OpenSER Admin Course

Security in OpenSER

Voice System SRL

http://www.voice-system.ro

http://www.openser.org

Security mechanisms

- IP authentication for known hosts (not 100% reliable)
- check and authentication user to prevent spam
- don't allow not-authenticated anonymous calls, only authenticated requests from local users allowed
- don't allow direct access to critical resources (gateways, media servers)
- anti-flood detection via pike
- calls from foreign domain should be always considered untrusted unless via secure channel (TLS)

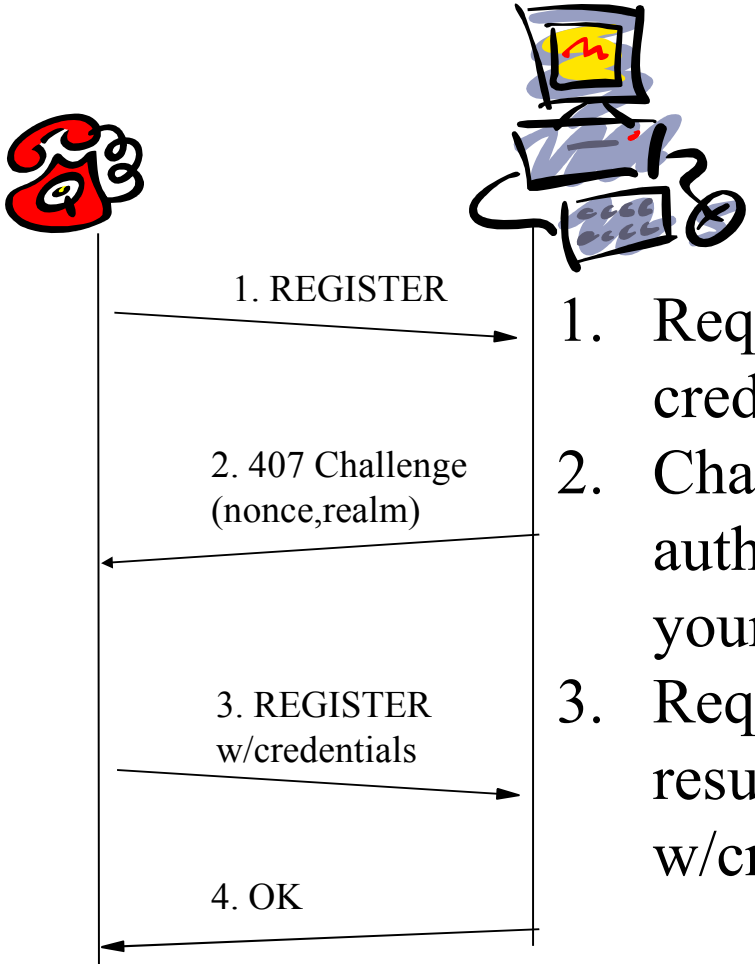
SIP authentication with OpenSER

SIP authentication

- www digest authentication
- IETF RFC2617
- challenge response paradigm
  - send non-authenticated request
  - server replies with challenge
  - send authenticated request
  - server accepts or rejects authentication
- registration authentication (end-point)
  - www-authorize
- call authentication (relay)
  - proxy-authorize
- the auth* modules in OpenSER
  - against database, radius or diameter

SIP authentication

- Protocol:
  - challenge-response using MD5
  - Based on secret shared between client and server
  - No message integrity provided



1. Request w/o credentials
2. Challenge: authenticate yourself
3. Request resubmitted w/credentials

- why to authenticate?
  - verifying user identify is the foundation of all other services and checks (ACLs, profile, attributes, etc)
  - this security check combines with other types of checks (during registration, check also the validity of the uploaded information)

- what to authenticate?
  - all requests pretending to come from one of yours subscriber
  - it is a good practice to authenticate both initial and sequential requests, but there are devices yet not supporting authentication within the dialog

- what kind of backend to use?
  - the backend may be required by integration restrictions
  - you can use multiple backend in parallel for different domains/classes of subscribers
- the authentication check may fail due multiple causes
  - depending of the backend, the check may report the cause of the failure:
    - subscriber does not exist
    - invalid password
    - expired nounce
  - based on failure code, take the appropriate action from script (like do not try to challenge if the subscriber does not exist)

- **auth**
  - common frame for authentication
  - provides functionalities for auth challenge and nonce management
- **auth_db**
  - authentication check against database
- **auth_radius**
  - authentication check against a RADIUS server
- **auth_diameter**
  - authentication check against a DIAMETER server

- subscribers are stored in DB
- password may be store in plain text (insecure) or in a pre-computed format (HA1)

  modparam("auth_db", "password_column", "password")

  versus

  modparam("auth_db", "calculate_ha1", 1)
  modparam("auth_db", "password_column", "ha1")

- authentication means checking the user profile (password) in DB and. in most scenarios, we need more than only the password:
  - OpenSER provides a mechanism to configure a custom set of attributes to be loaded from DB during the authentication process
  - advantage: reduce the number of DB hits

    modparam("auth_db", "load_credentials",
    "$avp(i:12)=rpid; $avp(i:14)=email_address")

Radius backend

- passwords and subscriber data is fetched from a RADIUS server
- requires a radius client library to compile against
  - lib radius client
  - freeradius
- the authentication check is done by the RADIUS server; openser does only the challeng part.
- there is a similar (as for DB) support for fetching additional attributes during authentication, but in this case, the logic is on the RADIUS server side – it decides what extra data to send.

Script example

Credentials protection

Nonce protection

- the nonce generated by OpenSER contain a lifetime – if it is expired, it will not be accepted
  modparam("auth", "nonce_expire", 100)
- as shorter is the lifetime, as smaller is the probability of a client to reuse/fake the nonce

- even so, OpenSER cannot check if a nonce was used more than once during its lifetime.

Credential checking..

```
REGISTER sip:siphub.net.org SIP/2.0
From: <sip:alice@siphub.net>;tag=c775
To: <sip:alice@siphub.net>
Authorization: Digest username="bob",
   realm="siphub.net.de", algorithm="md5",
   uri="sip:siphub.net",
   nonce="3edab81b7a8427be362c2a924f3171d215a8
   f7d3",
   response="4a868f9cbffd2b1f39c778abca78f75b"
```

- Cheating attempt: user “bob” with tries to register as user “alice”
- To do so, the cheater submits proper bob’s credentials but uses alice’s address of record in To header field
- Registrar must enforce a policy that links digest identity to permissible addresses of records

- as the SIP specs allow usage on different SIP ID and Auth ID during authentication, it the responsibility of the platform implementer to do the logic mapping
- What SIP ID is allow a specific Auth ID?? - use the "**uri_db**" module to define the mapping.
  - in most of the case there is a 1 to 1 mapping
  - if a Auth ID can be used by several SIP Ids, that create the mapping in the "**uri**" table
- then use the check functions:

```
# if REGISTER request
if ( check_to() ) {...}
# if other request
if ( check_from() ) {...}
```

- 1 to 1 mapping

  modparam("uri_db","use_uri_table", 0)

  - auth realm must be the same as the From/To domain

- n to 1 mapping

  modparam("uri_db","use_uri_table", 1)

  - auth realm still must be the same as the From/To domain

```
+----+------------+----------+----------+-------------------------+
| id | username | domain  | uri_user | last_modified           |
+----+------------+----------+----------+-------------------------+
|  1 | auth1      | sip.com  | sip1     |0000-00-00 00:00:00|
|  2 | auth1      | sip.com  | sip2     |0000-00-00 00:00:00|
+----+------------+----------+----------+-------------------------+
```

Script example

IP Black lists

- when the script does request forwarding, the destination's IP still may be to discover

  Example:

  seturi("sip:user@voip.domainX.com");
  t_relay();

- solving the destination via DNS (NAPTR, SRV, A) is does after the last script intervention

  ⇒ final IP destination cannot be checked from the script

  ⇒ **security bridge**

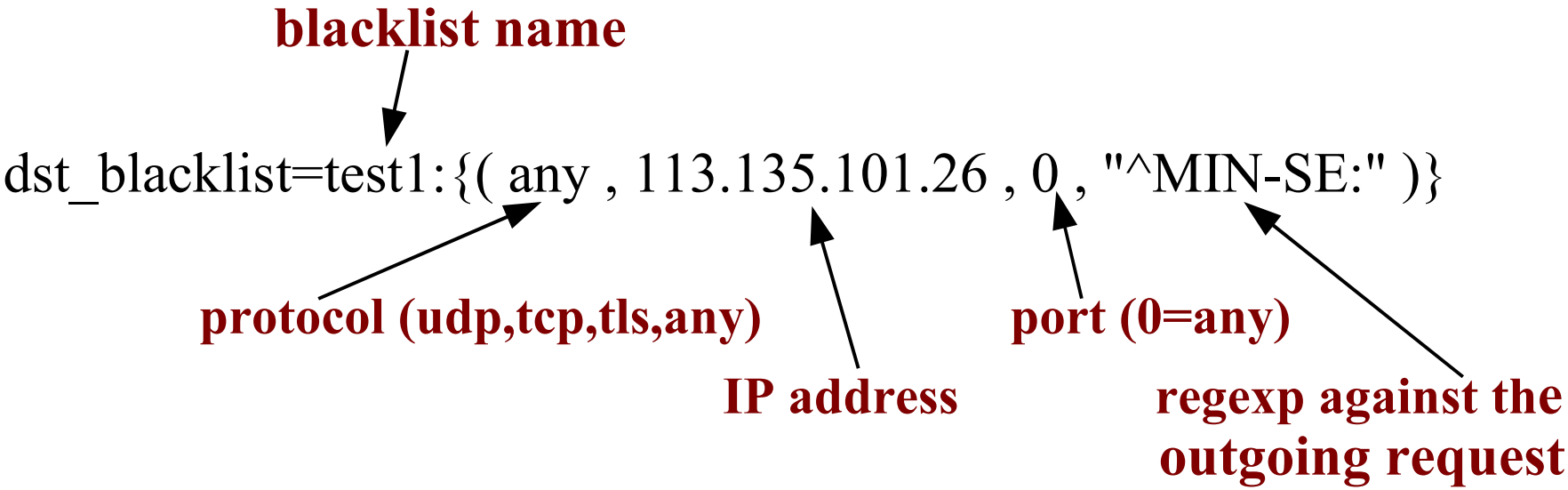
- define lists of IPs to be blocked for relaying

  ⇒ IP black lists
- depending of the destination you determined via script, apply different IP blacklists in order to be sure that the request does not get relayed to a protected IP (via DNS).

Example:

- define a blacklist with the IP address(es) of the PSTN gateway
- if you do a relay based on user location, that activate the blacklist in order to prevent DNS based routing to IP's of the GWs
- if you do relay to a the GWs, do not apply the blacklist as the IPs are allowed in this case.

- global parameter in the configuration script:



Example:

```
dst_blacklist=bl_tls:{( tls , 127.0.0.1 , 5060 , "" ),( tls , 192.168.2.7 ,
    5061 , "^X-auth:" )}
dst_blacklist=bl_gw:{( any , 192.168.2.10 , 0 , "" )}
```

- before doing request relay, just select which blacklist should apply:

  use_blacklist("bl_gw");

- can be used multiple time to set more than one blacklist
- the set lists will be automatically reset at the end of the script

- the requests that hits a blacklist will be automatically rejected by a "473 Destination Filtered" with no failure route triggering

- automatic black listing based on DNS failover
  - when a IP destination is unavailable, it is temporary blacklisted as a self protection mechanism (Ex: tcp setup to an unresponsive host may temporary block a process)
  - the reasons for blacklisting:
    - error at transport layer (TCP only supported)
    - error at SIP layer (timeout with no response, 5xx class reply)
  - it may be combine with dns_failover – if a destination failed, dns will be used to provide an alternative (if any)
- enabled via global parameter

  disable_dns_blacklist = no
- when enabled, it is used all the time

Script example