Authen::SASL - SASL in Perl for client/server authentications

Patrick Boettcher
Technische Fachhochschule Wildau
patrick.boettcher@desy.de

March 2, 2004

SASL overview

- What is SASL? (theory
- How SASL works
- SASL in Perl

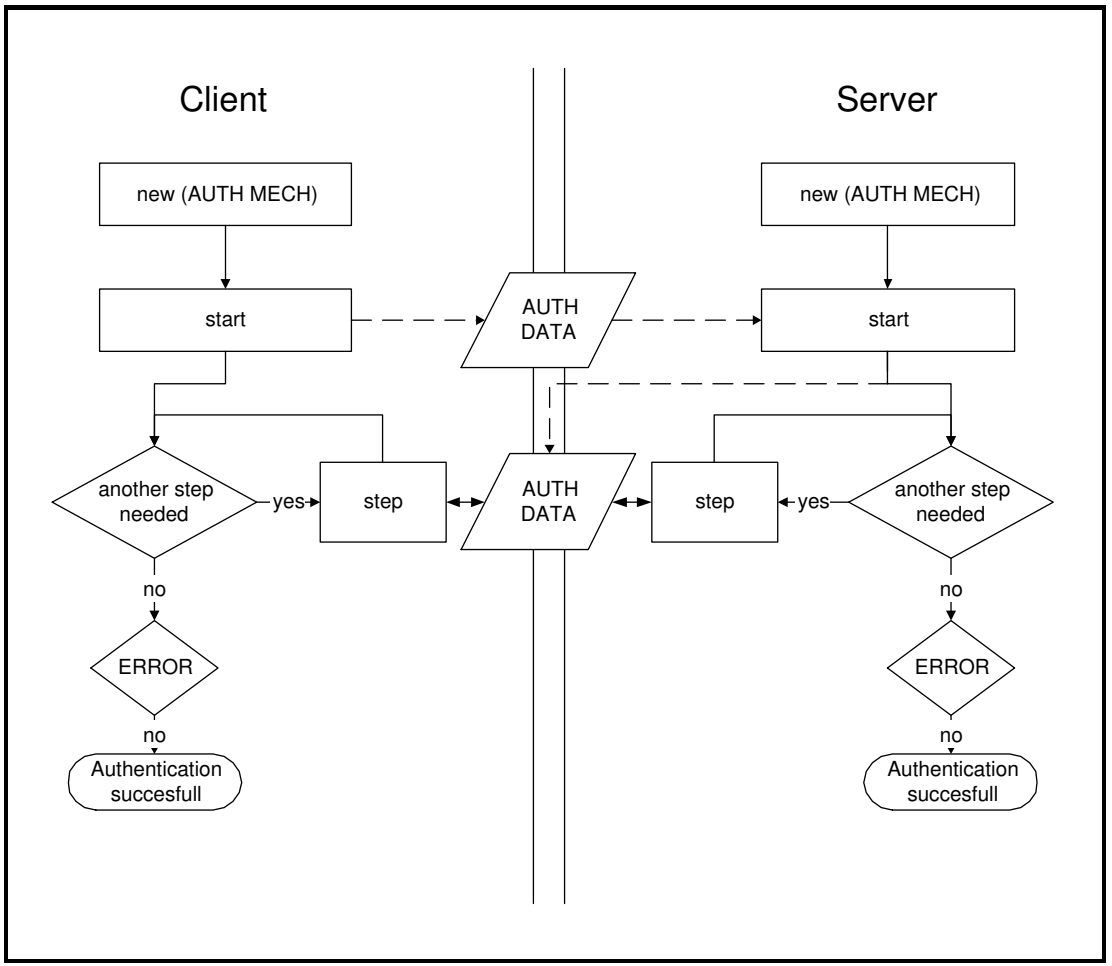
What is SASL?

- SASL stands for ”Simple Authentication and Security Layer”
- abstracts a wide range of authentication mechanisms
- even supports challenge-response-based mechanisms like GSSAPI (Kerberos 5) or MD5 algorithms
- provides some generic functions on client- and server-side

How does SASL work (1)

- there is an *init*, a *new*, a *start* and a *step* method
- the wanted mechanism must be given to the SASL-library at the *new*-call
- *start* and *step* methods are used to handle the authentication
- SASL gets user specific data via different callback functions

How does SASL work (2)



SASL implementations

- several implementations of SASL libraries exist
  - Cyrus SASL (C library, most popular)
  - GNU SASL (C library, not very matured)
  - Authen::SASL::Perl (Perl, client only, less mech)
  - Authen::SASL::Cyrus (Perl, same features like Cyrus SASL)

Authen::SASL::Cyrus

- at first, no server functionality was available
- in ARCv2 server side SASL is necessary, so I had to extend Perl's Authen::SASL framework (i.e. Authen::SASL::Cyrus) using XS language
- patches are on their way to CPAN; hopefully with the end of March it is available to non-desy users

SASL in Perl for client authentication

- `use Authen::SASL;` includes the Perl SASL framework
- `client_new` method creates an Authen::SASL::Cyrus object
- example `sasl-client.pl`

SASL in Perl for server authentication

- `use Authen::SASL;` includes the Perl SASL framework
- `server_new` method creates an Authen::SASL::Cyrus object
- example `sasl-server.pl`

sasl-client.pl:

```
use strict;
use Authen::SASL;

my $sasl = Authen::SASL->new (
mechanism => "PLAIN",
callback => {
user => \&getusername,
pass => \&getpassword,
}
);

# Creating the Authen::SASL::Cyrus instance
my $conn = $sasl->client_new("service", "hostname.domain.tld");

# Client begins always
sendreply($conn->client_start());

while ($conn->need_step) {
sendreply($conn->client_step( &getreply() ) );
}

if ($conn->code == 0) {
print STDERR "Negotiation succeeded.\n";
```

```
} else {
print STDERR "Negotiation failed.\n";
}

sub getusername
{
print "Username: ";
return <>;
}

sub getpassword
{
print "Password: ";
return <>;
}
```

sasl-server.pl:

```
use strict;
use Authen::SASL;

my $sasl = Authen::SASL->new (
mechanism => "PLAIN",
callback => {
checkpass => \&checkpass,
```

```
canonuser => \&canonuser,
}
);

# Creating the Authen::SASL::Cyrus instance
my $conn = $sasl->server_new("service");

# Clients first string (maybe "", depends on mechanism)
# Client has to start always
sendreply( $conn->server_start( &getreply() ) );

while ($conn->need_step) {
sendreply( $conn->server_step( &getreply() ) );
}

if ($conn->code == 0) {
print "Negotiation succeeded.\n";
} else {
print "Negotiation failed.\n";
}

sub checkpass
{
return 1;
}
```

```
sub canonuser
{
my ($username);
return $username;
}
```

The new ARC - A SASL based client/server application

Patrick Boettcher
Technische Fachhochschule Wildau
patrick.boettcher@desy.de

March 2, 2004

ARCv2 overview

- ARC - tasks, history, rewriting
- ARCv2 internals
- How to use it

ARC principle

- ARC is client/server application
- user command requests are shipped from a client to a server
- server runs the mostly privileged commands in an appropriate environment (privileged)
- server decides if the calling user is allowed to run the command
- one can say, ARC is a secure RPC

The history of ARC

- Originally written by Rainer Toebbicke in 1995/1996
- primarily to support administrators to manage the AFS File-space and for acron
- Alf Wachsmann extends ARC to enable Kerberos V authentication

Weaknesses of ARC

- only Kerberos 4 authentication is possible
- it is triggered through the super-server ”inetd”
- originally released under a CERN license, was not freely downloadable
- no comprehensive documentation
- sometimes no (complete) response is returned by the server

Expected features of ARCv2 (1)

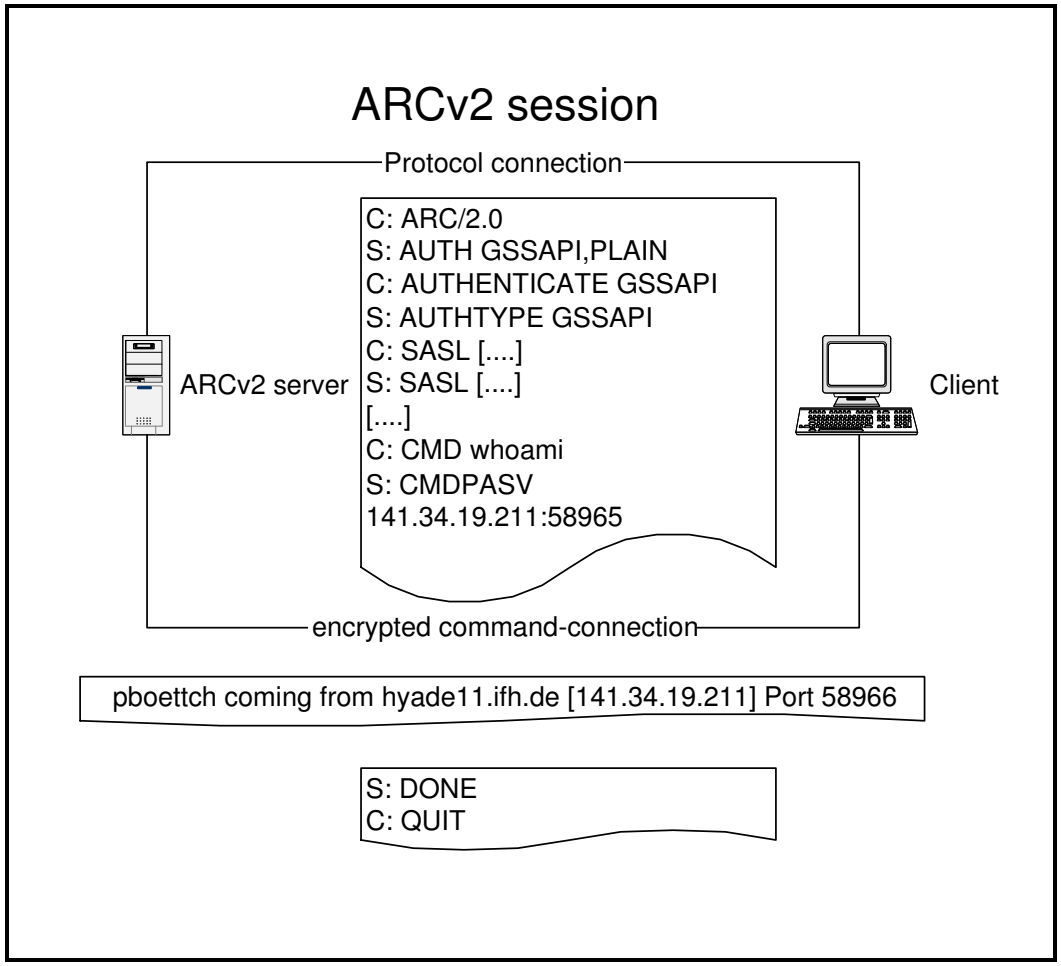
- portability across a wide range of platforms (achieved by using Perl)
- a very robust server has to be available (serve a high number of requests, stand-alone server)
- Authentication has to support at least Kerberos 4 and 5 (SASL)
- ARCv2 command writing has to be easy

Expected features of ARCv2 (2)

- a comprehensive documentation has to be available
- it should be free software, accessible by everybody (CPAN)

ARCv2 internals

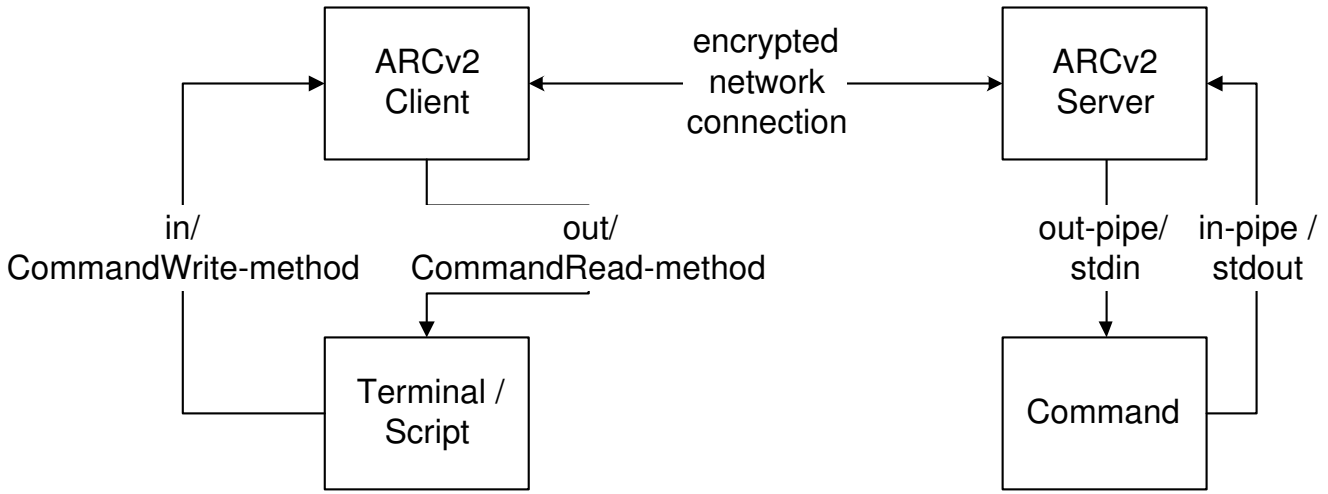
- two connection types are existing
  - protocol connection
  - command connection
- run several ARCv2 commands within one protocol connection; improves speed for more complex commands
- for each command a new connection will be opened (like FTP)


ARCv2 session
Protocol connection
C: ARC/2.0
S: AUTH GSSAPI,PLAIN
C: AUTHENTICATE GSSAPI
S: AUTHTYPE GSSAPI
C: SASL [....]
S: SASL [....]
[....]
C: CMD whoami
S: CMDPASV
141.34.19.211:58965
ARCv2 server
Client
encrypted command-connection
pboettch coming from hyade11.ifh.de [141.34.19.211] Port 58966
S: DONE
C: QUIT

Command environment

- the server runs the requested command in a separate child process
- the STDIN and STDOUT of this process are duplicated to pipes
- the server process reads and writes the data from/to the command connection to these pipes and vice versa

Command data transfer



Usage of the client - arcx

- command line interface: `arcx` (ARC eXtended)
- parameters are like the old `arc` + some extensions
- Examples: (see arcx(3))

```
arcx whoami
arcx -h hyade11
cat /etc/passwd | arcx put /etc/passwd
arcx get /etc/passwd > /etc/passwd
```

Usage of the server - arcxd

- `arcxd` is a command line script to start the stand-alone server
- must be configured by a configuration file
- can fork into the background
- see arcxd(3)

ARCv2 in Perl - client

- `use Arc::Connection::Client;`

  includes the approp. class to use client functionality in a Perl script

- `my $arc = new Arc:: .. ( ... );`

  creates a Perl object

- several methods are available to run commands

- example `arc-client.pl`

ARCv2 in Perl - server (1)

- `use Arc::Server;` includes the approp. class to run the server in a Perl script
- `my $arc = new Arc::Server(..);` creates a Perl object
- the parameters given at the new-call are influencing the server a lot
- before starting the server it is important to configure the environment (e.g. creating/extracting keys for Kerberos V)

ARCv2 in Perl - server (2)

- the method `Start` starts the server, including:
  - listening on the given port
  - accepting connections
  - forking into background
- the method `Interrupt` ends the server
- example `arc-server.pl`

Writing an ARCv2 command (1)

- self-made ARCv2 commands should be derive from Arc::Command
- have to be a Perl class (have a new function, bless)
- have to implement a `Execute` method (entry function for the server)
- several variables are available (e.g. user name (SASL), client, authentication mechanism)

Writing an ARCv2 command (2)

- a command can get client data by reading from STDIN
- and can send data to client by writing to STDOUT
- example `whoami.pm`

Register a command to a server

- via configuration file when using arcxd or at the *new* call as a parameter
- Perl *class name* can be assigned to one or more *command names*
- the called command name is given as a member variable to the actual command class

arc-client.pl:

```
#!/usr/bin/perl -w

use Arc::Connection::Client;
use strict;

my $arc = new Arc::Connection::Client(
   server => "hyade11",
   port => 4242,
   timeout => 30,
   loglevel=> 7,
   logdestination => 'stderr',
   service => 'arc',
   sasl_mechanism =>undef,
   sasl_cb_user => $ENV{'USER'},
   sasl_cb_auth => $ENV{'USER'},
   sasl_cb_pass => \&password,
);

if (my $m = $arc->IsError()) {
   die $m;
}

if ($arc->StartSession) {
```

```
    $arc->CommandStart("test");
    $arc->CommandWrite("hello\n");
    if (my $t = $arc->CommandRead()) {
       print $t; # should give 'ell'
    }
    $arc->CommandEnd();
}

sub password
{
    return "pw";
}
```

arc-server.pl:

```
#!/usr/bin/perl -w
use Arc::Server;
use strict;

$SIG{INT} = \&int;
my $arc = new Arc::Server(
     listenport => 4243,
     loglevel => 7,
     logdestination => "stderr",
```

```
    maxpidcount => 5,
    daemonize => 0,
    connection_vars => {
        loglevel => 7,
        logdestination => 'stderr',
        timeout => 30,
        sasl_mechanisms => ["GSSAPI","KERBEROS_V4","PLAIN"],
        sasl_cb_getsecret => &getsecret,
        sasl_cb_checkpass => &checkpass,
        commands => {
           'whoami' => 'Arc::Command::Whoami',
           'uptime' => 'Arc::Command::Uptime',
        },
        service => "service",
    }
);

if (my $m = $arc->IsError()) {
    die $m;
}

$arc->Start();

sub int
{
```

```
    $arc->Interrupt() if defined $arc;
}

sub getsecret
{
return "pw";
}

sub checkpass
{
return 1;
}
```