CURRENT RPC SECURITY FLAVORS

• **AUTH_UNIX**
  - simple, fast, de-facto standard
  - trivial to defeat

• **AUTH_DES**
  - uses Diffie/Hellman public key algorithm
  - documented weaknesses (LaMacchia and Odlyzko, 1990)

• **AUTH_KERB**
  - uses Kerberos V4
  - missed market window to complete a product
  - documented weaknesses (Bellovin and Merritt, 1991)
EXERCISING HINDSIGHT ON PAST APPROACHES

• Flavors had designed in (unforeseen) limitations
  - AUTH_UNIX had too few Unix group ids
  - AUTH_DES had too small a key size
  - neither AUTH_DES nor AUTH_KERB provided integrity/privacy

• Adding a security mechanism required per application changes
  - no notion of flavor or security mechanism independence

• Security mechanism specifics had to be ported into kernel for NFS implementation.
INTEGRITY ATTACK

Intervening Network

Client Machine

Auth Header

RPC Call Body

Server Machine

Auth Header

Bogus RPC Call Body

AUTH_DES and AUTH_KERB

Mike Eisler
REQUIREMENTS FOR A SOLUTION

• Support multiple underlying security mechanisms
• Support all of Authentication, Integrity, Privacy
• Mechanism should be transparent to the application
• ISVs should be able to add new mechanisms
  - Modulo U.S.A export control laws
• Preservation of Binary and Source compatibility
• Use standards where possible

GSS-API has the above characteristics
GSS-API OVERVIEW

• RFC 1508 describes the framework
• RFC 1509 describes the C language bindings
• Similar to TLI
  - normalizes access to security mechanisms
  - like TLI, punts on generic naming issues

Application

GSS-API

Kerberos V5  SPKM  Others...
GSS-API OVERVIEW

- Binds authentication with mechanism
- Offers optional integrity or privacy
- Permits specification of Quality of Protection
  - cryptographic algorithm used with integrity or privacy
- Transport is the responsibility of application
  - However, some support for channel bindings.
  - GSS-API primitives return tokens which are sent to application’s peer
  - However, some support for channel bindings.
RPCSEC_GSS SECURITY FLAVOR

- A new flavor that encapsulates GSS-API:

  ONC RPC Application

  RPCSEC_GSS

  GSS-API

  Kerberos V5  SPKM  Others...

- Provides virtually all of the GSS-API interfaces to ONC application.
  - punt on channel bindings
API OF RPCSEC_GSS

Client side example:

AUTH *rpc_gss_seccreate(
    CLIENT *clnt,    /* in */
    char *principal, /* in */
    char *mechanism, /* in */
    rpc_gss_service_t service_type, /* in */
    char *qop,       /* in */
    rpc_gss_options_req_t
        *options_req,/* in */
    rpc_gss_options_ret_t
        *options_ret);/* out */

clnt -> cl_auth = rpc_gss_seccreate(clnt,
    "nfs@jurassic.eng.sun.com","kerberos_v5",
    rpc_gss_svc_integrity,
    "GSS_KRB5_INTEG_C_QOP_DES_MD5", NULL,
    NULL);
API OF RPCSEC_GSS

Server side example:

```
server_prog(struct svc_req *rqstp, SVCXprt *xprt)
{
    rpc_gss_ucred_t  *ucred;
    rpc_gss_rawcred_t  *rcred;

    switch (rqstp->rq_cred.oa_flavor) {
    case RPCSEC_GSS:
        /* get credential information */
        rpc_gss_getcred(rqstp, &rcred,&ucred,NULL);
        if (!authenticate_user(ucred->uid, rcred->mechanism,
                               rcred->qop, rcred->service)) {
            svcerr_weakauth(xprt);
            return;
        }
        break; /* allow the user in */
    default:
        svcerr_weakauth(xprt);
        return;
    } /* end switch */
    ...
}
```
RPCSEC_GSS PROTOCOL

• Session-based like AUTH_DES and AUTH_KERB

• Based on OpenVision’s AUTH_GSSAPI protocol

• Session has three phases:
  - Context creation
  - RPC Data Exchange
  - Context Destruction
RPCSEC_GSS PROTOCOL

Context creation request

• Procedure number in call header set to NULLPROC

• AUTH header’s credential:

```c
struct opaque_auth { /* credential */
    sec_flavor flavor; /* Set to RPCSEC_GSS */
    opaque body<400>; /* body encoded as rpc_gss_cred_t */
};

struct rpc_gss_cred_t {
    unsigned int version; /* set to 1 */
    unsigned int gss_proc; /* RPCSEC_GSS_INIT */
    unsigned int seq_num; /* ignored */
    enum service; /* ignored */
    opaque handle<>; /* zero length */
};
```

• AUTH header’s verifier is NULL on context create.
RPCSEC_GSS PROTOCOL

Context creation request

• Call arguments don’t contain NULLPROC args, but instead:

```c
struct rpc_gss_init_arg {
    opaque gss_token<>; /* from GSS-API’s gss_init_sec_context() */
    unsigned int qop;
    enum service; /*integrity, privacy, default, or none */
};
```
RPCSEC_GSS PROTOCOL

Context creation response

• Response results don’t contain NULLPROC results, but instead:

```c
struct rpc_gss_init_res {
    opaque handle<>; /* context identifier */
    /* gss_major/gss_minor returned from GSS-API’s
       gss_accept_sec_context() interface */
    unsigned int gss_major;
    unsigned int gss_minor;
    unsigned int seq_window; /* maximum number of
        outstanding RPC requests for this context. */
    opaque gss_token<>; /* token from
        gss_accept_sec_context() */
};
```
RPCSEC_GSS PROTOCOL

RPC Call

- AUTH header format:

```c
struct opaque_auth { /* credential */
    sec_flavor flavor; /* Set to RPCSEC_GSS */
    opaque body<400>; /* encoded as rpc_gss_cred_t */
};

struct rpc_gss_cred_t {
    unsigned int version; /* set to 1 */
    unsigned int gss_proc; /* RPCSEC_GSS_NULL */
    unsigned int seq_num; /* monotonically increasing */
    enum service; /* integrity, privacy, none */
    opaque handle<>; /* context id from context create response */
};
```
RPCSEC_GSS PROTOCOL

RPC Call

- AUTH header format:

  ```c
  struct opaque_auth { /* verifier */
    sec_flavor flavor; /* Set to RPCSEC_GSS */
    opaque body<400>;
  }
  ```

  - opaque_auth.body is set to gss_sign() (check sum) of rest of RPC call header:

    ```plaintext
    dir  xid  rpc ver  prog #  prog vers  prog proc  cred  verifier
    signed portion of call header
    ```

    ```plaintext
    gss_sign
    ```
RPCSEC_GSS PROTOCOL

RPC Call

- Integrity protected requests

request header
  credential
  sequence #
  otw request header

sequence #

XDR encoded args
  gss_sign()

otw request data
RPCSEC_GSS PROTOCOL

RPC Call

• Privacy protected requests

request header
  credential
    sequence #

sequence #

XDR encoded args

gss_seal()

otw request header

otw request data
RPCSEC_GSS PROTOCOL

Server processing of requests

- Server verifies
  - version number of RPCSEC_GSS from cred
  - service specified in cred
  - context handle in cred
  - sequence number in cred
  - header checksum (gss_sign()) in verifier
RPCSEC_GSS PROTOCOL

Server processing of requests

• Sequence number processing
  - Server maintains WINDOW of sequence numbers
  - WINDOW starts from last sequence number seen and extends backwards.
  - WINDOW moves forward to the highest sequence number seen.
  - In case of integrity or privacy, the server will reject message if the sequence number in request body differs from that in cred.
  - requests with sequence #s below the range are silently discarded
    - prevents reply attacks and problems with networks sending duplicates.
    - no danger of denial of service attack because creds are required for attacker to forge requests. Seq# check occurs after the other processing of the AUTH header.
RPCSEC_GSS PROTOCOL

Server replies

• Note that ONC RPC doesn’t have creds on replies, just verifiers.

• The verifier is a gss_sign() of the sequence number of the request.

• Integrity or privacy are specified on the call, the reply is encoded the same way.
RPCSEC_GSS PROTOCOL

Context destruction from client

- Like a regular data call but:
  - Procedure number set to NULLPROC
  - gss_proc in the credential set to RPCSEC_GSS_DESTROY

Reply to context destruction

- Like a regular reply
RPCSEC_GSS PROTOCOL: Preliminary Performance

- AUTH_NONE
- Kerberos V5, just authentication
- Kerberos V5, integrity
- Kerberos V5, privacy

Mike Eisler