted for using strict variables only (sorry).

The supported OpenSSL version

The required features by OpenSSL are supported only from version 0.9.8.

The client certificate

Client certificate is stored in a PEM (Privacy Enhanced Mail) format file as just its private key. The private key's password is stored in the extended script as clear text (that is not too secure at the moment so it's suggested to be stored in an encrypted device such as USB key or similar).

Additional HTTP header fields

The required additional HTTP header fields were detected via WebScarab – available on the OWASP web site.

- Header field "X-SSLClientCertificate" must be defined in all HTTP request
- Header field "User Agent" defined as an additional feature for FirePass logging

The "X-SSLClientCertificate" HTTP header field

The value of the HTTP header field "X-SSLClientCertificate" has been generated using two parts of the digital certificate:

- The MD5 digest of the certificate's fingerprint and
- The subject DN of the certificate

Both of values are in special, pre-formatted format within the HTTP header field.

The first part of this field's value is based on the certificate's MD5 digest fingerprint where only the

The first part of this field's value is based on the certificate's MD5 digest fingerprint where only the

```
# openssl x509 -noout -md5 -fingerprint -in cert.pem
MD5 Fingerprint=E3:D7:86:73:58:89:AD:42:FD:EE:53:0A:55:28:76:82
```

upper nibbles must be contained by the X-SSLClientCertificate in comma separated form.

```
# The value of the X-SSLClientCertificate is the upper nibbles of the comma separated hexadecimal values of the
certificate's MD5 digest fingerprint.

MD5 Fingerprint       =E3:D7:86:73:58:89:AD:42:FD:EE:53:0A:55:28:76:82
X-SSLClientCertificate =E :D :8 :7 :5 :8 :A :4 :F :E :5 :0 :5 :2 :7 :8

X-SSLClientCertificate =E:D:8:7:5:8:A:4:F:E:5:0:5:2:7:8
```

The second part of this field's value is based on the subject DN of the certificate

```
# openssl openssl x509 -noout -md5 -fingerprint -in cert.pem
subject= /C=NZ/L=Wellington/O=Zord/OU=EUC/CN=00345/emailAddress=admin@zord.co.nz
```

but in reverse version with some additional pre-formatting and separated with commas:

```
EMAILADDRESS=admin@zord.co.nz, CN=00345, OU=EUC, O=Zord, L=Wellington, C=NZ
```

So the correct value of the HTTP header field of the X-SSLClientCertificate is using the previous examples (this is only one line of course):

```
E:D:8:7:5:8:A:4:F:E:5:0:5:2:7:8 EMAILADDRESS=admin@zord.co.nz, CN=00345, OU=EUC, O=Zord, L=Wellington,
C=NZ
```

The purpose of the subroutine get_x_scc isto generate the value of the X-SSLClientCertificate HTTP header field from the client certificate.

Dedicated SSL VPN service on the FirePass gateway

The dedicated SSL VPN service for passwordless authentication uses port 8443 in this example.

Changes to the original script are highlighted below:

```
#!/usr/bin/perl -w
#use strict 'vars';

###
### STEP 0 :: Verify that we have the necessary requirements.
###
### This script requires recent versions of Perl, OpenSSL and PPPD (including
### the 'chat' program), and that they all be in our PATH.  This script was
### written and tested with the following versions:
```

```
### Written and tested with the following versions:
###    FirePass:     5.5 and 6.0
###    Perl:         5.8.8
###    OpenSSL:      0.9.8b
###    PPPD:         2.4.4

###
### STEP 1 :: Set up variables with the proper information to log in.
###

# There is no need for username so its value is empty
my $username="";
# Paths to client certificate related files (certificate, private key)
my $client_cert="/home/kuksi/ca2/cert.pem";
my $client_key="/home/kuksi/ca2/priv.pem";
my $client_pass="pass:VerySecurePaSSWORD";

# Required additional HTTP header field is “X-SSLClientCertificate”
my $x_sslclientcertificate = &get_x_scc();

# Default values for the FQDN or IP of the FirePass we wish to connect to, the
# name of our Network Access favorite, and our username/pasword.  All of these
# can be passed as arguments, if desired.
my $host = $ARGV[0] || 'fqdn.sslvpn.server.com';
my $name = $ARGV[1] || 'UNIXClient00345';
# There are no need for user and pass arguments here
#my $user = $ARGV[2] || '';
#my $pass = $ARGV[3] || '';

# Declare variables used throughout the rest of the script.
my($request, $response, $sessionid, $favorite);

# Store the OpenSSL command in a variable for convienence.
# The extended openssl command using the client certificate related options
my $openssl = "openssl s_client -ign_eof -cert ${client_cert} -key ${client_key} -pass ${client_pass} -connect
${host}:8443";
system("echo Openssl: \"$openssl\"");

###
### STEP 2 :: Log in to FirePass.
###

# This is the bare minimum required in order to successfully log in.  A normal
# browser will make many more requests than this to complete the log in
# sequence, but all that is required is this POST with our credentials.  This
# may fail if the FirePass has End-Point Security Policies configured.
#$request = "username=${user}&password=${pass}&mrhlogonform=1";

# The username's value is empty and
# password's value is really just six stars “******”.
$request = "username=${username}&password=******&rsa_port=&vhost=standard&state=&mrhlogonform=1";

$request = "POST /my.activation.php3 HTTP/1.1\r\n"
      . "Host: ${host}:8443\r\n"
      . "User Agent: Embedded UNIX Client (EUC)\r\n"
      . "Content-Type: application/x-www-form-urlencoded\r\n"
```

```
        . "Content-Length: " . length($request) . "\r\n"
        . "Connection: close\r\n"
        . "X-SSLClientCertificate: ${x_sslclientcertificate}\r\n"
        . "\r\n"
        . "${request}\r\n";

$response = qx(echo "${request}" | ${openssl}  2>&1);

# We can then parse the response for the MRHSession Cookie, which contains our
# SessionID.  In this example, we print out the SessionID in order to verify
# that our log in attempt worked.
$response =~ /MRHSession=(\w+);/;
$sessionid = $1;
print "SessionID: ${sessionid}\n";

###
### STEP 3 :: Create the SSL VPN tunnel.
###

# Now that we are authenticated and have a valid SessionID, we must request
# specific pages/objects in order to initiate a SSL VPN tunnel.  Before we do
# this, let's determine the resource locator for our Network Access favorite.

$request = "GET /vdesk/vpn/index.php3?outform=xml HTTP/1.0\r\n"
        . "Host: ${host}:8443\r\n"
        . "User Agent: Embedded UNIX Client (EUC)\r\n"
        . "Content-Type: application/x-www-form-urlencoded\r\n"
        . "Cookie: MRHSession=${sessionid}\r\n"
        . "X-SSLClientCertificate: ${x_sslclientcertificate}\r\n"
        . "\r\n";

$response = qx(echo "${request}" | ${openssl} 2>&1);

# The response is XML, so we can safely grab what we are looking for using some
# regular expression magic.  Same with the SessionID, we're printing out the
# final value to make sure we're on the right track.
$response =~ /${name}[^\n]+\n[^Z]+Z=\d+,(\d+)/;
$favorite = $1;
print "Favorite: ${favorite}\n";

# We're all set!  Let's visit the necessary pages/objects to notify FirePass
# that we wish to open a SSL VPN tunnel.
foreach my $uri (
        "/vdesk/",
        "/vdesk/vpn/connect.php3?Z=0,${favorite}",
) {
        $request = "GET ${uri} HTTP/1.0\r\n"
            . "Host: ${host}:8443\r\n"
            . "User Agent: Embedded UNIX Client (EUC)\r\n"
            . "Content-Type: application/x-www-form-urlencoded\r\n"
            . "Cookie: MRHSession=${sessionid}\r\n"
            . "X-SSLClientCertificate: ${x_sslclientcertificate}\r\n"
            . "\r\n";
        system("echo \"${request}\" | ${openssl} 2>&1");
}
```

```
# We are now authenticated, and have requested the necessary pre-tunnel
# pages/objects.  It's time to start our SSL VPN connection.  To do this, we are
# simply calling PPPD, and having it use OpenSSL as a psuedo terminal device.
$request = "GET /myvpn?sess=${sessionid} HTTP/1.0\r\n"
          . "Host: ${host}:8443\r\n"
          . "User Agent: Embedded UNIX Client (EUC)\r\n"
          . "Content-Type: application/x-www-form-urlencoded\r\n"
          . "Cookie: MRHSession=${sessionid}\r\n"
          . "X-SSLClientCertificate: ${x_sslclientcertificate}\r\n"
          . "\r\n";
$request = "chat -v '' '${request}'";
system("pppd noauth pty \"${openssl}\" connect \"${request}\"");

# Voila!  We should now have a PPP connection running over SSL.  We can exit
# from this script cleanly, and move on to setting up routes to the remote
# network using our favorite networking tools.  Happy hacking!
exit(0);

###
### End of file.
###



sub get_x_scc
{
    my $i=0;
    my $subfields=0;
    my $variable = "";
    my $value = "";
    my $zz = "";
    my $fgp  ="openssl x509 -noout -md5 -fingerprint -in ${client_cert}";
    my $subj = "openssl x509 -noout -subject -in ${client_cert}";
    $subj = qx( ${subj} );
    $subj = substr($subj, 0, length($subj)-1);
    $subj =~ s/^subject\=\ \//g;
    # print("$subj\n");

    my @subfields = split(/\//, $subj);

    $subj = "";
    my $elemszam = @subfields;
    # print("Numbers of fields: $elemszam\n");
    for($i=$elemszam-1;$i>=0;$i--)
    {
       ($variable, $value) = split(/=/, $subfields[$i]);
       $variable =~ tr/a-z/A-Z/;
       if($variable eq "CN")
       {
           $username=$value;
       }
       $subfields[$i] = $variable."=".$value;
       $subj=$subj.$subfields[$i].", ";
       #print("$i. @subfields[$i]\n");
    }
```

```
    $subj = substr($subj, 0, length($subj)-2);
    # print("-$subj-\n");

    my $fingerprint = qx( ${fgp} );
    $zz="";

    # print("$fingerprint\n");
    $fingerprint = substr($fingerprint, 16, length($fingerprint)-15);
    # print("$fingerprint\n");
    $fingerprint=~ s/://g;
    # print("SG: $fingerprint\n");

    $fgp="";

    for($i=1;$i<length($fingerprint);$i++)
    {
      if($i%2>0)
      {
        $zz = substr($fingerprint, $i-1, 1);
        $fgp=$fgp.$zz.":";
      }

    }
    $fgp = substr($fgp, 0, length($fgp)-1);
    #$fgp = $fgp." ".
    # print("En1: $fgp\n");
    # print("End: E:D:8:7:5:8:A:4:F:E:5:0:5:2:7:8\n");

    return($fgp." ".$subj);
}
```

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