

Interface to Network Security Functions (I2NSF)
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2019

L. Xia
Q. Lin
Huawei
October 21, 2018

I2NSF Security Policy Object YANG Data Model
draft-xia-i2nsf-sec-object-dm-01

Abstract

This document describes a set of policy objects which are reusable and can be referenced by variable I2NSF policy rules. And the YANG data models of these policy objects are provided.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 24, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Internet-Draft

Security Policy Object Data Model

October 2018

Table of Contents

1.	Introduction	2
2.	Requirements Language	3
3.	Terminology	3
4.	Tree Diagrams	3
5.	Policy Object	4
5.1.	Address Object and Address Group	4
5.2.	Service Object and Service Group	5
5.3.	Application Object and Application Group	7
5.4.	User Object, User Group and Security Group	9
5.5.	Time Range Object	11
5.6.	Region Object and Region Group	11
5.7.	Domain Object	12
6.	I2NSF Security Policy Object YANG Module	13
7.	Acknowledgements	46
8.	IANA Considerations	46
9.	Security Considerations	46
10.	References	46
10.1.	Normative References	46
10.2.	Informative References	46
	Authors' Addresses	47

[1.](#) Introduction

As described in [[RFC8329](#)], provisioning to NSFs can be standardized by using policy rules, and I2NSF uses Event-Condition-Action (ECA) model to represent policy rules. According to [[I-D.ietf-i2nsf-terminology](#)], an I2SNF condition is defined as a set of attributes, features, and/or values that are to be compared with a set of known attributes, features, and/or values in order to determine whether the set of actions in that I2NSF policy rules can be executed or not. Information Model of NSFs Capabilities [[I-D.ietf-i2nsf-capability](#)] describes attributes of different condition subclasses. When configuring I2NSF condition clause by attributes or features, it is common to see that the same value of an attribute or the same value set of several attributes are configured for many times. And modifications of the policy rules are also very tedious and time-consuming.

To facilitate the provisioning of NSF instances, this document describes a set of policy objects which are reusable. These policy objects can then be referenced in the condition clause of variable

I2NSF policy rules. A policy object consists of a name attribute that identifies itself and one or several attributes that are typically used together to represent a certain condition. For example, protocol type and port number are usually used together to represent a certain service. Each policy object should be predefined

and named in order to be used in I2NSF policy rules. By defining policy objects, the creation and maintenance of policy rules are greatly simplified.

- o A policy object can be referenced in different policy rules as required to provide re-usability. And a policy rule can reference several policy objects.
- o The modification of a policy object will be propagated to the I2NSF policy rules that reference this object. No modification should be made to the related policy rules.

According to [[I-D.ietf-i2nsf-terminology](#)], there are two kinds of I2NSF policy rules, I2NSF Directly Consumable Policy Rule and I2NSF Indirectly Consumable Policy Rule. The former one can be executed by a network device without translating its content or structure, while the latter one can not be executed by a network device without first translating its content or structure. In this document, policy objects are defined for I2NSF directly consumable policy rules.

[2.](#) Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)].

[3.](#) Terminology

This document uses the terms defined in [[I-D.ietf-i2nsf-terminology](#)] and [[RFC7950](#)].

[4.](#) Tree Diagrams

Tree diagram defined in [[RFC8340](#)] is used to represent the policy objects defined in this document. The meaning of the symbols used in the tree diagrams of following sections and the syntax are as

follows:

- o Groupings, offset by 2 spaces, and identified by the keyword "grouping" followed by the name of the grouping and a colon (":") character.
- o Each node in the tree is prefaced with "+--". Schema nodes that are children of another node are offset from the parent by 3 spaces.
- o Brackets "[" and "]" enclose list keys.

- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" means state data (read-only), and "-u" indicates the use of a predefined grouping.
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a "list" and "leaf-list".
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Curly brackets and a question mark "{...}?" are combined to represent the features that node depends on.

5. Policy Object

This document defines policy objects that are commonly used. Figure 1 shows all the defined policy objects and their relationships.

Policy Object						
Address	Service	Application	User	Security		
Group	Group	Group	Group	Group		Domain

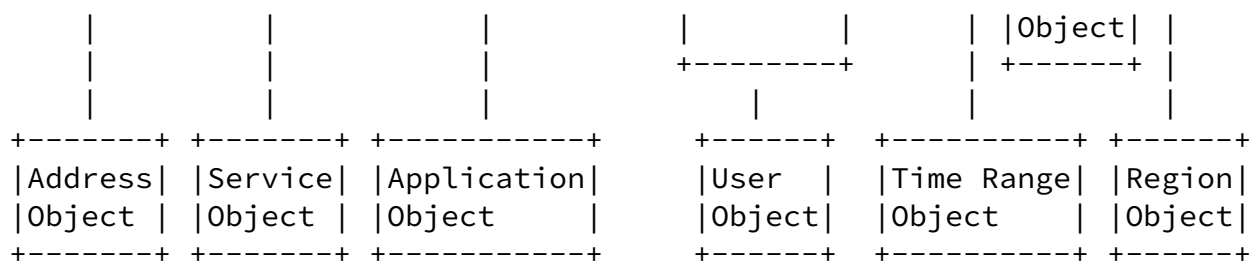


Figure 1: The Policy Objects Overview

5.1. Address Object and Address Group

An address object is identified by a unique name, which contains a set of IPv4/IPv6 addresses or MAC addresses. Several address objects can be organized into an address group object.

This document defines groupings for address objects and address groups.

The tree diagram of address object is:

grouping address-objects:

```

+--rw address-object* [name]
  +--rw name                address-set-name
  +--rw desc?               string
  +--rw vpn-instance?       string
  +--rw elements* [elem-id]
    +--rw elem-id           uint16
    +--rw (object-items)
      +--:(ipv4)
        | +--rw address-ipv4  inet:ipv4-prefix
      +--:(ipv6)
        | +--rw address-ipv6  inet:ipv6-prefix
      +--:(mac)
        | +--rw mac-address   yang:mac-address
        | +--rw mac-address-mask yang:mac-address
      +--:(ipv4-range)
        | +--rw start-ipv4    inet:ipv4-address
        | +--rw end-ipv4      inet:ipv4-address
      +--:(ipv6-range)
        | +--rw start-ipv6    inet:ipv6-address

```

```
    +--rw end-ipv6          inet:ipv6-address
```

The tree diagram of address group is:

grouping address-groups:

```
  +--rw address-group* [name]
    +--rw name          address-set-name
    +--rw desc?         string
    +--rw vpn-instance  string
    +--rw elements* [elem-id]
      +--rw elem-id     uint16
      +--rw addr-object-name address-set-name
```

[5.2.](#) Service Object and Service Group

A service object is a kind of service based on IP, or ICMP, or UDP, or TCP, or SCTP. Several related objects consist a service group. To identify different kinds of services, different kinds of attributes should be specified.

- o UDP, TCP, or SCTP based service is recognized by port number. The source port number and destination port number are used to identify the sending and receiving service respectively.
- o ICMP or ICMPv6 based service is recognized by two header fields in the ICMP/ICMPv6 packets: type field and code field.

- o IP based service is recognized by the value of the protocol field in IP packet header.

Besides, a set of well-known services should be predefined by NSFs as service objects to support direct usage.

The tree diagram of service object is:

grouping service-objects:

```
  +--ro pre-defined-service* [name]
    | +--ro name          string
    | +--ro session-aging-time uint16
  +--rw service-object* [name]
    +--rw name          service-set-name
    +--rw session-aging-time uint16
```

```

+--rw desc?                                string
+--rw items* [id]
  +--rw id                                  uint16
  +--rw (item)
    +--:(tcp-item)
    |   +--rw tcp
    |       +---u port-items
    +--:(udp-item)
    |   +--rw udp
    |       +---u port-items
    +--:(sctp-item)
    |   +--rw sctp
    |       +---u port-items
    +--:(icmp-item)
    |   +--rw (icmp-type)
    |       +--:(name-type)
    |       |   +--rw icmp-name                icmp-name-type
    |       +--:(type-code)
    |       |   +--rw icmp-type-code
    |       |       +--rw icmp-type-number        uint8
    |       |       +--rw icmp-code-number        string
    +--:(icmp6-item)
    |   +--rw (icmp6)
    |       +--:(name-type)
    |       |   +--rw icmp6-name                icmp6-name-type
    |       +--:(type-code)
    |       |   +--rw icmp6-type-code
    |       |       +--rw icmp-type-number        uint8
    |       |       +--rw icmp-code-number        string
    +--:(protocol-id)
    |   +--rw proto-id                          proto-id-range

```

The "port-items" grouping reuses "port-range-or-operator" grouping defined in [[I-D.ietf-netmod-acl-model](#)].

grouping port-items:

```

+--rw source-port
|   +---u pf:port-range-or-operator
+--rw dest-port
|   +---u pf:port-range-or-operator

```

The tree diagram of service group is:

grouping service-groups:

```

+--rw service-group* [name]
  +--rw name                service-set-name
  +--rw desc?               string
  +--rw items* [id]
    +--rw id                uint16
    +--rw service-set-name  service-set-name
```

[5.3.](#) Application Object and Application Group

Due to the diversity and large amount of applications, it is not able to identify a certain application based on protocol type and port number. For example, there are many web applications with different risk levels run on ports 80 and 443 using HTTP and HTTPS, such as web gaming application and web chat application. Protocol type and port number could not distinguish applications using the same application protocol. In this document, category, subcategory, data transmission model, and risk level are used to describe an application. A set of well-known application objects should be predefined in NSFs to support direct reference.

The tree diagram of application object is:

grouping application-objects:


```

+---rw user-defined-application {user-defined-application}?
|   +---rw application* [name]
|       +---rw name                string
|       +---rw label*              string
|       +---rw data-model?         string
|       +---rw category?           string
|       +---rw subcategory?        string
|       +---ro risk-value?         uint32
|       +---rw desc?  string
|       +---rw rule* [name]
|           +---rw name            string
|           +---rw protocol?       protocol
|           +---rw signature
|               |   +---rw mode?    mode
|               |   +---rw direction? direction
|               |   +---rw pattern-type? pattern-type
|               |   +---rw pattern?  string
|               |   +---rw field?    identityref
|           +---rw ip-address*      inet:ip-prefix
|           +---rw port*            inet:port-number
|           +---rw desc?  string
+---ro predefined-application
    +---ro application* [name]
        +---ro name                string
        +---ro protocol*           string
        +---ro risk-value?         uint32
        +---ro label*              string
        +---ro abandon?            boolean
        +---ro multichannel?       boolean
        +---ro data-model?         string
        +---ro category?           string
        +---ro subcategory?        string
        +---ro desc?              string

```

The tree diagram of application group is:

grouping application-groups:

```

+---rw application-group* [name]
    +---rw name                string
    +---rw desc?              string
    +---rw items* [id]
        +---rw id              uint16
        +---rw application-object-name  string

```

5.4. User Object, User Group and Security Group

A user object identifies a person who may access network resources. It is the basis of implementing user-based policy control. The user objects may be created locally on the NSFs, or be imported from third parties, such as authentication servers. User objects that require the same policy enforcement are grouped as user group objects or security group objects. The user group objects are organized as a hierarchical structure. A security group object consists of user objects from different user group objects that require the same policy enforcement.

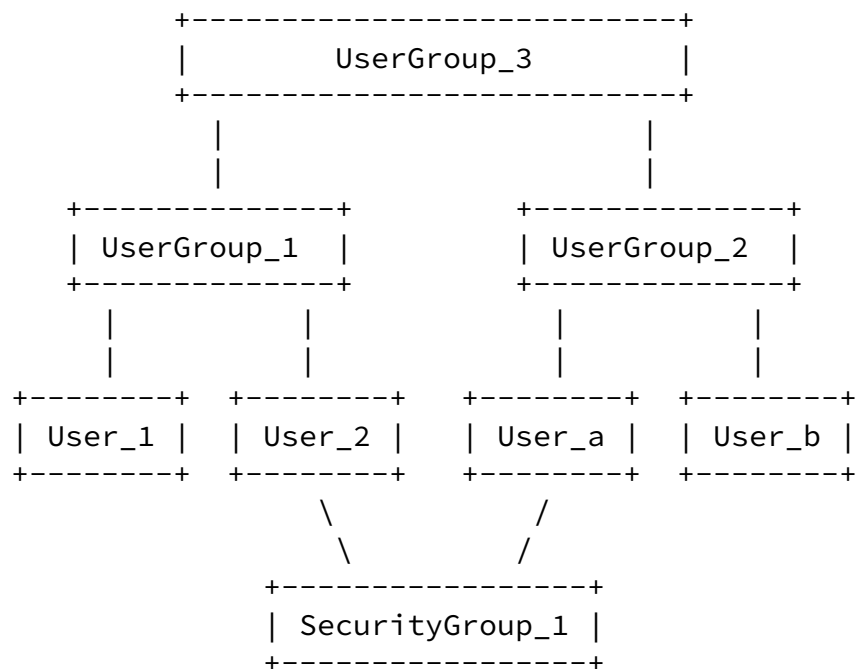


Figure 2: Example of User, User Group and Security Group Structure

The tree diagram of user object is:

grouping user-objects:

```
+++rw user-object* [name aaa-domain]
  +++rw name          user-name
  +++rw aaa-domain    string
  +++rw desc?         string
  +++rw password?     ianach:crypt-hash
  +++rw parent-user-group      user-group-name
  +++rw parent-security-group  user-security-group-name
  +++rw expiration-time
  |   +---:(expiration-type)
  |   |   +++rw (never-expire)
  |   |   |   +++rw never-expire
  |   |   +++rw (expire-after-this-time)
  |   |   |   +++rw expiration-time  yang:date-and-time
  +++rw ip-mac-binding
  |   +---: (bind-state)
  |   |   +++rw (no-binding)
  |   |   |   +++rw no-binding
  |   |   +++rw (binding)
  |   |   |   +++rw bind-mode          ip-mac-binding-type
  |   |   |   +++rw ip-binding*        inet:ipv4-address
  |   |   |   +++rw mac-binding*       yang:mac-address
  |   |   |   +++rw ip-mac-bindings [ip-binding]
  |   |   |   |   +++rw ip-binding    inet:ipv4-address
  |   |   |   |   +++rw mac-binding   yang:mac-address;
```

The tree diagram of user group is:

grouping user-groups:

```
+++rw user-group* [name]
  +++rw name          user-group-name
  +++rw desc?         string
  +++rw parent-user-group  user-group-name
```

The tree diagram of security group is:

grouping security-groups:

```
+++rw security-group* [name]
  +++rw name          user-security-group-name
```

```

+--rw desc? string
+--rw parent-security-group*? user-security-group-name
+--rw filter-action
  +--:(filter-type)
    +--rw (static)
      | +--rw static
    +--rw (dynamic)
      +--rw dynamic
      +--rw filter-rule* string

```

[5.5.](#) Time Range Object

There are two kinds of time ranges: periodic time range and absolute time range. A periodic time range occurs every week. An absolute time range occurs only once.

The tree diagram of time range object is:

grouping time-range-objects:

```

+--rw time-range-object* [name]
  +--rw name time-range-name
  +--rw period-time* [start end]
    | +--rw start hour-minute-second
    | +--rw end hour-minute-second
    | +--rw weekday weekday
  +--rw absolute-time* [start end]
    +--rw start yang:date-and-time
    +--rw end yang:date-and-time

```

[5.6.](#) Region Object and Region Group

A region object is a set of public IP addresses that are assigned to a certain geographic location. A region group consists of a set of region objects.

The tree diagram of region object is:

grouping region-objects:

```

+---ro pre-defined-region* [name]
|   +---ro name                region-name
|   +---ro desc?               string
|   +---ro region-ipv4-address
|   |   +---ro address-ipv4*   inet:ipv4-prefix
|   |   +---ro address-ipv4-range* [start-ipv4 end-ipv4]
|   |       +---ro start-ipv4   inet:ipv4-address
|   |       +---ro end-ipv4     inet:ipv4-address
|   +---ro region-ipv6-address {support-ipv6-address}?
|       +---ro address-ipv6*   inet:ipv6-prefix
|       +---ro address-ipv6-range* [start-ipv6 end-ipv6]
|           +---ro start-ipv6   inet:ipv6-address
|           +---ro end-ipv6     inet:ipv6-address
+---rw user-defined-region* [name]
    +---rw name                region-name
    +---rw desc?               string
    +---rw coordinate
    |   +---rw longitude        region-longitude
    |   +---rw latitude         region-latitude
    +---rw region-ipv4-address
    |   +---rw address-ipv4*   inet:ipv4-prefix
    |   +---rw address-ipv4-range* [start-ipv4 end-ipv4]
    |       +---rw start-ipv4   inet:ipv4-address
    |       +---rw end-ipv4     inet:ipv4-address
```

```

+--rw region-ipv6-address {support-ipv6-address}?
  +--rw address-ipv6*   inet:ipv6-prefix
  +--rw address-ipv6-range* [start-ipv6 end-ipv6]
    +--rw start-ipv6   inet:ipv6-address
    +--rw end-ipv6     inet:ipv6-address

```

The tree diagram of region group is:

```

grouping region-groups:
  +--rw region-group* [name]
    +--rw name          region-name
    +--rw desc?         string
    +--rw region-name*  region-name
    +--rw region-group-name* region-name

```

[5.7.](#) Domain Object

The tree diagram of domain object is:

```

grouping domain-objects:
  +--rw domain-object* [name]
    +--rw name          domain-name
    +--rw desc?         string
    +--rw domain*       string

```

[6.](#) I2NSF Security Policy Object YANG Module

```

<CODE BEGINS> file "ietf-policy-object@2018-10-12.yang"
module ietf-policy-object {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-policy-object";
  prefix policy-object;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991 - Common YANG Data Types.";

```

```

}

import ietf-yang-types {
    prefix yang;
    reference
        "RFC 6991 - Common YANG Data Types.";
}

import iana-crypt-hash {
    prefix ianach;
    reference
        "RFC7317 - A YANG Data Model for System Management.";
}

import ietf-packet-fields {
    prefix pf;
    reference
        "draft-ietf-netmod-acl-model - Network Access Control List (ACL) YANG Data Model";
}

organization
    "IETF I2NSF (Interface To Network Security Functions) Working Group";

contact
    "WG Web: http://tools.ietf.org/wg/i2nsf/
    WG List: i2nsf@ietf.org

    Editor: Liang Xia
           frank.xialiang@huawei.com
    Editor: Qiushi Lin

```

linqiushi@huawei.com";

```

description
    "This YANG module defines groupings that are used by ietf-policy-object YANG module";

revision 2018-10-12 {
    description "Initial version.";
    reference
        "draft-xia-i2nsf-sec-object-dm-01";
}

```

```

/*
* Typedefs for address object and address group
*/
typedef address-set-name {
    type string {
        length "1..63";
    }
    description
        "This type represents an address object or an address group name.";
}

/*
* Typedefs for service object and service group
*/
typedef service-set-name {
    type string {
        length "1..63";
    }
    description
        "This type represents a service object or a service group name.";
}

typedef port-range {
    type uint16;
    description
        "This type represents a port number, which may be a start port of a port
}

typedef proto-id-range {
    type uint8 {
        range "0..255";
    }
    description
        "This type represents the range of protocol id.";
}

typedef icmp-name-type {

```

```

    type enumeration {
        enum echo {
            description
                "ICMP type number 8, ICMP code number 0";

```



```

    }
enum echo-reply {
    description
    "ICMP type number 0, ICMP code number 0";
}
enum fragmentneed-DFset {
    description
    "ICMP type number 3, ICMP code number 4";
}
enum host-redirect {
    description
    "ICMP type number 5, ICMP code number 1";
}
enum host-tos-redirect {
    description
    "ICMP type number 5, ICMP code number 3";
}
enum host-unreachable {
    description
    "ICMP type number 3, ICMP code number 1";
}
enum information-reply {
    description
    "ICMP type number 16, ICMP code number 0";
}
enum information-request {
    description
    "ICMP type number 15, ICMP code number 0";
}
enum net-redirect {
    description
    "ICMP type number 5, ICMP code number 0";
}
enum net-tos-redirect {
    description
    "ICMP type number 5, ICMP code number 2";
}
enum net-unreachable {
    description
    "ICMP type number 3, ICMP code number 0";
}
enum parameter-problem {
    description
    "ICMP type number 12, ICMP code number 0";
}

```

```
    }
    enum port-unreachable {
        description
        "ICMP type number 3, ICMP code number 3";
    }
    enum protocol-unreachable {
        description
        "ICMP type number 3, ICMP code number 2";
    }
    enum reassembly-timeout {
        description
        "ICMP type number 11, ICMP code number 1";
    }
    enum source-quench {
        description
        "ICMP type number 4, ICMP code number 0";
    }
    enum source-soute-failed {
        description
        "ICMP type number 3, ICMP code number 5";
    }
    enum timestamp-reply {
        description
        "ICMP type number 14, ICMP code number 0";
    }
    enum timestamp-request {
        description
        "ICMP type number 13, ICMP code number 0";
    }
    enum ttl-exceeded {
        description
        "ICMP type number 11, ICMP code number 0";
    }
}
description
    "This type is an enumeration of ICMP type names.";
}

typedef icmp6-name-type {
    type enumeration {
        enum redirect {
            description
            "ICMPv6 type number 137, ICMPv6 code number 0";
        }
        enum echo {
            description
            "ICMPv6 type number 128, ICMPv6 code number 0";
```

}

```
enum echo-reply {
    description
    "ICMPv6 type number 129, ICMPv6 code number 0";
}
enum err-Header-field {
    description
    "ICMPv6 type number 4, ICMPv6 code number 0";
}
enum frag-time-exceeded {
    description
    "ICMPv6 type number 3, ICMPv6 code number 1";
}
enum hop-limit-exceeded {
    description
    "ICMPv6 type number 3, ICMPv6 code number 0";
}
enum host-admin-prohib {
    description
    "ICMPv6 type number 1, ICMPv6 code number 1";
}
enum host-unreachable {
    description
    "ICMPv6 type number 1, ICMPv6 code number 3";
}
enum neighbor-advertisement {
    description
    "ICMPv6 type number 136, ICMPv6 code number 0";
}
enum neighbor-solicitation {
    description
    "ICMPv6 type number 135, ICMPv6 code number 0";
}
enum network-unreachable {
    description
    "ICMPv6 type number 1, ICMPv6 code number 0";
}
enum packet-too-big {
    description
    "ICMPv6 type number 2, ICMPv6 code number 0";
}
```

```

enum port-unreachable {
    description
    "ICMPv6 type number 1, ICMPv6 code number 4";
}
enum router-advertisement {
    description
    "ICMPv6 type number 134, ICMPv6 code number 0";
}

```

```

enum router-solicitation {
    description
    "ICMPv6 type number 133, ICMPv6 code number 0";
}
enum unknown-ipv6-opt {
    description
    "ICMPv6 type number 4, ICMPv6 code number 2";
}
enum unknown-next-hdr {
    description
    "ICMPv6 type number 4, ICMPv6 code number 1";
}
}
description
    "This type is an enumeration of ICMPv6 type names.";
}

/*
 * Typedefs for application object and application group
 */
typedef protocol {
    type enumeration {
        enum tcp {
            description
            "tcp protocol";
        }
        enum udp {
            description
            "udp protocol";
        }
        enum any {
            description
            "any protocol";
        }
    }
}

```

```

    }
}
description
    "The protocol of user-defined application rule:tcp/udp/any.";
}

```

```

typedef mode {
    type enumeration {
        enum flow {
            description
                "Keyword exists in multiple packets";
        }
        enum packet{
            description
                "Keyword exists in one packet";
        }
    }
}

```

```

    }
}
description
    "The mode of keyword identification to identify user-defined applications";
}

```

```

typedef direction {
    type enumeration {
        enum request {
            description
                "Request indicates that data to the server is monitored to detect app";
        }
        enum response {
            description
                "Response indicates that data from the server is monitored to detect";
        }
        enum both {
            description
                "Both indicates that data from and to the server is monitored to detect";
        }
    }
}
description
    "The data flow direction that is monitored to identify user-defined applications";
}

```

```

typedef pattern-type {

```

```

type enumeration {
    enum regular {
        description
            "Regular indicates that the keyword of the match pattern is not a fix
    }
    enum plain {
        description
            "Plain indicates that the keyword of the match pattern is a fixed str
    }
}
description
    "The match pattern of the user-defined application rule. If the keyword i
}

```

```

/*
* Typedefs for user object, user group, and security group
*/

```

```

typedef user-name {
    type string {
        length "1..63";
    }
}

```

```

    }
    description
        "This type represents a user name.";
}

typedef user-group-name {
    type string {
        length "1..63";
    }
    description
        "This type represents a user group name.";
}

typedef user-security-group-name {
    type string {
        length "1..63";
    }
    description
        "This type represents a security group name.";
}

```

```

}

typedef ip-mac-binding-type {
    type enumeration {
        enum bidirectional {
            description
                "Bidirectional binding indicates that a user must use the specified I
        }
        enum unidirectional {
            description
                "Unidirectional binding indicates that a user must use the specified
        }
    }
    description
        "The user and IP/MAC address binding mode: bidirectional, or unidirection
}

/*
 * Typedefs for time range object
 */
typedef time-range-name {
    type string {
        length "1..32";
    }
    description
        "This type represents a time-range name.";
}

```

```

typedef hour-minute-second {
    type string {
        pattern '\d{1,2}:\d{1,2}:\d{1,2}';
    }
    description
        "The representation of Hour, Minute, Sencond - hh:mm:ss";
}

typedef weekday {
    type enumeration {
        enum sunday {
            description

```

```

        "Sunday of the week";
    }
    enum monday {
        description
        "Monday of the week";
    }
    enum tuesday {
        description
        "Tuesday of the week";
    }
    enum wednesday {
        description
        "Wednesday of the week";
    }
    enum thursday {
        description
        "Thursday of the week";
    }
    enum friday {
        description
        "Friday of the week";
    }
    enum saturday {
        description
        "Saturday of the week";
    }
}
description
    "A type modeling the weekdays in the Greco-Roman tradition.";
}

```

```

/*
 * Typedefs for region object and region group
 */
typedef region-name {

```

```

    type string;
    description
        "This type represents a location or location set name.";
}

```



```

typedef region-longitude {
    type string;
    description
        "This type represents a region longitude number(-180.00 - 180.00).";
}

typedef region-latitude {
    type string;
    description
        "This type represents a region latitude number(-90.00 - 90.00).";
}

typedef domain-name {
    type string {
        length "1..63";
    }
    description
        "This type represents a domain object name.";
}

/*
 * Identities for application object and application group
 */
identity protocol-field {
    description
        "Base type of protocol field.";
}

identity general-payload {
    base protocol-field;
    description
        "The field of signature is general-payload.";
}

identity http-method {
    base protocol-field;
    description
        "The field of signature is http.method.";
}

identity http-uri {
    base protocol-field;
}

```

```

    description
        "The field of signature is http.uri.";
}

identity http-user-agent {
    base protocol-field;
    description
        "The field of signature is http.user-agent.";
}

identity http-host {
    base protocol-field;
    description
        "The field of signature is http.host.";
}

identity http-content-type {
    base protocol-field;
    description
        "The field of signature is http.content-type.";
}

identity http-cookie {
    base protocol-field;
    description
        "The field of signature is http.cookie.";
}

identity http-body {
    base protocol-field;
    description
        "The field of signature is http.body.";
}

/*
 * Features for application object
 */
feature user-defined-application {
    description
        "This feature means the NSF supports user-defined application function th
}

/*
 * Features for region object
 */
feature support-ipv6-address {
    description

```

Internet-Draft

Security Policy Object Data Model

October 2018

```
    "This feature means the NSF support configuring IPv6 addresses for Region  
  }
```

```
  /*
```

```
  * Groupings for address object and address group
```

```
  */
```

```
  grouping address-objects {
```

```
    list address-object {
```

```
      key "name";
```

```
      leaf name {
```

```
        type address-set-name;
```

```
        description
```

```
          "The name of the address object.";
```

```
      }
```

```
      leaf desc {
```

```
        type string{
```

```
          length "1..127";
```

```
        }
```

```
        description
```

```
          "The description of the address object.";
```

```
      }
```

```
      leaf vpn-instance {
```

```
        type string;
```

```
        description
```

```
          "The name of the vpn-instrance.";
```

```
      }
```

```
      list elements {
```

```
        key "elem-id";
```

```
        leaf elem-id {
```

```
          type uint16;
```

```
          description
```

```
            "The id of the element in address object.";
```

```
        }
```

```
      choice object-items {
```

```
        case ipv4 {
```

```
          leaf address-ipv4 {
```

```
            type inet:ipv4-prefix;
```

```
            description
```

```
              "A set of IPv4 addresses that are represented by an IPv4 address";
```

```
          }
```

```
        }
```

```
        case ipv6 {
```

```

    leaf address-ipv6 {
      type inet:ipv6-prefix;
      description
        "A set of IPv6 addresses that are represented by an IPv6 address"
    }
  }
}

```

```

case mac {
  leaf mac-address {
    type yang:mac-address;
    description
      "MAC address. This leaf is combined with the mac-address-mask leaf"
  }
  leaf mac-address-mask {
    type yang:mac-address;
    description
      "If this leaf is not presented, the mac-address leaf represents"
  }
}
case ipv4-range {
  leaf start-ipv4 {
    type inet:ipv4-address;
    description
      "The start IPv4 address of an IPv4 address range."
  }
  leaf end-ipv4 {
    type inet:ipv4-address;
    description
      "The end IPv4 address of an IPv4 address range."
  }
}
case ipv6-range {
  leaf start-ipv6 {
    type inet:ipv6-address;
    description
      "The start IPv6 address of an IPv6 address range."
  }
  leaf end-ipv6 {
    type inet:ipv6-address;
    description
      "The end IPv6 address of an IPv6 address range."
  }
}

```

```

    }
    description
    "Diffrent types of addresses: IPv4, IPv6, MAC.";
  }
    description
    "A list of addresses that belong to a specific address object.";
  }
    description
    "A list of address objects.";
  }
description
    "This grouping represents a list of address objects. An address object is
}

```

```

grouping address-groups {
  list address-group {
    key "name";
    leaf name {
      type address-set-name;
      description
        "The name of the address group.";
    }
    leaf desc {
      type string{
        length "1..127";
      }
      description
        "The description of the address group.";
    }
    leaf vpn-instance {
      type string;
      description
        "The name of the vpn-instrance.";
    }
    list elements {
      key "elem-id";
      leaf elem-id {
        type uint16;
        description
          "The id of the element in address group.";
      }
      leaf addr-object-name {

```

```

        type address-set-name;
        mandatory true;
        description
            "The name of the address object that consists the address group.";
    }
        description
            "A list of address objects that consists the address group object.";
    }
        description
            "A list of address group objects.";
    }
description
    "An address group object is comprised of several address objects that req
}

/*
* Groupings for service object and service group
*/
grouping port-items {

```

```

    container source-port {
        uses pf:port-range-or-operator;
        description
            "Source port definition from range or operator.";
    }
    container dest-port {
        uses pf:port-range-or-operator;
        description
            "Destination port definition from range or operator.";
    }
description
    "This grouping consists of the source port numbers and destination port n
}

grouping service-objects {
    list pre-defined-service {
        key "name";
        config false;
        leaf name {
            type service-set-name;

```

```

    config false;
    description
        "The name of the predefined service object.";
}
leaf session-aging-time {
    type uint16;
    units second;
    config false;
    description
        "The aging time of the predefined service object.";
}
description
    "A list of the predefined service objects.";
}
list service-object {
    key "name";
    leaf name {
        type service-set-name;
        description
            "The name of the service object.";
    }
    leaf session-aging-time {
        type uint16;
        units second;
        description
            "The aging time of the service object.";
    }
}

```

```

leaf desc {
    type string{
        length "1..127";
    }
    description
        "The description of the service object.";
}
list items {
    key "id";
    leaf id {
        type uint16;
        description
            "The id of the element in service object.";
    }
}

```

```

choice item {
  case tcp-item {
    container tcp {
      uses port-items;
      description
        "TCP based service is recognized by source port number and dest
    }
  }
  case udp-item {
    container udp {
      uses port-items;
      description
        "UDP based service is recognized by source port number and dest
    }
  }
  case sctp-item {
    container sctp {
      uses port-items;
      description
        "SCTP based service is recognized by source port number and des
    }
  }
  case icmp-item {
    choice icmp-type {
      case name-type {
        leaf icmp-name {
          type icmp-name-type;
          mandatory true;
          description
            "The ICMP based service is identified by the predefined ICM
        }
      }
      case type-code {
        container icmp-type-code {

```

```

    leaf icmp-type-number {
      type uint8;
      mandatory true;
      description
        "The ICMP type number.";
    }
    leaf icmp-code-number {

```



```

        type string;
        mandatory true;
        description
            "The ICMP code number.";
    }
    description
        "The ICMP based service is recognized by two header fields";
}
}

        description
            "The ICMP based service object and its attributes.";
    }
}
case icmp6-item {
    choice icmp6-type {
        case name-type {
            leaf icmp6-name {
                type icmp6-name-type;
                mandatory true;
                description
                    "The ICMPv6 based service is identified by the predefined I";
            }
        }
        case type-code {
            container icmp6-type-code {
                leaf icmp6-type-number {
                    type uint8;
                    mandatory true;
                    description
                        "The ICMPv6 type number.";
                }
                leaf icmp6-code-number {
                    type string;
                    mandatory true;
                    description
                        "The ICMP code number.";
                }
            }
            description
                "The ICMPv6 based service is recognized by two header field";
        }
    }
}

```

```

        description
        "The ICMPv6 based service object and its attributes.";
    }
    description
    "The ICMPv6 based service object and its attributes.";
}
case protocol-id {
    leaf proto-id {
        type proto-id-range;
        mandatory true;
        description
        "IP based service is identified by the value of the protocol fi
    }
}
    description
    "Diffrent types of protocols for service definition.";
}
    description
    "A list of service items that consist an service object.";
}
description
    "A list of user defined service objects.";
}
description
    "A list of the predefined service objects and user defined service object
}

grouping service-groups {
    list service-group {
        key "name";
        leaf name {
            type service-set-name;
            description
            "The name of the service group.";
        }
        leaf desc {
            type string{
                length "1..127";
            }
            description
            "The description of the service group.";
        }
    }
    list items {
        key "id";
        leaf id {
            type uint16;
            description
            "The id of the element in service group.";
        }
    }
}

```

Internet-Draft

Security Policy Object Data Model

October 2018

```
    }
    leaf service-object-name {
        type service-set-name;
        mandatory true;
        description
            "The name of the service object that consists the service group.";
    }
        description
            "A list of service objects that consists the service group object.";
    }
        description
            "A list of service group objects.";
    }
description
    "A service group object is comprised of several service objects that requ
}

/*
* Groupings for application object and application group
*/
grouping application-objects {
    container user-defined-application {
        if-feature user-defined-application;
        container applications {
            list application {
                key "name";
                leaf name {
                    type string;
                    description
                        "The name of user-defined application object.";
                }
                leaf-list label {
                    type string;
                    description
                        "A list of labels for user-defined application.";
                }
                leaf data-model {
                    type string;
                    description
                        "The data transmission model of user-defined application. Example
                }
                leaf category {
```

```

    type string;
    description
        "The category of user-defined application. The value of this leaf
    }
    leaf subcategory {

```

```

    type string;
    description
        "The subcategory of user-defined application. ";
    }
    leaf risk-value {
        type uint32;
        config false;
        description
            "The risk value of predefined application.";
    }
    leaf desc {
        type string;
        description
            "The description information of user-defined application.";
    }
    list rule {
        key "name";
        leaf name {
            type string;
            description
                "The name of the user-defined application rule.";
        }
        leaf protocol {
            type protocol;
            description
                "The protocol that user-defined application is based on.";
        }
        container signature {
            leaf mode {
                type string;
                description
                    "The mode of keyword identification. If the keyword exists in
            }
            leaf direction {
                type direction;
                description

```

```

        "The traffic direction for application identification. Request
    }
    leaf pattern-type{
        type pattern-type;
        description
            "The match pattern of the user-defined application rule. If t
    }
    leaf pattern {
        type string;
        description
            "The keyword of user-defined application rule.";
    }

```

```

    leaf field {
        type identityref {
            base protocol-field;
        }
        default general-payload;
        description
            "The protocol field to search for a signature. The default pr
    }
    description
        "The signature/characteristics of user-defined application.";
    }
    description
        "The rule used to identify the user-defined application.";
    }
    leaf-list ip-address {
        type inet:ip-prefix;
        description
            "The destination IPv4/IPv6 address of user-defined application.";
    }
    leaf-list port {
        type inet:port-number;
        description
            "The destination port number of user-defined application.";
    }
    description
        "A list of user-defined application objects.";
    }
    description
        "When the NSF supports user-defined application function, these are a

```

```

    }
    description
        "When the NSF supports user-defined application function, this contains"
    }
    container predefined-application {
        config false;
        list application {
            key "name";
            leaf name {
                type string;
                config false;
                description
                    "The name of the predefined application.";
            }
            leaf-list protocol {
                type string;
                config false;
                description
                    "The protocol information of application.";
            }
        }
    }

```

```

    }
    leaf risk-value {
        type uint32;
        config false;
        description
            "The risk value of predefined application.";
    }
    leaf-list label {
        type string;
        config false;
        description
            "The label of predefined application,an application may have multiple labels";
    }
    leaf abandon {
        type boolean;
        config false;
        description
            "The abandon flag of predefined application.";
    }
    leaf multichannel {
        type boolean;
        config false;
    }

```

```

        description
            "The multi channel flag of predefined application.";
    }
    leaf data-model {
        type string;
        description
            "The data transmission model of user-defined application. Examples
    }
    leaf category {
        type string;
        config false;
        description
            "The category of user-defined application. The value of this leaf i
    }
    leaf subcategory {
        type string;
        config false;
        description
            "The name of application subcategory.";
    }
    leaf desc {
        type string;
        config false;
        description
            "The description information of application.";
    }
}

```

```

        description
            "The attributes of a predefined application.";
    }
    description
        "The information of all predefined applications.";
    }
    description
        "A list of predefined application objects.";
}

grouping application-groups {
    list application-group {
        key "name";
        leaf name {
            type string;

```

```

        description
            "The name of the application group.";
    }
    leaf desc {
        type string{
            length "1..127";
        }
        description
            "The description of the application group.";
    }
    list items {
        key "id";
        leaf id {
            type uint16;
            description
                "The id of the element in application group.";
        }
        leaf application-object-name {
            type string;
            mandatory true;
            description
                "The name of the application object that consists the application g
        }
        description
            "A list of application objects that consist an application group obje
    }
    description
        "A list of application group objects.";
}
description
    "An application group object is comprised of several application objects
}

```

```

/*
 * Groupings for user object, user group and security group
 */
grouping user-objects {
    list user-object {
        key "name aaa-domain";
        leaf name {
            type user-name;
        }
    }
}

```



```

        description
            "The name of the user.";
    }
    leaf aaa-domain {
        type string {
            length "1..64";
        }
        description
            "The name of the domain to which the user belong.";
    }
    leaf desc {
        type string {
            length "1..127";
        }
        description
            "The description of the user.";
    }
    leaf password {
        type ianach:crypt-hash;
        description
            "If user is authenticated locally on the NSF, this attribute is manda
    }
    leaf parent-user-group {
        type user-group-name;
        description
            "The name of the parent group. User objects and user groups are in a
    }
    leaf-list parent-security-group {
        type user-security-group-name;
        max-elements 40;
        description
            "The name of the parent security group. A user object can belong to s
    }
    container expiration-time {
        choice expiration-type {
            case never-expire {
                leaf never-expire {
                    type empty;
                    description
                        "This case indicates that the user never expire.";

```

}

```

    }
    case expire-after-this-time {
        leaf expiration-time {
            type yang:date-and-time;
            description
                "User expired time.";
        }
    }
    description
        "Two types of user expiration configurations.";
}
description
    "User expiration time.";
}
container ip-mac-binding {
    choice bind-state {
        case no-binding {
            leaf no-binding{
                type empty;
                mandatory true;
                description
                    "No binding: Indicates that a user is not bound to any IP or MA
            }
        }
        case binding {
            leaf bind-mode{
                type ip-mac-binding-type;
                description
                    "The user and IP/MAC address binding mode: bidirectional, or un
            }
            leaf-list ip-binding {
                type inet:ipv4-address;
                description
                    "The IP address bound to the user.";
            }
            leaf-list mac-binding {
                type yang:mac-address;
                description
                    "The MAC address bound to the user.";
            }
            list ip-mac-bindings {
                key "ip-binding";
                unique "mac-binding";
                leaf ip-binding {
                    type inet:ipv4-address;
                    description
                        "The bound IPv4 address";
                }
            }
        }
    }
}

```

```
    }
    leaf mac-binding {
      type yang:mac-address;
      description
        "The bound mac address";
    }
    description
      "Configure the IP address and MAC address pairs bound to the user."
  }
  }
  description
    "The binding state: no-binding, binding."
}
description
  "Whether there are IP/MAC addresses bound to the user."
}
description
  "User Object and its attributes."
}
description
  "A list of user objects."
}

grouping security-groups {
  list security-group {
    key "name";
    leaf name {
      type user-security-group-name;
      description
        "The name of the security-group."
    }
    leaf desc {
      type string {
        length "1..127";
      }
      description
        "The description of the security-group."
    }
  }
  leaf-list parent-security-group {
    type user-security-group-name;
    max-elements 40;
    description
      "Configure the name of the parent-security-group."
  }
  container filter-action {
    choice filter-type {
```

```
case static {
  leaf static {
```

```
    type empty;
    mandatory true;
    description
      "Empty leaf indicates that this is a static security group.";
  }
}
case dynamic {
  leaf dynamic {
    type empty;
    mandatory true;
    description
      "Empty leaf indicates that this is a dynamic security group.";
  }
  leaf-list filter-rule {
    type string {
      length "1..256";
    }
    max-elements 5;
    description
      "Filter rules for dynamic security group.";
  }
}
description
  "The filter type: static, dynamic.";
}
description
  "The filter type of the security group, static and dynamic. For dynamic
  security group, the filter type is dynamic."
}
description
  "Security group and its attributes.";
}
description
  "A list of security groups.";
}

grouping user-groups {
  list user-group {
    key "name";
    leaf name {
```

```

    type user-group-name;
    description
        "The name of the user group.";
}
leaf desc {
    type string {
        length "1..63";
    }
    description

```

```

        "The description of the user group.";
    }
    leaf parent-user-group {
        type user-group-name;
        description
            "The name of the user group. A user group can only belong to one parent user group.";
    }
    description
        "User group and its attributes.";
}
description
    "A list of user groups";
}

/*
 * Groupings for time range object
 */
grouping time-range-objects {
    list time-range-object {
        key "name";
        leaf name {
            type time-range-name;
            description
                "The name of the time range object.";
        }
    }
    list period-time {
        key "start end";
        leaf start {
            type hour-minute-second;
            mandatory true;
            description

```

```

        "Start time of the periodic time range.";
    }
    leaf end {
        type hour-minute-second;
        mandatory true;
        description
            "End time of the periodic time range.";
    }
    leaf-list weekday {
        type weekday;
        min-elements 1;
        max-elements 7;
        description
            "The weekday to which the periodic time range belongs.";
    }
    description

```

```

        "Periodic time that the associated function starts going into effect.
    }
    list absolute-time {
        key "start end";
        leaf start {
            type yang:date-and-time;
            description
                "Absolute start time and date";
        }
        leaf end {
            type yang:date-and-time;
            description
                "Absolute end time and date";
        }
        description
            "Absolute time and date that the associated function starts going into
    }
    description
        "The time range object and its attributes.";
    }
    description
        "A list of time range objects";
}

```

```

/*
 * Groupings for region object and region group
 */
grouping region-objects {
  list pre-defined-region {
    key "name";
    config false;
    leaf name {
      type region-name;
      config false;
      description
        "The name of the predefined region.";
    }
    leaf desc {
      type string;
      config false;
      description
        "The description of the predefined region.";
    }
  }
  container region-ipv4-address {
    leaf-list address-ipv4 {
      type inet:ipv4-prefix;
      config false;
    }
  }
}

```

```

    description
      "IPv4 address.";
  }
  list address-ipv4-range {
    key "start-ipv4 end-ipv4";
    leaf start-ipv4 {
      type inet:ipv4-address;
      config false;
      description
        "Start ipv4 address.";
    }
    leaf end-ipv4 {
      type inet:ipv4-address;
      config false;
      description
        "End ipv4 address.";
    }
  }
  description

```

```

        "A list of ipv4 address ranges";
    }
    description
        "The IPv4 addresses of the predefined region.";
}
container region-ipv6-address {
    if-feature support-ipv6-address;
    leaf-list address-ipv6 {
        type inet:ipv6-prefix;
        config false;
        description
            "IPv6 address.";
    }
    list address-ipv6-range {
        key "start-ipv6 end-ipv6";
        leaf start-ipv6 {
            type inet:ipv6-address;
            config false;
            description
                "Start ipv6 address.";
        }
        leaf end-ipv6 {
            type inet:ipv6-address;
            config false;
            description
                "End ipv6 address.";
        }
        description
            "A list of ipv6 address ranges";
    }
}

```

```

    description
        "The IPv6 addresses of the predefined region.";
    }
    description
        "A list of predefined region objects.";
}
list user-defined-region {
    key "name";
    leaf name {
        type region-name;
        description

```



```

        "The name of the user-defined region.";
    }
    leaf desc {
        type string;
        description
            "The description of the user-defined region.";
    }
    container coordinate {
        leaf longitude {
            type region-longitude;
            description
                "The latitude of the user-defined region.";
        }
        leaf latitude {
            type region-latitude;
            description
                "The longitude of the user-defined region.";
        }
        description
            "The latitude and longitude of the user-defined region.";
    }
    container region-ipv4-address {
        leaf-list address-ipv4 {
            type inet:ipv4-prefix;
            description
                "IPv4 address.";
        }
        list address-ipv4-range {
            key "start-ipv4 end-ipv4";
            leaf start-ipv4 {
                type inet:ipv4-address;
                description
                    "Start ipv4 address.";
            }
            leaf end-ipv4 {
                type inet:ipv4-address;
                description

```

```

        "End ipv4 address.";
    }
    description
        "A list of ipv4 address ranges";

```

```

    }
    description
        "The IPv4 addresses of the predefined region.";
}
container region-ipv6-address {
    if-feature support-ipv6-address;
    leaf-list address-ipv6 {
        type inet:ipv6-prefix;
        description
            "IPv6 address.";
    }
    list address-ipv6-range {
        key "start-ipv6 end-ipv6";
        leaf start-ipv6 {
            type inet:ipv6-address;
            description
                "Start ipv6 address.";
        }
        leaf end-ipv6 {
            type inet:ipv6-address;
            description
                "End ipv6 address.";
        }
        description
            "A list of ipv6 address ranges";
    }
    description
        "The IPv6 addresses of the user-defined region.";
}
description
    "A list of user-defined region objects.";
}
description
    "A list of predefined region objects and a list of user-defined region ob
}

grouping region-groups {
    list region-group {
        key "name";
        leaf name {
            type region-name;
            description
                "The name of the region group.";
        }
    }
}

```

```
    leaf desc {
      type string;
      description
        "The description of the region group.";
    }
    leaf-list region-name {
      type region-name;
      description
        "A list of region objects.";
    }
    leaf-list region-group-name {
      type region-name;
      description
        "A list of region groups.";
    }
    description
      "Region group consists of a set of region objects or region groups.";
  }
  description
    "A list of region group objects.";
}

/*
 * Groupings for domain object
 */
grouping domain-objects {
  list domain-object {
    key "name";
    leaf name {
      type domain-name;
      description
        "The name of the domain object.";
    }
    leaf desc {
      type string;
      description
        "The description of the domain object.";
    }
  }
  leaf-list domain {
    type string;
    description
      "A list of domains that consists the domain objects.";
  }
  description
    "Domain object and its attributes.";
}
description
```

"A list of domain objects.";

Internet-Draft

Security Policy Object Data Model

October 2018

```
}  
}
```

[7.](#) Acknowledgements

[8.](#) IANA Considerations

This document requires no IANA actions.

[9.](#) Security Considerations

Secure transport should be used to retrieve the current status of management plane security baseline.

[10.](#) References

[10.1.](#) Normative References

[I-D.ietf-i2nsf-capability]

Xia, L., Strassner, J., Basile, C., and D. Lopez,
"Information Model of NSFs Capabilities", [draft-ietf-i2nsf-capability-02](#) (work in progress), July 2018.

[I-D.ietf-i2nsf-terminology]

Hares, S., Strassner, J., Lopez, D., Xia, L., and H.
Birkholz, "Interface to Network Security Functions (I2NSF)
Terminology", [draft-ietf-i2nsf-terminology-06](#) (work in
progress), July 2018.

[I-D.ietf-netmod-acl-model]

Jethanandani, M., Agarwal, S., Huang, L., and D. Blair,
"Network Access Control List (ACL) YANG Data Model",
[draft-ietf-netmod-acl-model-20](#) (work in progress), October
2018.

[RFC8329]

Lopez, D., Lopez, E., Dunbar, L., Strassner, J., and R.
Kumar, "Framework for Interface to Network Security
Functions", [RFC 8329](#), DOI 10.17487/RFC8329, February 2018,
<<https://www.rfc-editor.org/info/rfc8329>>.

10.2. Informative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

Xia & Lin

Expires April 24, 2019

[Page 46]

Internet-Draft

Security Policy Object Data Model

October 2018

- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", [RFC 7950](#), DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", [BCP 215](#), [RFC 8340](#), DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.

Authors' Addresses

Liang Xia
Huawei
101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu 210012
China

Email: Frank.xialiang@huawei.com

Qiushi Lin
Huawei
Huawei Industrial Base
Shenzhen, Guangdong 518129
China

Email: linqiushi@huawei.com

