NATS Authorization Callouts

Metadata	Value
Date	2022-11-28
Author	@derekcollison
Status	Implemented
Tags	server

Context and Problem Statement

Enable an external NATS service to generate NATS authorization and credentials by authenticating connection requests.

Overview

This document describes an implementation for a server-signed authorization request mechanism. The mechanism will be used in both server configuration and operator mode servers.

The server that the client connects to will send an authorization request that includes information about the client, the server, and any client options that were set. This will include traditional authentication parameters such as username and password, tokens, nkeys, or JWT credentials. It may also include TLS information such as fully signed client certificates if presented during the `CONNECT` call.

The authorization request will be sent to `$SYS.REQ.USER.AUTH` on the default `$G` account, or a named account in server configuration mode. In operator mode, it will utilize the public ID of the connected account. The request will be signed by the server's nkey and can be encrypted if configured.

The response to the request will be a signed user JWT that must be signed by the declared issuer in server configuration mode or an account signing key in operator mode. If the request was encrypted, the response can also be encrypted for the issuing server. The request will contain a public user nkey that will be required to be the subject of the user JWT response. An authorization service can request that the client be bound to a different account.

Details and Security

The main security concerns with the auth callout proposal are spoofing of the request or response. To address these concerns, the requests sent from the server will be signed by the server's nkey and can be encrypted. The requests will also contain a public user nkey that must be the subject of the response. The responses, which are user JWTs, will be signed by an account nkey and can be encrypted.

The server will be configured with an authorization callout issuer, which is a public account nkey, in server configuration mode. The response must be signed by this issuer and the subject must be the public user nkey from the request. In operator mode, the response must be signed by the account bound to the client. The server will generate a server keypair and an xkey keypair on startup, which will be used to sign and optionally encrypt the authorization requests.

When the server receives a client connection request, it will call out to the authorization service with a signed request if needed. The request is signed by the server and sent to the `$SYS.REQ.USER.AUTH` subject. This subject is automatically protected in the system for all users on the authorization callout account.

A good pattern to follow is that the authorization service runs isolated in its own account, since the system can bind a user to any authorized account.

For example, given this configuration in the server:

```
listen: "127.0.0.1:-1",
server_name: A,
authorization {
    timeout: 1s
    users: [ { user: "auth", password: "pwd" } ]
    auth_callout {
        issuer: "ABJHLOVMPA4CI6R5KLNGOB4GSLNIY7IOUPAJC4YFNDLQVIOBYQGUWVLA"
        auth_users: [ auth ]
    }
}
```

In this setup, all users are on the `$G` global account. The user auth is predefined. That is allowed for any number of users, using username/password or nkeys, and you can mix them. But since the authorization callout section lists user *auth* as an `auth_user`, the callout mechanism will not be used for that user, and that user can be used for the callout service itself. Note that even if the server configures a user and the system authenticates that user, unless specified in auth_users the callout service will be called regardless. Also since the callout configuration does not specify an account, it defaults to the global account and hence that account will not allow any users to send on `$SYS.REQ.USER.AUTH`.

The request will be signed by the server and currently is an authorization request JWT claim. It includes all client information known by the server, all options that were passed to it from the client application, including any authentication, and optionally TLS information if client certificates are used. It also sets the public user nkey for any response that the server would accept. This assists against replay attacks from the authorization service.

Here is an example matching the above server configuration when a user tries to connect with the following:

```
nc, err = nats.Connect(s.ClientURL(), nats.UserInfo("dlc", "zzz")).
```

AUTH REQUEST

```
{
    "aud": "ABJHLOVMPA4CI6R5KLNGOB4GSLNIY7IOUPAJC4YFNDLQVIOBYQGUWVLA",
    "exp": 1669850675,
    "jti": "WHCQ5EYR7NOB5LCJMRI2YYE3NMQE2G6JCKTU56DFFR6LFH7ZOSEQ",
    "iat": 1669850674,
    "iss": "NB5FCQYBGNXSL27AGZYUX5QZ2KKIFUKVDZCL5R7NIUS4562JT4WEWKQV",
```

```
    "sub": "nats_user_auth_request",
    "nats": {
        "client_info": {
        "client_type": "nats",
        "host": "127.0.0.1",
        "id": 7,
        "kind": "Client",
        "lang": "go",
        "rtt": 199000,
        "server": "A",
        "start": "2022-11-30T23:24:34.239773Z",
        "user": "dlc",
        "ver": "1.19.0"
    },
    "client_opts": {
        "echo": true,
        "headers": true,
        "lang": "go",
        "name": "",
        "no_responders": true,
        "pass": "zzz",
        "pedantic": false,
        "protocol": 1,
        "tls_required": false,
        "user": "dlc",
        "verbose": false,
        "version": "1.19.0"
    },
    "user_nkey": "UBO2MQV67TQTVIRV3XFTEZOACM4WLOCMCDMAWN5QVN5PI2N6JHTVDRON",
    "server_id": {
        "host": "127.0.0.1",
        "id": "NB5FCQYBGNXSL27AGZYUX5QZ2KKIFUKVDZCL5R7NIUS4562JT4WEWKQV",
        "name": "A"
    },
    "version": 2
}
```


An application will receive the encoded authorization claim JWT and can decode it proving that the server `NB5FCQYBGNXSL27AGZYUX5QZ2KKIFUKVDZCL5R7NIUS4562JT4WEWKQV` properly signed the claim.

The response is a signed AuthorizationResponse which will have a User JWT or an error message if not authorized. The signing key must be the private key for the authorization callout issuer or the account in operator mode. Also the subject must be the public user nkey that was presented in the request and the audience (`aud`) needs to be the public key for the issuing server. In the above, the public account key can be seen as the audience for this request and the user_nkey represents the required subject for the signed user JWT response.

The server will make sure that the response is well formed and was properly signed by a possessor of the private account key.

Changing Accounts

A response to an authorization request can override the account that the request was processed in. For example, an authorization service could be set up to handle all authorization requests and determine which account each user should be assigned to. This is primarily applicable in server configuration mode, but can also be used in operator mode. In operator mode, the account that enables authorization callouts will specify which accounts a client can be assigned to. The response user JWT must be signed by the private key of one of these other accounts, so that the server can verify that the owner of the new bound account authorized the switch.

TLS

If client certificates are used, the authorization request will contain information about the TLS state. This will include the cipher and version in use, as well as any client certificates that were specified and verified. These certificates will be included in pem-encoded format, along with any verified chains. If the client certificates were specified but not verified, they will also be included in pem-encoded format.

Example part of client TLS request.

```
"client_tls": {
      "cipher": "TLS_AES_128_GCM_SHA256",
      "verified_chains": [
        [
          "-----BEGIN CERTIFICATE-----
\nMIIDgzCCAmugAwIBAgIUXSH0jKq+6x2WG4RHqN8tATdptokwDQYJKoZIhvcNAQEL\nBQAwTDEkMCIGA1
UEChMbU3luYWRpYSBDb21tdW5pY2F0aW9ucyBJbmMuMRAwDgYD\nVQQLEwdOQVRTLmlvMRIwEAYDVQQDEw
lsb2NhbGhvc3QwHhcNMTkwMjA0MTk1ODAw\nWhcNMjQwMjAzMTk1ODAwWjAlMQ0wCwYDVQQLEwRDTkNGMR
QwEgYDVQQDEwtleGFt\ncGxlLmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKUb4UO/JF
ov\n4DdQ1rk65NL4Uumb2tVNU3R77adJb84GKsu2e5iBJF+rVWcSFe0V6zlrvghlrZAm\n/N7qULnWHbBa
m7Tkm4bJnrqQLCj3j2EkSbmGnwUoTThEPlD/sRECTr/Swhxhc/Lp\nHBfV1yQJaNHOXujwy5iz2zDOzZsm
ZaixpFxIGXGRBBvgWsX0e+YGKDg58ZJr6cfB\nfZ1yqQ5SETT/QNokyFp2tAF/6+37Ir7wSQsI/Y94/YOY
JhLq0+aFXgmsmujetDVU\nmzLoOFBe9DRdhALcPEyiGre9wE3YH/hBWOMDtsXXHjPTjIrq651dxTDhzH1N
ePK3\nRWx2/fcyLqcCAwEAAaOBgzCBgDAOBgNVHQ8BAf8EBAMCBaAwEwYDVR0lBAwwCgYI\nKwYBBQUHAw
IwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQUoS4dpE8Slaffykf+cVSc\ng7IXvcYwHwYDVR0jBBgwFoAUbw
bb4b9Hyi/JdmgKO0hyj272GsswCwYDVR0RBAQw\nAoIAMA0GCSqGSIb3DQEBCwUAA4IBAQChjRkAiIuEXc
o4AkdoLO4wSN0i0b/toZ9b\nU6X91UPCOQMYGLqe81DFYh3JE/+YjrwQYZz5Yb/vRVBC2HmTYkBXdP/74k
Ru4LCz\ncdiVimz4GF2cBfFdxadNEJTQ8GW0fPtOIVwDZtJlNwi7ep58uR9Zld6Zo7FLRSzx\nPtzBP6eE
twMJtVCk6PFluA7MY7k4c/TUW8bK0m9ybHIB8nqKuSWhZQBLdOhISyBz\n/12xzX3An1NUpUaJnnD6ypEy
fd8nZC0oAFC6+SAUMBWxcWYvhE5zcMaZQ3YtJUiC\n0gR5d0Z1sjPYsq4KPow7IaTnzu3+0nLjZUHdU9RM
fehJAxgBm3x0\n-----END CERTIFICATE-----\n",
          "-----BEGIN CERTIFICATE-----
\nMIIDaDCCAlCgAwIBAgIUWyR/qbLooFMu+VcvmQhLAjokntQwDQYJKoZIhvcNAQEL\nBQAwTDEkMCIGA1
UEChMbU3luYWRpYSBDb21tdW5pY2F0aW9ucyBJbmMuMRAwDgYD\nVQQLEwdOQVRTLmlvMRIwEAYDVQQDEw
lsb2NhbGhvc3QwHhcNMTkwMjA0MTk1MDAw\nWhcNMjQwMjAzMTk1MDAwWjBMMSQwIgYDVQQKExtTeW5hZG
lhIENvbW11bmljYXRp\nb25zIEluYy4xEDAOBgNVBAsTB05BVFMuaW8xEjAQBgNVBAMTCWxvY2FsaG9zdD
CC\nASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAN9ryA3PTdAPjC2VQkjy9JXJ\nbOq2GpvGU+2/
gC3TNRXOPJ5ZVy4svV8C9VA9t8gIbQHTYMzBFxyGz0+a/9+DEXot\ncrcVvsqaE5mewU9yjifDqUCGqOn9
fo/zsYwD96KYtukEZ73D1Pyv+7EmkHNYqBKB\n4/1gY/7AuuBcNp5bSpC4isGySZlL0wDjURyjfInrbDdM
Zi3QK2lPZP1okLZG5SCX\n7pQM9riHwnzN94HINTzLTUdjxDBrm0Av9HCEeGT+iXwtXIhNaTkxjEy3a6b2
saVl\nwcaqcZbdGmJVgoncNlA3+277BPOAfbw4X5nGATaWPWxStkqeuhSaxahbCLNJGJcC\nAwEAAaNCME
AwDgYDVR0PAQH/BAQDAgEGMA8GA1UdEwEB/wQFMAMBAf8wHQYDVR0O\nBBYEFG8G2+G/R8ovyXZoCjtIco
9u9hrLMA0GCSqGSIb3DQEBCwUAA4IBAQBmuKij\nsa+RKEoSVrdUWYwAhQJd17I1crhyLjzk3c5k4cXSIU
M0XlGK81GZdPRV5EVym7FN\nn8rhjAYizFykFbIcmiUrNa73jm2QTdMiL8WEzywNB0/X+XSJd+I1VeWOvY
```

```
JMPTiY\nKH/vcNYugVeWUzn6EF+iWnlpS9IHxcDvm6yjMJ242+KQWO7DGkHzbadB/BcryAdz\nv6oBlHTJ
oPqgHUwaHfnTfqCQPTaTACUSFGNEnLuuXvLbbhZlpmLHRoqBiwpa0YQW\n1EAICjLa6q5vSDSBrYJL2tIZ
z2vv/powIWMU1tdGFSALtpMucUH5Opi0Eaa+3cQB\nfvl1Mck/CPY8e4/j\n-----END CERTIFICATE--
---\n"
        ]
      ],
      "version": "1.3"
    },
```

Encryption

Nkeys now support Xkeys, which are x25519 keys that are compatible with NaCl's Seal and Open functions. The nkeys library automatically handles all nonces to prevent misuse.

These xkeys enable the encryption and signing of requests and responses to the authorization service. In server configuration mode, the auth_callout can optionally include a public XKey.

```
auth_callout {
    issuer: "ABJHLOVMPA4CI6R5KLNGOB4GSLNIY7IOUPAJC4YFNDLQVIOBYQGUWVLA"
    auth_users: [ auth ]
    xkey: “XAB3NANV3M6N7AHSQP2U5FRWKKUT7EG2ZXXABV4XVXYQRJGM4S2CZGHT”
}
```

When a server sees the xkey set, it will use the public xkey and the server's private key to asymmetrically encrypt the request. The `ServerID` section of the request will include the server's public xkey so that it is signed, and it will also be included in plaintext in the header of the request message under the key `Nats-Server-Xkey`.

The authorization service can also use the server's public key to optionally encrypt the response. In operator mode, the auth account claim will include the public xkey to be used for encrypting for that account in the `ServerID` portion of the `ExternalAuthorization` property for the account.

In operator mode, the auth account claim will have the public xkey to be used for encrypting for that account in the `ServerID` portion of the `ExternalAuthorization` property for the account.