GNU Network Security Labyrinth

- or: an howto for network application authors

TLS
SASL
Kerberos
GSS-API

About me

Free software hacker

Independent consultant

http://josefsson.org/

Swedish

Nordic Free Software Award 2009

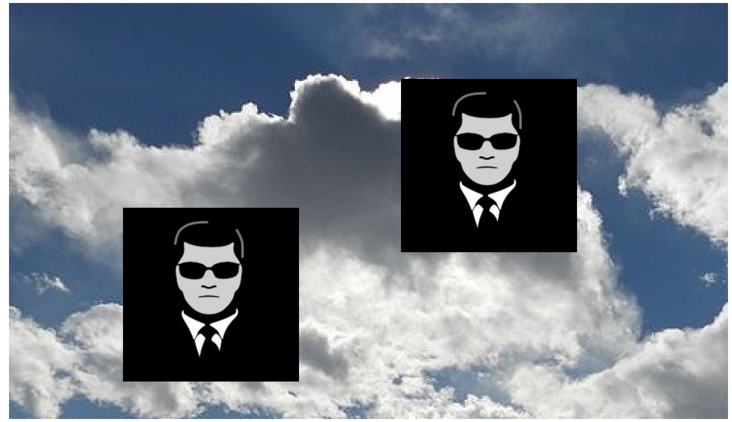
Member of fossgruppen.se

I'll talk about technologies
and their implementations

Technologies – Implementations

Kerberos – GNU Shishi
GSS-API – GNU GSS
SASL – GNU SASL
SSL/TLS – GnuTLS

What is all this about?



Alice & Mallory & Bob

Alice wants to talk to Bob

In private → encrypted

They want to know who they are talking to → authenticated

We will write the tool that
Alice and Bob is using

It is a client and server

(could be peer-to-peer, but not today)

Client inputs: ADDR

1. Lookup ADDR in DNS
2. Open socket to destination adress
3. Exchange message

Server inputs: None

1. Listen on socket
2. Exchange messages

The tool is flawed

Alice doesn't know she is talking to Bob
Bob doesn't know he is talking to Alice
Mallory can listen to the conversation
Mallory can modify the conversation
Mallory can pretend to be Alice or Bob

Let's add TLS

TLS is the Transport Layer Security

TLS is the standardized
and improved variant of SSL

Client inputs: ADDR

1. Lookup ADDR in DNS
2. Open socket to destination adress
3. Perform TLS handshake
4. Exchange message

Server inputs: None

1. Listen on socket
2. Perform TLS handshake
3. Exchange messages

```
int socket;
gnutls_session_t session;
gnutls_anon_client_credentials_t anoncred;

gnutls_global_init ();
gnutls_anon_allocate_client_credentials (&anoncred);
gnutls_init (&session, GNUTLS_CLIENT);
gnutls_priority_set_direct (session, "PERFORMANCE:+ANON-DH",
                            NULL);
gnutls_credentials_set (session, GNUTLS_CRD_ANON, anoncred);
socket = tcp_connect ();
gnutls_transport_set_ptr (session, (gnutls_transport_ptr_t) socket);
ret = gnutls_handshake (session);
gnutls_record_send (session, MSG, strlen (MSG));
```

The tool is still flawed

Alice doesn't know she is talking to Bob
Bob doesn't know he is talking to Alice
Mallory can listen to the conversation (as MITM)
Mallory can modify the conversation (as MITM)
Mallory can pretend to be Alice or Bob

TLS can do many things

It supports different Key Exchange methods

Anonymous Diffie Hellman – DH_anon
only protects against passive attacks

TLS supports keyed Diffie-Hellman

Pre-shared symmetric key (PSK) or a verified public-key (RSA, DSA, ECDSA)

Let's skip PSK today

The server has a public-key
signed by a CA that the client trusts to verify
mapping between public-key and name

A signed public-key is stored in the form of a Certificate – X.509 or OpenPGP

Clients may also have a public key signed by a CA that the server trusts

Let's skip this today

Client inputs: ADDR, CA
1. Lookup ADDR in DNS
2. Open socket to destination adress
3. Perform TLS handshake
4. Verify server certificate against CA and ADDR
5. Exchange message

Server inputs: Certificate
1. Listen on socket
2. Perform TLS handshake with Certificate
3. Exchange messages

```
int sd;
gnutls_session_t session;
gnutls_certificate_credentials_t xcred;

gnutls_global_init ();
gnutls_certificate_allocate_credentials (&xcred);
gnutls_certificate_set_x509_trust_file (xcred, CAFILE,
                                         GNUTLS_X509_FMT_PEM);
gnutls_init (&session, GNUTLS_CLIENT);
gnutls_priority_set_direct (session, "NORMAL", NULL);
gnutls_credentials_set (session, GNUTLS_CRD_CERTIFICATE, xcred);
sd = tcp_connect ();
gnutls_transport_set_ptr (session, (gnutls_transport_ptr_t) sd);
gnutls_handshake (session);
gnutls_certificate_verify_peers2 (session, NULL);
gnutls_record_send (session, MSG, strlen (MSG));
```

Now we are getting somewhere

Alice knows she is talking to Bob
Bob doesn't know he is talking to Alice
Mallory cannot listen to the conversation
Mallory cannot modify the conversation
Mallory can pretend to be Alice

Alice needs to trust the CA used by Bob

Similar security as provided on the web

Let's add SASL

SASL is the Simple Authentication and Security Layer

SASL specified in RFC 4422

GNU SASL supports CRAM-MD5 EXTERNAL GSSAPI ANONYMOUS PLAIN SECURID DIGEST-MD5 SCRAM-SHA-1 SCRAM-SHA-1-PLUS GS2-KRB5 LOGIN NTLM KERBEROS_V5

Most common mechanism is CRAM-MD5

CRAM-MD5 takes a username and a password

```
. AUTHENTICATE CRAM-MD5
+ PDUzMzMxMTg1MjUwMjM0OTQxMjM0LjBAbG9jYWxob3N0Pg==
YWxpY2UgM2MwOTI5ZjdkY2JjOTkyMDcyZWRhYzZjZTM3YWQ2ZjE=
. OK AUTHENTICATE CRAM-MD5 authentication success
```

Client inputs: ADDR, CA, USER, PASSWD

1. Lookup ADDR in DNS
2. Open socket to destination adress
3. Perform TLS handshake
4. Verify server certificate against CA and ADDR
5. Perform CRAM-MD5 with USER/PASSWD
6. Exchange message

Server inputs: Certificate, USER, PASSWD

1. Listen on socket
2. Perform TLS handshake with Certificate
3. Perform CRAM-MD5 with USER/PASSWD
4. Exchange messages

```
Gsasl *ctx = NULL;
Gsasl_session *session;
int rc;

gsasl_init (&ctx);
gsasl_client_start (ctx, "CRAM-MD5", &session);
gsasl_property_set (session, GSASL_AUTHID, "jas");
gsasl_property_set (session, GSASL_PASSWORD, "secret");

do
  {
    char buf[BUFSIZ] = "";
    char *p;
    rc = gsasl_step64 (session, buf, &p);
    send (p);
    recv (buf);
  }
while (rc == GSASL_NEEDS_MORE);

gsasl_finish (session);
gsasl_done (ctx);
```

Alice knows she is talking to Bob
Bob knows he is talking to Alice
Mallory cannot listen to the conversation
Mallory cannot modify the conversation
Mallory cannot pretend to be Alice

Alice needs to trust the CA that Bob used
Bob needs to know Alice's password
Alice needs a password for every Bob

Let's use SCRAM-SHA-1-PLUS

(but call it SCRAM+)

SCRAM+ clients hash username, password and a unique name (CB) of the TLS session

SCRAM+ servers can verify the hash using a hashed form of the password

Client inputs: ADDR, ~~CA~~, USER, PASSWD

1. Lookup ADDR in DNS
2. Open socket to destination adress
3. Perform TLS handshake
4. ~~Verify server certificate against CA and ADDR~~
5. Extract CB from TLS session
6. Perform SCRAM+ with USER/PASSWD/CB
7. Exchange message

Server inputs: (Certificate), USER, PASSWD

1. Listen on socket
2. Perform TLS handshake (with Certificate)
3. Extract CB from TLS session
4. Perform SCRAM+ with USER/PASSWD/CB
5. Exchange messages

Alice knows she is talking to Bob
Bob knows he is talking to Alice
Mallory cannot listen to the conversation
Mallory cannot modify the conversation
Mallory cannot pretend to be Alice
Alice doesn't need to trust the CA used by Bob
Bob doesn't need to know Alice's password

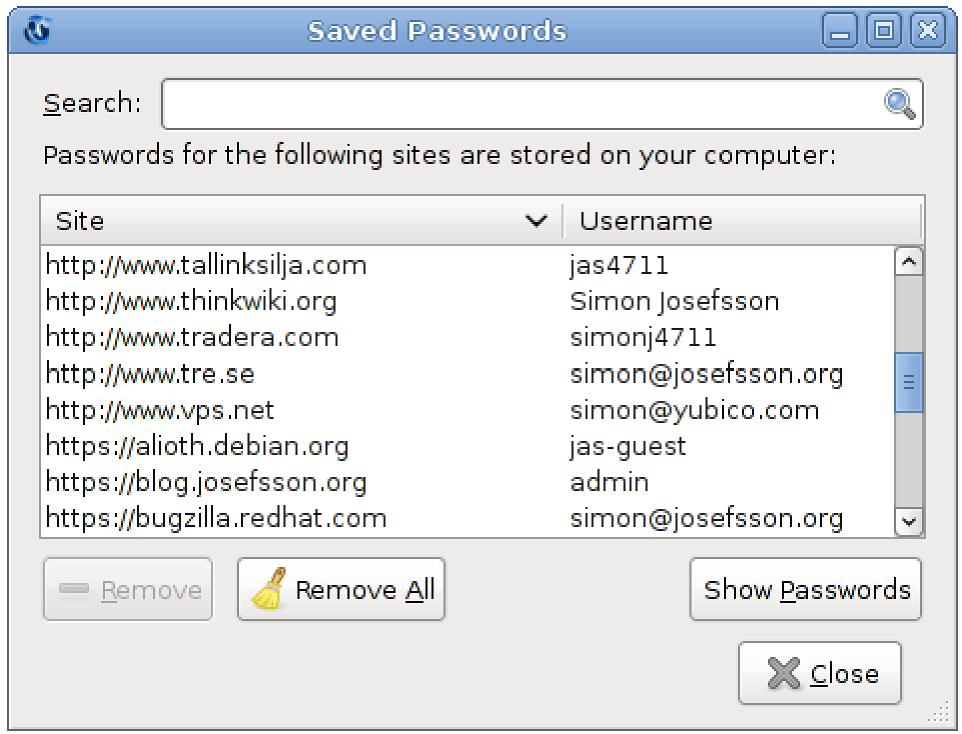
Alice needs a password for every Bob

One password per service does not scale

Password reuse between services

Phishing

Don't forget to synchronize passwords between all your devices


Saved Passwords
Search:
Passwords for the following sites are stored on your computer:
Site
Username
http://www.tallinksilja.com jas4711
http://www.thinkwiki.org Simon Josefsson
http://www.tradera.com simonj4711
http://www.tre.se simon@josefsson.org
http://www.vps.net simon@yubico.com
https://alioth.debian.org jas-guest
https://blog.josefsson.org admin
https://bugzilla.redhat.com simon@josefsson.org
Remove
Remove All
Show Passwords
Close

Let's add Kerberos

Kerberos introduces a trusted third party

Works well if Alice's and Bob's trust the same third party

There are many Alice & Bob's at universities and large enterprises

Alice acquires a ticket-granting-ticket (TGT) using a username (principal) and password

The ticket-granting-ticket is used to acquire one ticket per service

GNU Shishi implements Kerberos V5

GNU GSS implements the GSS-API for ”simpler” Kerberos programming

```
OM_uint32
gss_init_sec_context (
    OM_uint32                          *minor_status,
    const gss_cred_id_t                 initiator_cred_handle,
    gss_ctx_id_t                       *context_handle,
    const gss_name_t                    target_name,
    const gss_OID                       mech_type,
    OM_uint32                           req_flags,
    OM_uint32                           time_req,
    const gss_channel_bindings_t        input_chan_bindings,
    const gss_buffer_t                  input_token
    gss_OID                            *actual_mech_type,
    gss_buffer_t                        output_token,
    OM_uint32                          *ret_flags,
    OM_uint32                          *time_rec);
```

Preserve your sanity: use Kerberos/GSS-API through your friendly SASL library

SASL mechanism for Kerberos is called GS2-KRB5

GS2 specified in RFC 5801

(the author sounds familiar)

Client inputs: ADDR, KDC, USER, PASSWD

1. Get TGT with USER/PASSWD from KDC
2. Get service ticket for ADDR using TGT
3. Lookup ADDR in DNS
4. Open socket to destination adress
5. Perform TLS handshake
6. Extract CB from TLS session
7. Perform GS2KRB5+ with TGT/CB
8. Exchange message

Server inputs: (Certificate), SRVTAB

1. Listen on socket
2. Perform TLS handshake (with Certificate)
3. Extract CB from TLS session
4. Perform GS2KRB5+ with SRVTAB/CB
5. Exchange messages

Alice knows she is talking to Bob
Bob knows he is talking to Alice
Mallory cannot listen to the conversation
Mallory cannot modify the conversation
Mallory cannot pretend to be Alice
Alice doesn't need to trust the CA used by Bob
Bob doesn't need to know Alice's password
Alice doesn't need a password for every Bob

Alice and Bob needs to trust the same third party

We don't go further than this today

(to go beyond this you want to learn about federated authentication)

This is the end my friend

Questions?