Overview of the ZPL Policy Language

Michael Dubno, Mathias Kolehmainen, Danny Hillis, October 2024
(Updated September 8, 2025)

1. Introduction

ZPL (pronounced "zipple") is an acronym for Zero-trust Policy Language. ZPL is used to define security policies for network communication that are independent of network configuration. The policies ZPL defines are enforced across a group of interconnected networks called a ***ZPRnet***. The policies are defined in terms of the ***attributes*** of the communicators. ZPL provides a way to express ***permissions*** that allow communication under specific circumstances and ***denials*** that can override the permissions. A ZPL compiler generates ***enforcement rules*** for implementing the policies.

Any communication through a ZPRnet must be specifically allowed by a permission statement. Policy is written in terms of permissions that state what is allowed to happen and denials that state types of communication that should never take place. For example, a permission statement that allows managed laptops to be used by sales employees to access customer databases could be written like this in ZPL:

```
Allow sales employees on managed laptops to access customer databases.
```

A denial statement might state that the intent of the permissions is that no one who is not an employee is allowed to access a customer database. If a permission accidentally grants a non-employee access to customer databases, the enforcement mechanism will block the communication and report the policy conflict.

The permission example above would be expressed in a single ZPL statement that gives ***endpoints*** with specified attributes (the laptops) operated by ***users*** with specified attributes (the sales employees) access to ***services*** with specified attributes (the customer databases). Since ZPL polices are stated in terms of attributes of the communicators, not network addresses, a database server could be moved anywhere within the ZPRnet, say from one cloud to another or from on-premises to the cloud, without any change in security policy.

Trusted sources are used to determine the attribute values of the communicators. The ZPL compiler combines policy statements with a ***configuration description*** to generate enforcement rules that implement the security policies.

2. Key Concepts

It is helpful to understand a few of the key concepts mentioned above in more detail before diving into the syntax of ZPL. After reading this section, it may be worth rereading the introduction.

2.1. What is a ZPRnet?

ZPL security policies apply to a real or virtual network or to a group of co-managed interconnected networks, such as an enterprise's internal networks and cloud tenancies. The network or group of networks over which the security policies apply is called a ***ZPRnet***. ZPL specifies the policies that are enforced throughout the ZPRnet.

ZPR (pronounced “zipper”) is an acronym for Zero-trust Packet Routing, a method of network-based policy enforcement. It is an IP-layer protocol that can be implemented as a software-defined network over a conventional IP network or can be used directly as the IP-layer protocol of a network. Every policy that can be expressed in ZPL can be enforced by ZPR.

ZPL policy can also be enforced by other methods such as firewalls, NATs, VLANs, smart NICS, etc. Enforcement methods may vary from one part of a ZPRnet to another. For example, a cloud tenancy may enforce the policy with mechanisms that are not available in an on-premises network. The ZPL compiler is able to combine the configuration description with the policy to generate enforcement rules for each part of the network.

ZPL can express some policies that are not easily enforceable by conventional methods. For example, it can express policies on global measures of communication, such as limits on the amount of information a user can move out of a facility over a specific time period. Since non-ZPR enforcement mechanisms may not be able to enforce such policies, the compiler will warn of any policy statement that cannot be enforced in the specific ZPRnet they are intended to protect.

2.2. What are Permissions?

ZPL permission statements grant limited rights for initiating and continuing a flow of packets through the network. The permissions are based on attributes that are associated with the communicators (endpoints, users, services) involved in the packet flow. Permissions can be based on attributes of any authenticated endpoint, user, or service involved in the communication.

2.3. What are Endpoints, Users, and Services?

An endpoint is the device that connects to the ZPRnet. It has an identity and associated attributes. The endpoint may be a virtual device (software adapter, vNIC), a physical device (adapter dongle, NIC), or an entire system that presents a single interface to the ZPRnet, such as a desktop computer or a server. There are always at least two endpoints involved in a packet flow, and a permission may depend on attributes of either one.

A user is an individual or authority that can be associated with a communication flow. Each user has an identity and associated attributes, just like each endpoint. Whenever a single user can be associated with an endpoint, permissions can be based on both the user’s and the endpoint’s attributes.

Services are applications that listen for communications packets and respond and/or act on them. There is always at least one service associated with a flow. Services have identities and associated attributes, and permissions can depend on attributes of a service involved in the flow.

2.4. What is ZPL’s Concept of Identity?

Every endpoint, user or service in a ZPR network (ZPRnet) has an identity that is unique within the ZPRnet. Each identity is authenticated and associated with an endpoint, user or service. The identity is used as a retrieval token to look up associated attributes. Identities are also used for logging and statistics. Policy statements are expressed in terms of the attributes of the communicators, not their identities.

2.5. What is an Attribute?

All endpoints, users and services have ***attributes*** associated with their identity, such as the manager of a device or the roles of an employee. Attributes come in three forms: tags (name only), single-valued attributes (name with one value), and multi-valued attributes (name with a set of values). These are called ***tags***, ***single-valued*** attributes and ***multi-valued*** attributes, respectively. No two attributes of an identity have the same name.

An attribute value is a string. If it contains only numerals, it may be interpreted as an integer. If it contains only numerals and a single period, it may be interpreted as a floating-point number. For example, the single-valued attribute with the name trust-score might have a value that is a string representing an integer. A multi-valued attribute with the name roles could have a set of values that are strings representing different roles. A policy can depend on attribute values or on the presence or absence of tags.

2.6. What are Denials?

Denials are statements of intent that limit what permissions are allowed. They are written using the keyword never. Denials apply to the combined policies of the entire ZPRnet and override any contradictory permissions. Any inconsistency of denials with permissions will always be reported when or before denied communications are attempted.

Every implementation must ensure that the ZPRnet will never deliver denied communication, but different implementations may report policy conflicts at different times. While the compiler can check for conflicts between denials and permissions, denials must also be rechecked while network is operating because attribute values can change.

2.7. What are Trusted Sources?

Trusted Sources are where attribute values come from. They are called “Trusted” because ZPL/ZPR uses these, and only these sources for policy checking. Trusted Sources return attributes and attribute values associated with an identity. Examples of Trusted Sources include Active Directory, LDAP, or cloud-provided services. An API is provided for connecting additional types of trusted sources.

The attribute service caches values from trusted sources, monitors for changes, and can translate identities into formats required by specific sources. The attribute service refreshes its cached values after attribute-dependent intervals specified in the source description. If the source supports notification of changes, the service also uses those notifications to update its cache. An API is also provided to change attribute values through the attribute service, in which case such changes will also update the cached values.

2.8. What is a Configuration Description?

ZPL separates security policies from network configuration. All configuration-specific information is specified in the configuration description, including static IP addresses, protocols, and network topology. The configuration description also specifies trusted sources and what enforcement mechanisms are used for each part of the ZPRnet. The ZPL compiler combines the security policy with the configuration description to produce the appropriate enforcement rules for each part of the network.

3. ZPL Policy

A ZPL policy description is a set of statements, each beginning on a new line and terminated by a period and a blank line. The order of the statements does not matter to the compiler. A statement may contain keywords, names, attributes, whitespace characters and punctuation marks. Extra whitespace characters are ignored.

3.1. ZPL Keywords

ZPL has a fixed set of pre-defined and reserved keywords. Keywords contain only letters of the English alphabet. For example:

`Allow`

`with`

`Define`

A keyword can be written in lowercase, with an initial capital, or in all capitals. To allow future language extensions, all English prepositions that are not keywords are reserved.

3.2. Strings

In ZPL, strings are case-sensitive sequences of UTF-8 characters enclosed in quotation marks, or unquoted sequences of numerals with an optional decimal point. Strings can be enclosed by either a pair of single quotes or a pair of double quotes. ZPL does not distinguish between neutral, forward and backward versions of quote characters (such as U+0022, U+201C, and U+201D). Strings of numerals may be interpreted as integers, or as floating-point numbers if they contain a decimal point.

For example:

`'12-34'` is a string

`"12'34"` is a string with an apostrophe in the middle position

`'1 2 3 4'` is a string with spaces

`1234` is a string that can represent an integer

`123.4` is a string that can represent a floating-point number

Single quotes, double quotes and backslash characters may be included in a string by preceding them with a backslash. For example, here is a string that contains five characters:

`'12\\34'` has a single backslash in the middle position

3.3. Names

A name is a string with an optional namespace prefix. A name can be used for an attribute, a class, an identity or a namespace. Every name exists in a namespace, and the same names in different namespaces are not equivalent. For example, the names that represent identity of endpoints, users and services are in separate namespaces.

A name written without a namespace prefix refers to the name in the default namespace of the context in which it is written.

Here are some examples of valid names without namespace prefixes:

```
employee

Web-server

'name with spaces'
```

The namespaces can be specified explicitly by adding the name of the namespace as a prefix to the name, separated by a period. For example, here is a name in the sales namespace:

```
sales.Web-server
```

In the above example, the namespace name sales is in the default namespace. Namespaces are hierarchical. Here is a name in the support namespace that exists within the sales namespace:

```
sales.support.Web-server
```

The default namespace at the top of the hierarchy is named global, although it is rarely referenced explicitly. Other namespaces are either predefined or created the first time they are referenced in a define statement.

3.4. Attributes

Attributes represent named properties of endpoints, users and services. Typical attributes include roles, group membership, capabilities, and identification numbers. An attribute always has a name. It may have a value or set of values.

Attributes without values are called tags. An attribute name may be any valid name, including names that are delimited by quotes. Their meaning is represented by their presence or absence.

Single-valued attribute values are written after the attribute name and separated by a colon:

```
<attribute-name>:<attribute-value>
```

Attribute values can be strings or integers. For example:

```
passphrase:'Speak friend and enter'

number-usbc-connectors:2
```

Multi-valued attribute values are written in curly brackets after the colon, separated by commas.

```
<attribute-name>:{<attribute-value1>,<attribute-value2>,...}
```

Attribute values can reference other entities (endpoints, users, or services), allowing access to the referenced entity's attributes.

3.5. Classes

ZPL comes with predefined classes of entities that have identity and attributes: `endpoints`, `users` and `services`, and a predefined sub-class of `endpoints` called `servers`, that has a set-valued attribute called `services`. Additional classes can be defined that inherit their attributes from a previously defined class, forming a strict hierarchy. The class definitions can also

specify additional attributes for members of the class. The definition can also restrict the allowable values of the attributes that it inherits. For example, a subclass of `endpoints` called `mobile-devices` can be defined with an attribute named `device-type`. Another class called `mobile-phones` can be defined as a subclass of `mobile-devices` with an attribute named `phone-number` and a required value of "phone" for the `device-type` attribute. Permissions that are given to `mobile-phones` will be restricted to mobile devices that have this attribute value.

To make policy easier to read, the names of classes are synonymous with their standard plurals (adding S or ES). For example, `mobile-phone` and `mobile-phones` can be used interchangeably, as can `service` and `services`, `user` and `users`, `endpoint` and `endpoints`, or `server` and `servers`. Non-standard plurals can also be specified, as described in Section 4.2.

Class names are case-insensitive, but by convention they normally are written in lower case.

3.6. Punctuation

As shown in the example above, statements are terminated by a period.

Punctuation marks are also used for comments. All characters on a line after # or // are ignored by the compiler. For example:

```
# Comments (like this) don't need a period at the end!

// The C/Java comment convention also works.
```

Comments can be on a line by themselves or at the end of lines within a statement.

4. ZPL Policy Statements

ZPL syntax makes it possible to write any policy statement in a form that reads like a grammatical English sentence. This makes policy easy to read, understand, and audit. Like any other computer language, the syntax of ZPL is formally defined.

4.1. Statements that Create Permissions

A permission statement describes conditions under which communication is allowed, based on the attributes of the users, endpoints and services involved in the communication.

Consider the example given earlier of a permission statement that allows managed laptops to be used by sales employees to access customer databases:

```
Allow sales employees on managed laptops to access customer databases.
```

In this example employees, laptops, and databases have been defined as classes of users, endpoints and services, respectively (How this happens is described in section 3.5). The words `sales`, `managed`, and `customer` are tags. For the permission to apply, the user would need to have to have attribute tag `sales`, the laptop would need to have tag `managed`, and the service would need to have tag `customer`. The class definition would also specify various required attributes, which would need to be present for the permission to apply.

Notice that the relationship of the users to the laptop they are using is expressed using the keyword `on`.

This single statement may give permission to many different pairs of laptops and database services. If the permission depends only on the attributes of the users and not the devices they are using, it could be written like this:

```
Allow sales employees to access customer databases.
```

This statement allows any endpoint to access the service, as long as the user has the required attributes.

When the `on` keyword appears before the `to access...` phrase then it describes endpoint attributes of the accessor. When it appears after `to access` and after a service clause it describes endpoint attributes of that which is being accessed. For example:

```
Allow sales employees to access customer databases on sales endpoints.
```

The above statement only applies, and permission is only granted if the communicator offering the database service has an endpoint tag named `sales`.

If employees have a `department` attribute instead of a `sales` tag, the permission would be written differently. In this case, permission could be expressed like this:

```
Allow department:sales employees on managed laptops to access customer
databases.
```

An expression of the form `<name>:<value>` will match either a single-valued attribute with the specified value or a multi-valued attribute with a set of values that contains the specified value. So, in the above example, the permission will be granted to users that have a set of departments that includes `sales`.

In the examples above, the statements may give permission to access any number of services. It is sometimes useful to write a permission that allows access to a single named service. That can be expressed by using the name of the service and the permission statement, like this:

```
Allow HR employees to access Timesheet-database.
```

In this example `Timesheet-database` is the name of a service, defined in the configuration description. By convention, proper names of services are capitalized, although this convention is not enforced by the compiler.

The named `Timesheet-database` service might be provided by a load balancer that connects to a group of servers through the ZPRnet, in which case a permission statement would be needed to allow the load balancer to communicate with the servers that implement the service, like this:

```
Allow cleared government users to access Timesheet-load-balancer.

Allow Timesheet-load-balancer to access Timesheet-database.
```

In other words, both the load balancer and the server endpoints must be separately permissioned.

4.2. Statements that Define New Classes

The only classes that are predefined in ZPL are `endpoints`, `users`, `services`, and `servers` (which are a sub-class of endpoints). ZPL also allows new classes to be defined as variants of an existing

class with additional attributes. For example, a user of type employee might be defined to be a user that has additional attributes:

```
Define employee as a user with an ID-number, roles and optional tags
full-time, part-time, and intern.
```

There are several syntactic constructions illustrated in this statement. The statement uses the keywords a and an, which are ignored by the compiler and are included only for readability. The possible tags are listed after the keyword pair `optional tags`.

Here is a statement that defines `gateway` as a `service` with a single-valued attribute named `external-network-connection`, with a value that specifies the network on the far side of the gateway:

```
Define gateway as a service with an external-network-connection.
```

Any communication outside of the ZPR net would need to go through such a gateway, which might only be permissioned to access specific services.

A definition may also specify a specific value for an attribute that it inherits. For example:

```
Define internet-gateway as a gateway with external-network-connection:public-internet.
```

A definition may also specify an alias, such as a nonstandard plural, for the class that it defines. This is done by adding an AKA (Also Known As) clause before the as keyword:

```
Define mouse AKA mice as peripheral with function:pointing.
```

In this example, both `mouses` and `mice` will be defined as synonyms for `mouse`.

Redefining ZPL keywords or reserved words is not permitted. Attempts to do so will be flagged by the compiler.

4.3. Statements Assert What Communication is Not Allowed

To make the policies easier to understand and audit, ZPL permission statements are normally written as positive allowances. The advantage of this is that the consequences of each statement can be determined without inspecting other statements and it allows components of policy to be compiled incrementally to produce additive enforcement rules. Communication that is not intended to be allowed can be specified by a ***denial***. Denials should not contradict permissions, but if they do the denial takes priority and the system reports the conflict.

ZPL provides a `Never` statement for asserting that a specific type of communication should not be allowed. A `Never` statement looks like an allow statement with the keyword `Never` added at the front:

```
Never allow internet-gateways to access internal services.

Never allow role:intern users to access classified services.
```

(Note: the keyword `Never` is used instead of `Deny` because it is consistent with English grammar, and because `Deny` has different meanings in other policy languages.)

4.4. Statements that Depend on Circumstances

Circumstances are similar to attributes in that they have a name and a typed value, but they describe the state of affairs when communication takes place. Their values are always determined at runtime. The time, date, and measures of the amount of data recently communicated are examples of circumstances.

Any `allow` or `never` statement can be conditioned by circumstances. For example:

```
Never allow backup:nightly servers to access backup-services before 18:00
GMT.

Allow Service2 access to Service1, limited to 10Gb/day.
```

(Note: The syntax for describing circumstances is not yet fully defined.)

4.5. Statements that Cause Signaling

ZPL provides a way to specify a message to be sent to a signal handler whenever a permission matches, by appending an `and signal` clause to an `allow` or `never` statement. For example, this statement will cause a specific type of permissioned access to be reported to a logging service:

```
Allow top-secret users to access top-secret services, and signal "accessing"
to Access-logger.
```

In this example Access-logger is a named service. The message that is sent to Access-logger includes the string “accessing” as well as the identities of all the entities involved in that access. It is sent to the service whenever the allowed communication is initiated. Multiple permissions could potentially allow or deny an access, but only the statement that actually allows or denies the access generates a signal.

5. Delegation of Policy Writing

Because ZPL permissions are additive, different teams can write and compile their policy sections independently, then combine the results. This is helpful if writing and maintenance of communication policy is delegated to different groups within an organization. Large organizations may also require tools to enforce policies about who can write what permissions and what attributes they can access. These meta-policies about policy writing and attribute management are outside the scope of ZPL.

6. Reports

ZPL policies can also be used to create reports. For example, it is possible to report the names of all users that have access to a given resource, or all services that can be accessed from the Internet gateway. These reports may be helpful in the approval or auditing of policies.

7. Revision History

1. Revision as of November 9, 2024, updated to make `Never` an assertion and explain incremental compilation and attribute change tracking (DH)
    a. In section 2.2, state explicitly that an endpoint may support multiple users or services

   b. In sections 2.3 and 2.4, clarify global scope of permissions and assertions
   c. In section 2.7, describe attribute service and how it caches values
   d. Change in sections 4.3 and 4.4, explain `Never` assertions and attribute monitoring
   e. In section 5, explain incremental compilation and the scope of ZPL
2. Revision as of February 24, 2025, remove postfix attribute notation and eliminate any ZPL specific meaning the word endpoint (DH)
   a. In section 2.2, clarify the relationship between endpoints and the associated identities of their devices, users and services
   b. In section 2.3, clarify definition of identity
3. Revision as of May 21, 2025 (DH)
   a. Define and use “endpoint” instead “device” where appropriate.
4. Revision as of July 10, 2025 (DH, MD)
   a. Add denials
   b. Remove assertions
   c. Remove from and multiple keywords
   d. Add on as synonym for with keyword
   e. Add through keyword for load balancers
   f. Allow both single of double quotes for strings
   g. Comments start with # or //
   h. Allow inline comments
   i. Remove Oxford comma text, as there are no use cases for it yet
   j. Add Unicode support
   k. Add early statement example, and reference it
5. Revision as of September 8, 2025 (MK)
   a. In section 1 and 4.1 rewrite ZPL examples that were using “with” to use “on” instead.
   b. In section 4.1 explain in more detail how “on” works.
   c. In section 4.1 remove the “through” keyword and rewrite examples that were using it.