How the Agent Gets Its First SPIFFE ID

The agent gets its identity **automatically from SPIRE** based on **where and how it runs** - not from secrets or manual configuration.

Overview

The identity issuance process has 4 main steps:

Step	Who	What
0	Admin	Registers workload entry in SPIRE Server (one-time)
1	Kubernetes	Starts agent pod with specific namespace/SA
2	Agent	Calls SPIRE Workload API (via Unix socket)
3	SPIRE Agent	Attests workload and issues X.509 certificate

Step 0: Admin Pre-Registers Workload Entry (One-Time Setup)

Before any agent can get an identity, an admin must register an **entry** in SPIRE Server:

```
# Register the sales-bot agent
spire-server entry create \
  -spiffeID spiffe://example.org/agent/sales-bot \
  -parentID spiffe://example.org/spire/agent/k8s_psat/demo-cluster/node-1 \
  -selector k8s:ns:ai-agents \
  -selector k8s:sa:sales-bot-sa
```

What each parameter means:

Parameter	Value	Meaning
`-spiffeID`	`spiffe://example.org/agent/sales-bot`	The identity to issue
`-parentID`	`spiffe://...node-1`	Which SPIRE Agent can issue it
`-selector k8s:ns:ai-agents`	Kubernetes namespace	Pod must be in this namespace
`-selector k8s:sa:sales-bot-sa`	Kubernetes ServiceAccount	Pod must use this SA

This creates a rule:

> "Any pod in namespace `ai-agents` using ServiceAccount `sales-bot-sa` gets the SPIFFE ID `spiffe://example.org/agent/sales-bot`"

Step 1: Agent Workload Starts

When the AI agent pod starts in Kubernetes:

```yaml
# Kubernetes Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sales-bot
  namespace: ai-agents        # Matches selector k8s:ns:ai-agents
spec:
  template:
    spec:
      serviceAccountName: sales-bot-sa  # Matches selector k8s:sa:sales-bot-sa
      containers:
        - name: agent
          image: company/sales-bot:v1.2
          volumeMounts:
            - name: spire-agent-socket
              mountPath: /run/spire/sockets
              readOnly: true
      volumes:
        - name: spire-agent-socket
          hostPath:
            path: /run/spire/sockets    # SPIRE Agent's Unix socket
            type: Directory
```

Key points:

- The pod runs in the `ai-agents` namespace
- It uses the `sales-bot-sa` ServiceAccount
- It mounts the SPIRE Agent's Unix socket

Step 2: Agent Requests SVID via Workload API

The agent code calls SPIRE's Workload API (no credentials needed!):

Go Example:

```go
import "github.com/spiffe/go-spiffe/v2/workloadapi"

func getIdentity() {
    ctx := context.Background()

    // Connect to SPIRE Agent via Unix socket
    source, err := workloadapi.NewX509Source(ctx,
        workloadapi.WithClientOptions(
            workloadapi.WithAddr("unix:///run/spire/sockets/agent.sock"),
        ),
```

```
    )
    if err != nil {
        log.Fatal(err)
    }
    defer source.Close()

    // Get the SVID (certificate + private key)
    svid, err := source.GetX509SVID()
    if err != nil {
        log.Fatal(err)
    }

    // Now we have our identity!
    fmt.Printf("My SPIFFE ID: %s\n", svid.ID)
    // Output: My SPIFFE ID: spiffe://example.org/agent/sales-bot
}
```

Python Example:

```
from pyspiffe.workloadapi import X509Source

# Connect to SPIRE Agent
source = X509Source(
    workload_api_addr="unix:///run/spire/sockets/agent.sock"
)

# Get our identity
svid = source.svid
print(f"My SPIFFE ID: {svid.spiffe_id}")
# Output: My SPIFFE ID: spiffe://example.org/agent/sales-bot
```

Step 3: SPIRE Agent Attests the Workload

When the agent calls the Workload API, SPIRE Agent performs attestation:

Attestation Checks:

1. **WHO is calling me?**
   - Check Unix socket peer credentials
   - Process ID (PID)
   - User ID (UID)
2. **WHAT Kubernetes context?**
   - Query Kubernetes API:
   - Pod name: sales-bot-7d4f8b-x2k9m
   - Namespace: ai-agents (MATCH)
   - Service Account: sales-bot-sa (MATCH)
   - Container image: company/sales-bot:v1.2
   - Node: node-1

3. **Match against registered entries:**
   - Entry found
   - SPIFFE ID: spiffe://example.org/agent/sales-bot
   - Selectors match: YES
4. **DECISION: Issue SVID**

Step 4: SPIRE Issues the SVID (X.509 Certificate)

SPIRE returns an X.509 certificate with the SPIFFE ID embedded:

X.509-SVID Certificate Contents:

Field	Value
Subject	O=SPIRE
Subject Alternative Name (SAN)	URI: spiffe://example.org/agent/sales-bot
Not Before	Jan 15, 2026 10:00:00 UTC
Not After	Jan 15, 2026 10:10:00 UTC (10 minutes!)
Public Key	Ed25519
Issuer	SPIRE CA (intermediate)

Key properties:

Property	Value	Why It Matters
Short-lived	Only 10 minutes	Limits damage if compromised
Auto-renewed	SPIRE Agent handles it	No manual rotation needed
No secrets on disk	Private key in memory only	Cannot be stolen from disk
Identity in SAN	SPIFFE URI in certificate	Cryptographically bound

---

Why This Is Secure

Traditional vs SPIFFE Approach

Traditional Approach	SPIFFE/SPIRE Approach
Store API keys in env vars	No secrets to manage
Rotate secrets manually	Auto-rotated every 10 min
Secrets can be leaked/stolen	Identity tied to workload attestation
Anyone with secret = access	Must be the right workload on right node

Attack Scenarios

Attack	Why It Fails
Steal the certificate	Expires in 10 minutes
Copy cert to another machine	Private key never leaves memory
Impersonate from different pod	Attestation checks namespace/SA
Create fake pod	Need K8s RBAC to create in namespace
Man-in-the-middle	Certificate chain validates to SPIRE CA

Different Attestation Methods

SPIRE supports multiple ways to attest workloads depending on your environment:

Kubernetes (Most Common for AI Agents)

```
spire-server entry create \
  -spiffeID spiffe://example.org/agent/sales-bot \
  -parentID spiffe://example.org/spire/agent/k8s_psat/demo-cluster \
  -selector k8s:ns:ai-agents \
  -selector k8s:sa:sales-bot-sa \
  -selector k8s:pod-label:app:sales-bot \
  -selector k8s:container-image:company/sales-bot:v1.2
```

Docker (Non-K8s Environments)

```
spire-server entry create \
  -spiffeID spiffe://example.org/agent/sales-bot \
  -parentID spiffe://example.org/spire/agent/docker \
  -selector docker:label:ai.agent:sales-bot
```

AWS (Cloud Workloads)

```
spire-server entry create \
  -spiffeID spiffe://example.org/agent/sales-bot \
  -parentID spiffe://example.org/spire/agent/aws_iid \
  -selector aws:iamrole:arn:aws:iam::123456789:role/SalesBotRole
```

Unix Process (Bare Metal)

```
spire-server entry create \
  -spiffeID spiffe://example.org/agent/sales-bot \
  -parentID spiffe://example.org/spire/agent/unix \
  -selector unix:uid:1001 \
  -selector unix:path:/opt/agents/sales-bot
```

Complete Example: From Zero to Identity

1. Setup SPIRE (One-Time)

```
# Install SPIRE Server
helm install spire-server spire/spire-server -n spire-system

# Install SPIRE Agent (DaemonSet on each node)
helm install spire-agent spire/spire-agent -n spire-system
```

2. Register the Agent Entry

```
kubectl exec -n spire-system spire-server-0 -- \
  spire-server entry create \
    -spiffeID spiffe://example.org/agent/sales-bot \
    -parentID spiffe://example.org/spire/agent/k8s_psat/demo-cluster \
    -selector k8s:ns:ai-agents \
    -selector k8s:sa:sales-bot-sa
```

3. Create the Agent ServiceAccount

```
# k8s/sales-bot-sa.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sales-bot-sa
  namespace: ai-agents
```

```
kubectl apply -f k8s/sales-bot-sa.yaml
```

4. Deploy the Agent

```
# k8s/sales-bot-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sales-bot
  namespace: ai-agents
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sales-bot
  template:
    metadata:
      labels:
        app: sales-bot
    spec:
      serviceAccountName: sales-bot-sa
      containers:
```

```
        - name: agent
          image: company/sales-bot:v1.2
          env:
            - name: SPIFFE_ENDPOINT_SOCKET
              value: "unix:///run/spire/sockets/agent.sock"
          volumeMounts:
            - name: spire-agent-socket
              mountPath: /run/spire/sockets
              readOnly: true
      volumes:
        - name: spire-agent-socket
          csi:
            driver: "csi.spiffe.io"
            readOnly: true
```

```
kubectl apply -f k8s/sales-bot-deployment.yaml
```

5. Agent Code Gets Identity Automatically

```python
# agent.py
from pyspiffe.workloadapi import X509Source
import os

def main():
    # Connect to SPIRE
    socket = os.environ.get(
        "SPIFFE_ENDPOINT_SOCKET",
        "unix:///run/spire/sockets/agent.sock"
    )
    source = X509Source(workload_api_addr=socket)

    # Get our identity - AUTOMATIC!
    svid = source.svid
    print(f"I am: {svid.spiffe_id}")
    # Output: I am: spiffe://example.org/agent/sales-bot

    # Now use identity to call AgentAuth...
```

Summary

The agent never stores secrets. Its identity comes from:

- **Where it runs** (Kubernetes namespace)
- **How it runs** (ServiceAccount)
- **Cryptographic proof** (SPIRE attestation)

The entire process is automatic once the admin registers the workload entry. The agent just needs to call the Workload API, and SPIRE handles:

- Attestation (proving the agent is who it claims to be)
- Certificate issuance (short-lived X.509 with SPIFFE ID)
- Automatic renewal (before the certificate expires)

Related Documentation

- How AgentAuth Works
- SPIFFE Integration Guide
- Security Hardening