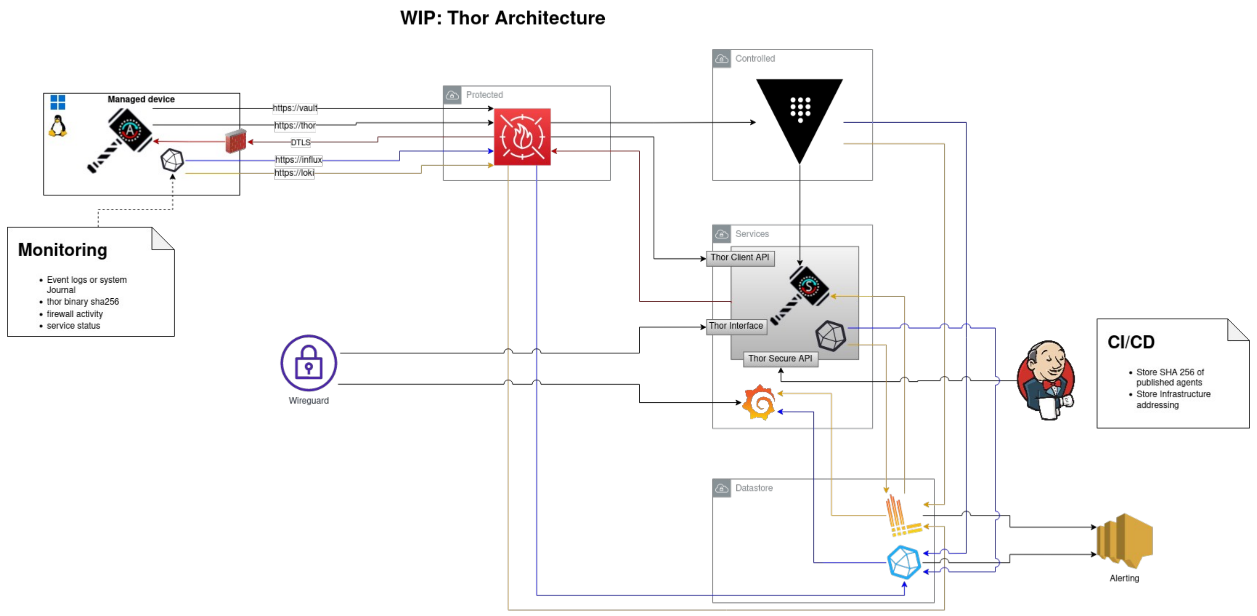
Thor

Managing credential rotation in a zero-trust environment

Martin Proffitt 2/Dec/2021

The following document outlines a mechanism of handling credential rotation on devices and servers where minimal or zero trust policies are in place.



The overall view taken is that no network connected device may own credentials to a Vault system but periodically require access to the vault for the purposes of rotating device passwords or changing application or storage credentials for interconnectivity between component systems.

The proposed design attempts to solve this problem by building a trust model between devices in its own inventory which are added to the list by secondary systems such as CI/CD pipelines via API calls, or manually inserted into the database.

Application function

The application known as Thor implements the concept of trusted entity and should be protected from external interference by secondary means such as a Web application firewall, network firewalls and private, protected access.

Thor offers 4 primary roles:

1. A mechanism to search the Vault audit logs for paths a user has interacted with.
    a. Primary use case: Rotating credentials of a privileged access user.
2. A mechanism to search Vault for all paths containing passwords that have been discovered to be compromised.

3. An agent which carries out the rotation of all device credentials instantly. This entails closing any active user sessions utilising the rotated credential immediately and with no warning. This is deliberate behaviour and should not be altered or bypassed in order to protect the integrity of the system being protected.
4. *An Edge server offering secure communication with an upstream server to manage devices inside a private or protected network. -* ***work not completed***

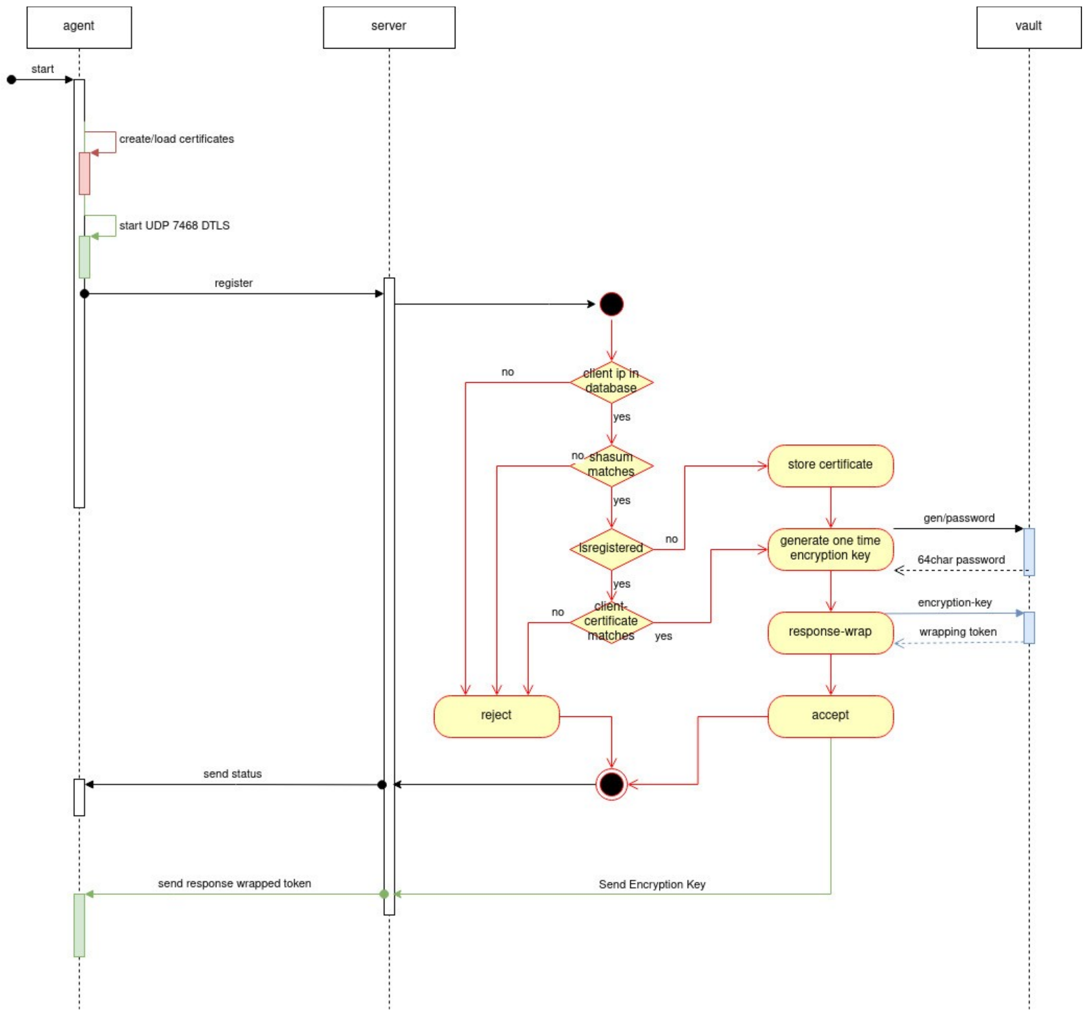
Periodic rotation of all credentials under all paths within a namespace is not a service currently offered by Thor but is a value-add service that can be offered with minor effort to complete. However, there are security concerns to be mitigated before this can be implemented.

Thor achieves this by offering a zero-trust capability inside its own system.

Zero trust stage 1

1. Sha256sums for the windows and linux binaries are added to a database that thor can access. This must be achieved by secondary means (CI/CD pipelines being the obvious choice)
   a. Thor presently uses an embedded BoltDB for this purpose.
   b. linux=$(sha256sum thor | awk '{print $1}')
      windows=$(sha256sum thor.exe | awk '{print $1}')
      curl -kvvvL -H 'Content-Type: application/json' -d '{"shas":
      [{"sha":"$linux","name":"thor"},{"sha":"$windows","name":"thor.exe"}]}'
      https://thor.example.com:9100/api/v1/shasum"
2. Devices are added to the database using the same mechanism as above.
   curl -kvvvL -X POST -H 'Content-Type: application/json' -d '{"devices":["127.0.0.1"]}'
   http://thor.example.com:9100/api/v1/adddevices
   a. The shasum and adddevice API endpoints are protected by IP address. This is normally set to 127.0.0.1 + the address(es) of the build environment and must be done in the config.yaml file, not in the api driven database
3. On startup, an Agent opens UDP Port 7468 and secures it with DTLS (Datagram TLS)
4. The Agent sends a registration request to the Thor server over HTTPS. This registration request contains the public certificate of the DTLS port Thor will use to talk back to the Agent, the Vault namespace that the agent needs to communicate with (or “root” if none), and the sha256sum of the binary.
5. Thor examines its database for the IP the Agent is communicating from. *It is important to note that NAT’d addresses are not supported by Thor. In this instance an Edge server must be used.*
6. If the client IP is discovered in the Database, Thor checks to see if the certificate matches one previously stored. If not, it rejects the connection.
   a. In future: Thor will offer an endpoint for clients to re-register a mutated certificate. *Endpoint to be protected with “I am going to change my ID, this is where I will put it, give me a token/policy to suit)*

Flow:



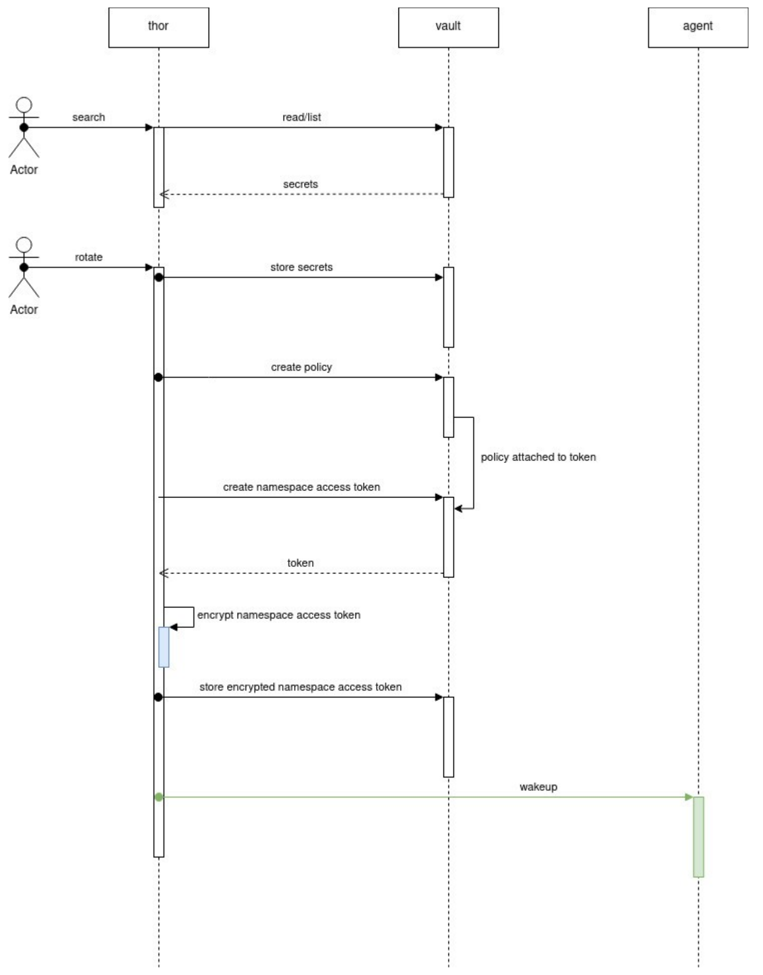
Thor checks the sha256sum against its list of known, trusted versions – this is anti-tampering. Any difference and the registration is rejected – this should be verifiable from the thor logs

7. If no certificate has been stored, or the certificate matches the existing device certificate, Thor generates a single use private encryption key, response wraps this into Vault then sends back the response wrapping token to the client via the DTLS encrypted channel.
8. A “Pending response” response is sent back to the HTTPS channel. This may arrive out of sequence but can be safely ignored by the client.
   Note for future work – *the client should ignore any response it has not sent a request for*

This is the first stage of the zero-trust model.

The purpose of the DTLS UDP channel is to mitigate any attempt at interference. Because we do not know if the IP address of the client is real or masqueraded, we break the trust in the client-initiated connection as soon as the request is initiated and respond back to the agent directly via a separate control channel.

User rotates passwords in Vault



Zero trust stage 2

1. On receipt of the single use response wrapped token, the agent unwraps this against the vault and stores the result in memory. It never gets written to file.
2. The agent then goes to sleep until instructed otherwise.
3. When passwords in a namespace the agent belongs to are rotated, Thor sends a "wakeup" signal back to all agents registered against that namespace.
4. The agent then sends the token back to Thor as part of a verification exercise along with the namespace it needs to access. This communication is carried out over HTTPS.
5. Thor looks up the client IP in its database and checks that the token matches its records
6. If both the address and key match, Thor checks if an access token has been stored for the namespace. If an access token is stored, Thor uses this to generate a time limited token for access to Vault. This token has a single policy attached to it with any path the agent needs to access.
7. This token is encrypted with the key previously sent to the agent during stage 1.
8. The encrypted token is response wrapped into Vault
9. Thor responds on HTTPS with "Pending response"
10. The response wrapped token is sent back to the agent via the DTLS encrypted channel.
11. The agent unwraps the token from Vault
12. The agent decrypts the token using the stage 1 encryption key

13. The agent logs into Vault using this token to recover any passwords that may have changed

If no access token is stored, or it has expired, the client is issued with a STANDBY response at stage 2.6.

All tokens and wrapping tokens are single use. This means that on every request, the agent needs to go through this process.

If a token or IP address does not match the database record, the client is separately instructed to re-register with the system.

Wrapping tokens can be queried against the Audit Log for tampering and the agent cannot access stage 2 if the encryption key is wrong. This gives a solid audit point against man-in-the-middle attacks. If the agent cannot access the wrapping token for stage 1, it can raise a flag either back to Thor, or to a separate system to indicate that a failure has taken place and there is potential interference in the system.

Access tokens from stage 2.6 can only be created from the token used to rotate passwords in Vault. These tokens are stored encrypted back into Vault and are time limited to 5 minutes usage with a policy attached for all paths rotated as part of the last operation.

Edge server

The purpose of an edge server is to grant access to agents in a private or protected network.

The Agent communicates with an edge device which forwards requests on its behalf. These forwarded requests go through the same 2 stage process direct access requests with the edge server holding open its own DTLS port. Edge devices can be chained together where necessary to give full estate coverage.

Vault access

Thor presently requires approle access read/write access to /secure in the security/devsecops namespace for backend operations. This is configurable. Here it will store its own encryption key used to encrypt tokens from stage 2.4 and on a separate path under /secure, stores the encrypted child tokens.

Future iterations of Thor will include alternatives to the approle access such as Instance profile authentication. These have been coded but never tested

Search and rotation tokens from primary functions 1 and 2 are provided at point of access. These are never stored but must be admin tokens for the namespace and must have create/update for the auth/token/create endpoint as the token used for rotation will be used to create the access token for all paths discovered.

Monitoring

Any device running the agent should be monitored heavily.

The following are recommended monitoring points for data collection

- ☐ Service restarts – this should be correlated with CI/CD deployments to ensure a restart can be tied to a pipeline deployment.

- Agent sha256 – The agent sends its own sha as part of the registration request. If the binary is compromised, a possible attack would be to fake the sha either by hard coding a sha or by inline capture of the sha of an uncompromised version. An external system such as telegraf should monitor the binary for changes and alert if the sha does not match a known good sha