Terraform static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your TERRAFORM code

Using clear-text protocols is security-sensitive

Analyze your code

Security Hotspot Critical aws azure cwe owasp

Clear-text protocols such as `ftp`, `telnet` or non-secure `http` lack encryption of transported data, as well as the capability to build an authenticated connection. It means that an attacker able to sniff traffic from the network can read, modify or corrupt the transported content. These protocols are not secure as they expose applications to an extensive range of risks:

- Sensitive data exposure
- Traffic redirected to a malicious endpoint
- Malware infected software update or installer
- Execution of client side code
- Corruption of critical information

Even in the context of isolated networks like offline environments or segmented cloud environments, the insider threat exists. Thus, attacks involving communications being sniffed or tampered with can still happen.

For example, attackers could successfully compromise prior security layers by:

- Bypassing isolation mechanisms
- Compromising a component of the network
- Getting the credentials of an internal IAM account (either from a service account or an actual person)

In such cases, encrypting communications would decrease the chances of attackers to successfully leak data or steal credentials from other network components. By layering various security practices (segmentation and encryption, for example), the application will follow the *defense-in-depth* principle.

Note that using the `http` protocol is being deprecated by major web browsers.

In the past, it has led to the following vulnerabilities:

- CVE-2019-6169
- CVE-2019-12327
- CVE-2019-11065

Ask Yourself Whether

- Application data needs to be protected against falsifications or leaks when transiting over the network.
- Application data transits over a network that is considered untrusted.
- Compliance rules require the service to encrypt data in transit.
- Your application renders web pages with a relaxed mixed content policy.
- OS level protections against clear-text traffic are deactivated.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

Disabling certificate-based authentication is security-sensitive

Security Hotspot

Assigning high privileges Azure Resource Manager built-in roles is security-sensitive

Security Hotspot

Authorizing anonymous access to Azure resources is security-sensitive

Security Hotspot

Enabling Azure resource-specific admin accounts is security-sensitive

Security Hotspot

- Make application data transit over a secure, authenticated and encrypted protocol like TLS or SSH. Here are a few alternatives to the most common clear-text protocols:
  - Use `ssh` as an alternative to `telnet`
  - Use `sftp`, `scp` or `ftps` instead of `ftp`
  - Use `https` instead of `http`
  - Use `SMTP` over `SSL/TLS` or `SMTP` with `STARTTLS` instead of clear-text SMTP
- Enable encryption of cloud components communications whenever it's possible.
- Configure your application to block mixed content when rendering web pages.
- If available, enforce OS level deatvation of all clear-text traffic

It is recommended to secure all transport channels (even local network) as it can take a single non secure connection to compromise an entire application or system.

Sensitive Code Example

For AWS Kinesis Data Streams server-side encryption:

```
resource "aws_kinesis_stream" "sensitive_stream" {
    encryption_type = "NONE" # Sensitive
}
```

For Amazon ElastiCache:

```
resource "aws_elasticache_replication_group" "example"
    replication_group_id = "example"
    replication_group_description = "example"
    transit_encryption_enabled = false  # Sensitive
}
```

For Amazon ECS:

```
resource "aws_ecs_task_definition" "ecs_task" {
  family = "service"
  container_definitions = file("task-definition.json")

  volume {
    name = "storage"
    efs_volume_configuration {
      file_system_id = aws_efs_file_system.fs.id
      transit_encryption = "DISABLED"  # Sensitive
    }
  }
}
```

For Amazon OpenSearch domains:

```
resource "aws_elasticsearch_domain" "example" {
  domain_name = "example"
  domain_endpoint_options {
    enforce_https = false # Sensitive
  }
  node_to_node_encryption {
    enabled = false # Sensitive
  }
}
```

For Amazon MSK communications between clients and brokers:

```
resource "aws_msk_cluster" "sensitive_data_cluster" {
    encryption_info {
        encryption_in_transit {
            client_broker = "TLS_PLAINTEXT" # Sensitive
            in_cluster = false # Sensitive
        }
    }
}
```

For AWS Load Balancer Listeners:

```
resource "aws_lb_listener" "front_load_balancer" {
  protocol = "HTTP" # Sensitive

  default_action {
    type = "redirect"

    redirect {
      protocol = "HTTP"
    }
  }
}
```

HTTP protocol is used for GCP Region Backend Services:

```
resource "google_compute_region_backend_service" "examp
  name                            = "example-service"
  region                          = "us-central1"
  health_checks                   = [google_compute_reg
  connection_draining_timeout_sec = 10
  session_affinity                = "CLIENT_IP"
  load_balancing_scheme           = "EXTERNAL"
  protocol                        = "HTTP" # Sensitive
}
```

Compliant Solution

For AWS Kinesis Data Streams server-side encryption:

```
resource "aws_kinesis_stream" "compliant_stream" {
    encryption_type = "KMS"
}
```

For Amazon ElastiCache:

```
resource "aws_elasticache_replication_group" "example"
    replication_group_id = "example"
    replication_group_description = "example"
    transit_encryption_enabled = true
}
```

For Amazon ECS:

```
resource "aws_ecs_task_definition" "ecs_task" {
  family = "service"
  container_definitions = file("task-definition.json")

  volume {
    name = "storage"
    efs_volume_configuration {
      file_system_id = aws_efs_file_system.fs.id
      transit_encryption = "ENABLED"
    }
  }
}
```

For Amazon OpenSearch domains:

```
resource "aws_elasticsearch_domain" "example" {
  domain_name = "example"
  domain_endpoint_options {
    enforce_https = true
  }
  node_to_node_encryption {
    enabled = true
  }
}
```

For Amazon MSK communications between clients and brokers, data in transit is encrypted by default, allowing you to omit writing the `encryption_in_transit` configuration. However, if you need to configure it explicitly, this configuration is compliant:

```
resource "aws_msk_cluster" "sensitive_data_cluster" {
    encryption_info {
        encryption_in_transit {
            client_broker = "TLS"
            in_cluster = true
        }
    }
}
```

For AWS Load Balancer Listeners:

```
resource "aws_lb_listener" "front_load_balancer" {
  protocol = "HTTP"

  default_action {
    type = "redirect"

    redirect {
      protocol = "HTTPS"
    }
  }
}
```

HTTPS protocol is used for GCP Region Backend Services:

```
resource "google_compute_region_backend_service" "examp
  name                            = "example-service"
  region                          = "us-central1"
  health_checks                   = [google_compute_reg
  connection_draining_timeout_sec = 10
  session_affinity                = "CLIENT_IP"
  load_balancing_scheme           = "EXTERNAL"
  protocol                        = "HTTPS"
}
```

Exceptions

No issue is reported for the following cases because they are not considered sensitive:

- Insecure protocol scheme followed by loopback addresses like 127.0.0.1 or `localhost`

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- Mobile AppSec Verification Standard - Network Communication Requirements
- OWASP Mobile Top 10 2016 Category M3 - Insecure Communication
- MITRE, CWE-200 - Exposure of Sensitive Information to an Unauthorized Actor
- MITRE, CWE-319 - Cleartext Transmission of Sensitive Information
- Google, Moving towards more secure web
- Mozilla, Deprecating non secure http

Available In:

sonarcloud | sonarqube