Easily Terraforming an EthSigner with an Azure Key Vault HSM Based Key

Itay Podhajcer
Apr 20 · 3 min read

EthSigner, an Ethereum transaction signer which separates private key management from transaction validation, can be used to sign transactions by using keys protected in a variety of storage mechanisms. One of those supported mechanisms is Azure Key Vault and its software and hardware security module (HSM) based key protection offerings.

In this article we will be deploying a serverless EthSigner container using Azure Container Instances and an Azure Key Vault with an HSM based key (which is the more secure option). We will also need to create an Active Directory application and service principal that will be used by EthSigner to authenticate and access the key.

Prerequisites

Will be using Terraform and its `azurerm` provider, so we will be needing the following installed on our workstation:

- Terraform: installation guide is here.
- Azure CLI: installation guide is here.

Example Repository

A complete example Terraform script, which creates the EthSigner container, Active Directory application, Azure Key Vault key and security policies is available in the following GitHub repository:

> cladular/ethsigner-azure-keyvault
>
> Contribute to cladular/ethsigner-azure-keyvault development by creating an account on GitHub.
>
> github.com

The Script

For brevity, I will only cover the area of the Terraform script that specifically handle the creation of the container, key, and the wiring required for integrating the two.

We will start by creating an Azure Active Directory application, service principal and service principal password (which will be later used as the client secret by EthSigner):

```
resource "azuread_application" "this" {
  display_name = local.deployment_name
}

resource "azuread_service_principal" "this" {
  application_id = azuread_application.this.application_id
}

resource "random_string" "this" {
  length  = 16
  special = true
  upper   = false
}

resource "azuread_service_principal_password" "this" {
  service_principal_id = azuread_service_principal.this.id
  value                = random_string.this.result
  end_date             = "2022-01-01T00:00:00Z"
}
```

main.tf hosted with ♡ by GitHub view raw

Note that we use a random string to generate the service principal password.

Next, we will create the Azure Key Vault and HSM based key:

```
resource "azurerm_key_vault" "this" {
  name                = "kv-${local.deployment_name}"
  location            = azurerm_resource_group.this.location
  resource_group_name = azurerm_resource_group.this.name
  tenant_id           = data.azurerm_client_config.current.tenant_id
  sku_name            = "premium"

  access_policy {
    key_permissions = ["Create", "List", "Get", "Delete", "Purge"]
    object_id       = data.azurerm_client_config.current.object_id
    tenant_id       = data.azurerm_client_config.current.tenant_id
  }

  access_policy {
    key_permissions = ["Get", "Sign"]
    object_id       = azuread_service_principal.this.object_id
    tenant_id       = data.azurerm_client_config.current.tenant_id
  }
}

resource "azurerm_key_vault_key" "this" {
  name         = "key-${local.deployment_name}"
  key_vault_id = azurerm_key_vault.this.id
  key_type     = "EC-HSM"
  curve        = "SECP256K1"

  key_opts = [
    "sign",
    "verify"
  ]
}
```

main.tf hosted with ♡ by GitHub view raw

A few things to note here:

- We create a policy that allows the identity that runs the script to create, list, get, delete, and purge keys on the vault, without it, the Terraform script won't be able to complete.

- The second policy, which is connected to the service principal we created earlier, only allows getting keys and signing, as those are the only operations required by EthSigner.
- We use `EC-HSM` as the type, which tells Azure to create a hardware-based key.
- We only allow the key to be used for signing and verifying.

Lastly, we create the EthSigner container, pointing it to Cloudflare's Ethereum `mainnet` gateway as the downstream host (instead of deploying a node on our own), and passing all the Azure Key Vault key related configurations:

```
resource "azurerm_container_group" "this" {
  name                = "aci-${local.deployment_name}"
  location            = azurerm_resource_group.this.location
  resource_group_name = azurerm_resource_group.this.name
  ip_address_type     = "public"
  os_type             = "Linux"

  container {
    name   = local.deployment_name
    image  = "pegasyseng/ethsigner:21.3.0"
    cpu    = "0.5"
    memory = "1.5"

    commands = ["/opt/ethsigner/bin/ethsigner", "azure-signer"]

    ports {
      port     = 8545
      protocol = "TCP"
    }

    volume {
      name       = "client-secrets"
      mount_path = "/mnt/secrets"
      secret = {
        "ethsigner" = base64encode(azuread_service_principal_password.this.value)
      }
    }

    environment_variables = {
      "ETHSIGNER_CHAIN_ID"                      = "1"
      "ETHSIGNER_HTTP_CORS_ORIGINS"             = "*"
      "ETHSIGNER_DOWNSTREAM_HTTP_HOST"          = "cloudflare-eth.com"
      "ETHSIGNER_DOWNSTREAM_HTTP_PORT"          = "443"
```

```
      "ETHSIGNER_DOWNSTREAM_HTTP_TLS_ENABLED"     = "true"
      "ETHSIGNER_AZURE_SIGNER_CLIENT_ID"          = azuread_application.this.application_id
      "ETHSIGNER_AZURE_SIGNER_CLIENT_SECRET_PATH" = "/mnt/secrets/ethsigner"
      "ETHSIGNER_AZURE_SIGNER_KEY_NAME"           = azurerm_key_vault_key.this.name
      "ETHSIGNER_AZURE_SIGNER_KEY_VERSION"        = azurerm_key_vault_key.this.version
      "ETHSIGNER_AZURE_SIGNER_KEY_VAULT_NAME"     = azurerm_key_vault.this.name
      "ETHSIGNER_AZURE_SIGNER_TENANT_ID"          = azurerm_key_vault.this.tenant_id
    }
  }
}
```

main.tf hosted with ♡ by GitHub view raw

All is left now, login to Azure using `az login` and then run the script using `terraform apply`.

Testing the Deployment

We can perform to tests once the EthSigner container is up and running. The first is to make sure that the process is up and running as expected by calling:

```
curl -X GET http://127.0.0.1:8545/upcheck
```

Next, we can check that EthSigner is passing the transactions to the downstream host by calling:

```
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_blockNumber","params":[],"id":51}'
http://127.0.0.1:8545
```

Conclusion

The example discussed in this article, although fully functional, should only be used as a reference in a real-world production deployment. Some areas, mostly security related, require more advanced concepts, such as internal-external networking separation, firewall protection, transport encryption (HTTPS), authentication and more.

Azure Ethereum Software Development Cloud Computing DevOps

About Help Legal

Get the Medium app