I2NSF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

J. Yang
J. Jeong
J. Kim
Sungkyunkwan University

March 11, 2019

Security Policy Translation in Interface to Network Security Functions
draft-yang-i2nsf-security-policy-translation-03

Abstract

This document proposes a scheme of security policy translation (i.e., Security Policy Translator) in Interface to Network Security Functions (I2NSF) Framework. When I2NSF User delivers a high-level security policy for a security service, Security Policy Translator in Security Controller translates it into a low-level security policy for Network Security Functions (NSFs).

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 September 12, 2019.

Table of Contents

1. Introduction

This document defines a scheme of a security policy translation in Interface to Network Security Functions (I2NSF) Framework [RFC8329]. First of all, this document explains the necessity of a security policy translator (shortly called policy translator) in the I2NSF framework.

The policy translator resides in Security Controller in the I2NSF framework and translates a high-level security policy to a low-level security policy for Network Security Functions (NSFs). A high-level policy is specified by I2NSF User in the I2NSF framework and is

delivered to Security Controller via Consumer-Facing Interface [consumer-facing-inf-dm]. It is translated into a low-level policy by Policy Translator in Security Controller and is delivered to NSFs to execute the rules corresponding to the low-level policy via NSF-Facing Interface [nsf-facing-inf-dm].

2. Terminology

This document uses the terminology specified in [i2nsf-terminology] [RFC8329].

3. Necessity for Policy Translator

Security Controller acts as a coordinator between I2NSF User and NSFs. Also, Security Controller has capability information of NSFs that are registered via Registration Interface [registration-inf-dm] by Developer's Management System [RFC8329]. As a coordinator, Security Controller needs to generate a low-level policy in the form of security rules intended by the high-level policy, which can be understood by the corresponding NSFs.

A high-level security policy is specified by RESTCONF/YANG [RFC8040][RFC6020], and a low-level security policy is specified by NETCONF/YANG [RFC6241][RFC6020]. The translation from a high-level security policy to the corresponding low-level security policy will be able to rapidly elevate I2NSF in real-world deployment. A rule in a high-level policy can include a broad target object, such as employees in a company for a security service (e.g., firewall and web filter). Such employees may be from human resource (HR) department, software engineering department, and advertisement department. A keyword of employee needs to be mapped to these employees from various departments. This mapping needs to be handled by a policy translator in a flexible way while understanding the intention of a policy specification. Let us consider the following two policies:

- Block my son's computers from malicious websites.
- Drop packets from the IP address 10.0.0.1 and 10.0.0.3 to harm.com and illegal.com

The above two sentences are examples of policies for blocking malicious websites. Both policies are for the same operation. However, NSF cannot understand the first policy, because the policy does not have any specified information for NSF. To set up the policy at an NSF, the NSF MUST receive at least the source IP address and website address for an operation. It means that the first sentence is NOT compatible for an NSF policy. Conversely, when I2NSF Users request a security policy to the system, they never make a

security policy like the second example. For generating a security policy like the second sentence, the user MUST know that the NSF needs to receive the specified information, source IP address and website address. It means that the user understands the NSF professionally, but there are not many professional users in a small size of company or at a residential area. In conclusion, the I2NSF User prefers to issue a security policy in the first sentence, but an NSF will require the same policy as the second sentence with specific information. Therefore, an advanced translation scheme of security policy is REQUIRED in I2NSF.

This document proposes an approach using Automata theory [Automata] for the policy tranlation, such as Deterministic Finite Automaton (DFA) and Context Free Grammar (CFG). Note that Automata theory is the foundation of programming language and compiler. Thus, with this approach, I2NSF User can easily specify a high-level security policy that will be enforced into the corresponding NSFs with a compatibly low-level security policy with the help of Policy Translator. Also, for easy management, a modularized translator structure is proposed.

4. Design of Policy Translator

Commonly used security policies are created as XML(Extensible Markup Language) [XML] files. A popular way to change the format of an XML file is to use an XSLT (Extensible Stylesheet Language Transformation) [XSLT] document. XSLT is an XML-based language to transform an input XML file into another output XML file. However, the use of XSLT makes it difficult to manage the policy translator and to handle the registration of new capabilities of NSFs. With the necessity for a policy translator, this document describes a policy translator based on Automata theory.

4.1. Overall Structure of Policy Translator

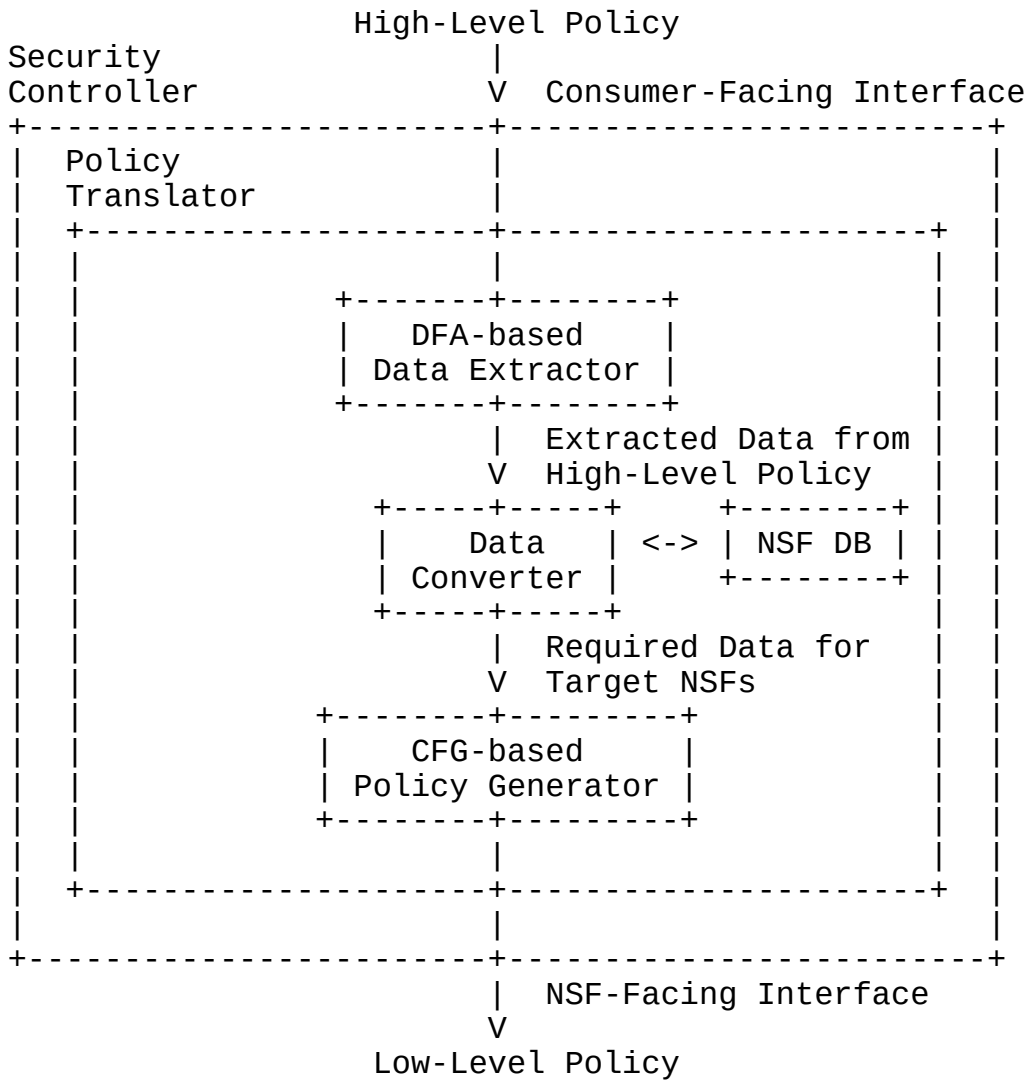

Figure 1: The Overall Design of Policy Translator

Figure 1 shows the overall design for Policy Translator in Security Controller. There are three main components for Policy Translator: Data Extractor, Data Converter, and Policy Generator.

Extractor is a DFA-based module for extracting data from a high-level policy which I2NSF User delivered via Consumer-Facing Interface. Data Converter converts the extracted data to the capabilities of target NSFs for a low-level policy. It refers to NSF Database (DB) in order to convert an abstract subject or object into the corresponding concrete subject or object (e.g., IP address and website URL). Policy Generator generates a low-level policy which will execute the NSF capabilities from Converter.

4.2. DFA-based Data Extractor

4.2.1. Design of DFA-based Data Extractor

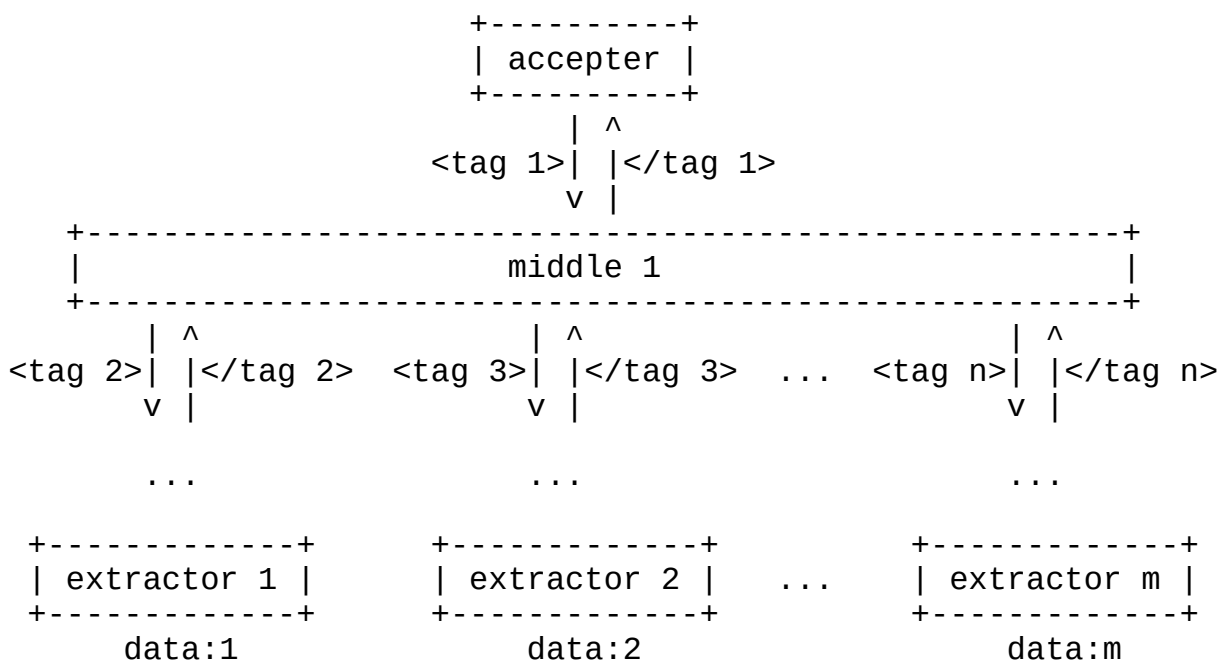

Figure 2: DFA Architecture of Data Extractor

Figure 2 shows a design for Data Extractor in the policy translator. If a high-level policy contains data along the hierarchical structure of the standard Consumer-Facing Interface YANG data model [consumer-facing-inf-dm], data can be easily extracted using the state transition machine, such as DFA. The extracted data can be processed and used by an NSF to understand it. Extractor can be constructed by designing a DFA with the same hierarchical structure as a YANG data model.

After constructing a DFA, Data Extractor can extract all of data in the enterred high-level policy by using state transitions. Also, the DFA can easily detect the grammar errors of the high-level policy. The extracting algorithm of Data Extractor is as follows:

1. Start from the ’accepter’ state.

2. Read the next tag from the high-level policy.

3. Transit to the corresponding state.

4. If the current state is in ’extractor’, extract the corresponding data, and then go back to step 2.

5. If the current state is in ’middle’, go back to step 2.

6. If there is no possible transition and arrived at ’accepter’ state, the policy has no grammar error. Otherwise, there is a grammar error, so stop the process with failure.

4.2.2. Example Scenario for Data Extractor

```
<I2NSF>
    <name>block_web</name>
    <cond>
        <src>Son’s_PC</src>
        <dest>malicious_websites</dest>
    </cond>
    <action>block<action>
</I2NSF>
```

Figure 3: The Example of High-level Policy

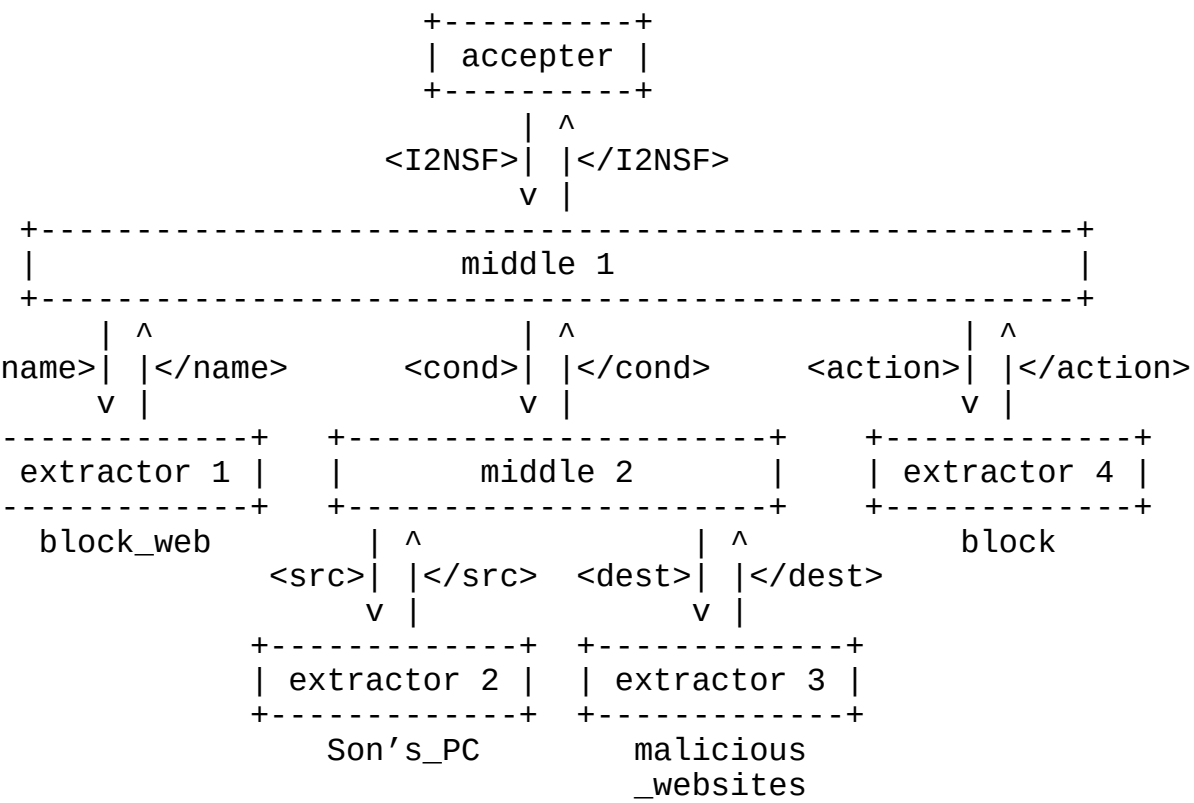

Figure 4: The Example of Data Extractor

To explain the Data Extractor process by referring to an example scenario, assume that Security Controller received a high-level

policy for a web-filtering as shown in Figure 3. Then we can construct DFA-based Data Extractor by using the design as shown in Figure 2. Figure 4 shows the architecture of Data Extractor that is based on the architection in Figure 2 along with the input high-level policy in Figure 3. Data Extractor can automatically extract all of data in the high-level policy according to the following process:

1. Start from the 'accepter' state.

2. Read the first opening tag called '<I2NSF>', and transit to the 'middle 1' state.

3. Read the second opening tag called '<name>', and transit to the 'extractor 1' state.

4. The current state is an 'extractor' state. Extract the data of 'name' field called 'block_web'.

5. Read the second closing tag called '</name>', and go back to the 'middle 1' state.

6. Read the third opening tag called '<cond>', and transit to the 'middle 2' state.

7. Read the fourth opening tag called '<src>', and transit to the 'extractor 2' state.

8. The current state is an 'extractor' state. Extract the data of 'src' field called 'Son's_PC'.

9. Read the fourth closing tag called '</src>', and go back to the 'middle 2' state.

10. Read the fifth opening tag called '<dest>', and transit to the 'extractor 3' state.

11. The current state is an 'extractor' state. Extract the data of 'dest' field called 'malicious_websites'.

12. Read the fifth closing tag called '</dest>', and go back to the 'middle 2' state.

13. Read the third closing tag called '</cond>', and go back to the 'middle 1' state.

14. Read the sixth opening tag called '<action>', and transit to the 'extractor 4' state.

15. The current state is an ’extractor’ state. Extract the data of ’action’ field called ’block’.

16. Read the sixth closing tag called ’</action>’, and go back to the ’middle 1’ state.

17. Read the first closing tag called ’</I2NSF>’, and go back to the ’accepter’ state.

18. There is no further possible transition, and the state is finally on ’accepter’ state. There is no grammar error in Figure 3 so the scanning for data extraction is finished.

The above process is constructed by an extracting algorithm. After finishing all the steps of the above process, Data Extractor can extract all of data in Figure 3, ’block_web’, ’Son’s_PC’, ’malicious’, and ’block’.

Since the translator is modularized into a DFA structure, a visual understanding is feasible. Also, The performance of Data Extractor is excellent compared to one-to-one searching of data for a particular field. In addition, the management is efficient because the DFA completely follows the hierarchy of Consumer-Facing Interface. If I2NSF User wants to modify the data model of a high-level policy, it only needs to change the connection of the relevant DFA node.

4.3. Data Converter

4.3.1. Role of Data Converter

Every NSF has its own unique capabilities. The capabilities of an NSF are registered into Security Controller by a Developer’s Management System, which manages the NSF, via Registration Interface. Therefore, Security Controller already has all information about the capabilities of NSFs. This means that Security Controller can find target NSFs with only the data (e.g., subject and object for a security policy) of the high-level policy by comparing the extracted data with all capabilities of each NSF. This search process for appropriate NSFs is called by policy provisioning, and it eliminates the need for I2NSF User to specify the target NSFs explicitly in a high-level security policy.

Data Converter selects target NSFs and converts the extracted data into the capabilities of selected NSFs. If Security Controller uses this data convertor, it can provide the policy provisioning function to I2NSF User automatically. Thus, the translator design provides big benefits to the I2NSF Framework.

4.3.2. NSF Database

The NSF Database contains all the information needed to convert high-level policy data to low-level policy data. The contents of NSF Database are classified as the following two: "endpoint information" and "NSF capability information".

The first is "endpoint information". Endpoint information is necessary to convert an abstract high-level policy data such as Son's_PC, malicious to a specific low-level policy data such as 10.0.0.1, illegal.com. In the high-level policy, the range of endpoints for applying security policy MUST be provided abstractly. Thus, endpoint information is needed to specify the abstracted high-level policy data. Endpoint information is provided by I2NSF User as the high-level policy through Consumer-Facing Interface, and Security Controller builds NSF Database based on received information.

The second is "NSF capability information". Since capability is information that allows NSF to know what features it can support, NSF capability information is used in policy provisioning process to search the appropriate NSFs through the security policy. NSF capability information is provided by Developer's Management System (DMS) through Registration Interface, and Security Controller builds NSF Database based on received information. In addition, if the NSF sends monitoring information such as initiating information to Security Controller through NSF-Facing Interface, Security Controller can modify NSF Database accordingly.

4.3.3. Data Conversion in Data Converter

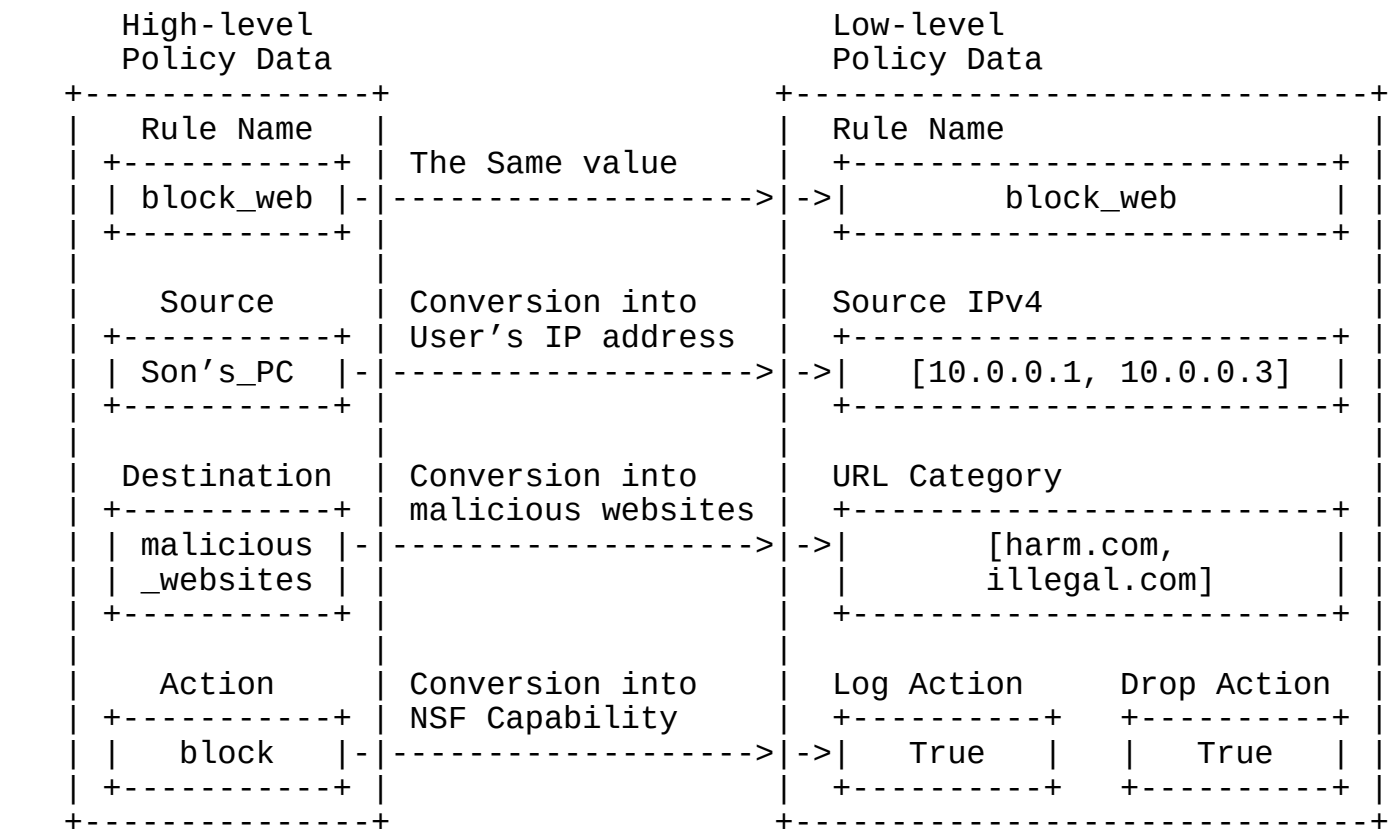

Figure 5: Example of Data Conversion

Figure 5 shows an example for describing a data conversion in Data Converter. High-level policy data MUST be converted into low-level policy data which are compatible with NSFs. If a ystem administrator attaches a database to Data Converter, it can convert contents by referring to the database with SQL queries. Data conversion in Figure 5 is based on the following list:

- 'Rule Name' field does NOT need the conversion.
- 'Source' field SHOULD be converted into a list of target IPv4 addresses.
- 'Destination' field SHOULD be converted into a URL category list of malicious websites.
- 'Action' field SHOULD be converted into the corresponding action(s) in NSF capabilities.

4.3.4. Policy Provisioning

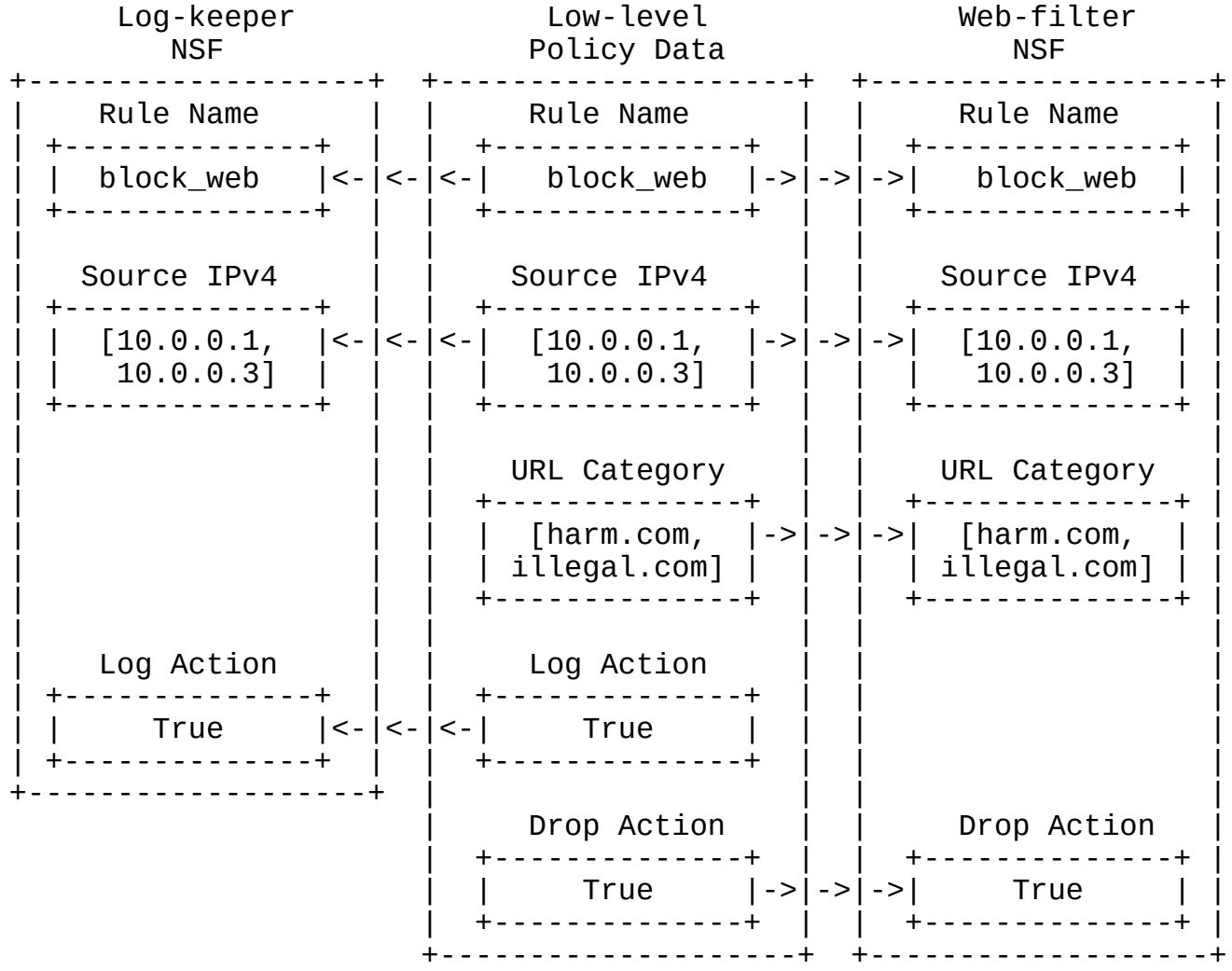

Figure 6: Example of Policy Provisioning

Generator searches for proper NSFs which can cover all of capabilities in the high-level policy. Generator searches for target NSFs by comparing only NSF capabilities which is registered by Vendor Management System. This process is called by "policy provisioning" because Generator finds proper NSFs by using only the policy. If target NSFs are found by using other data which is not included in a user's policy, it means that the user already knows the specific knowledge of an NSF in the I2NSF Framework. Figure 6 shows an example of policy provisioning. In this example, log-keeper NSF and web-filter NSF are selected for covering capabilities in the security policy. All of capabilities can be covered by two selected NSFs.

4.4. CFG-based Policy Generator

Generator makes low-level security policies for each target NSF with the extracted data. We constructed Generator by using Context Free Grammar (CFG). CFG is a set of production rules which can describe all possible strings in a given formal language(e.g., programming language). The low-level policy also has its own language based on a YANG data model of NSF-Facing Interface. Thus, we can construct the productions based on the YANG data model. The productions that makes up the low-level security policy are categorized into two types, 'Content Production' and 'Structure Production'.

4.4.1. Content Production

Content Production is for injecting data into low-level policies to be generated. A security manager(i.e., a person (or software) to make productions for security policies) can construct Content Productions in the form of an expression as the following productions:

- [cont_prod] -> [cont_prod][cont_prod] (Where duplication is allowed.)

- [cont_prod] -> <cont_tag>[cont_data]</cont_tag>

- [cont_data] -> data_1 | data_2 | ... | data_n

Square brackets mean non-terminal state. If there are no non-terminal states, it means that the string is completely generated. When the duplication of content tag is allowed, the security manager adds the first production for a rule. If there is no need to allow duplication, the first production can be skipped because it is an optional production.

The second production is the main production for Content Production because it generates the tag which contains data for low-level policy. Last, the third production is for injecting data into a tag which is generated by the second production. If data is changed for an NSF, the security manager needs to change "only the third production" for data mapping in each NSF.

For example, if the security manager wants to express a low-level policy for source IP address, Content Production can be constructed in the following productions:

- [cont_ipv4] -> [cont_ipv4][cont_ipv4] (Allow duplication.)

- [cont_ipv4] -> <ipv4>[cont_ipv4_data]</ipv4>

- [cont_ipv4_data] -> 10.0.0.1 | 10.0.0.3

4.4.2. Structure Production

Structure Production is for grouping other tags into a hierarchy. The security manager can construct Structure Production in the form of an expression as the following production:

- [struct_prod] -> <struct_tag>[prod_1]...[prod_n]</struct_tag>

Structure Production can be expressed as a single production. The above production means to group other tags by the name of a tag which is called by 'struct_tag'. [prod_x] is a state for generating a tag which wants to be grouped by Structure Production. [prod_x] can be both Content Production and Structure Production. For example, if the security manager wants to express the low-level policy for the I2NSF tag, which is grouping 'name' and 'rules', Structure Production can be constructed as the following production where [cont_name] is the state for Content Production and [struct_rule] is the state for Structure Production.

- [struct_i2nsf] -> <I2NSF>[cont_name][struct_rules]</I2NSF>

4.4.3. Generator Construction

The security manager can build a generator by combining the two productions which are described in Section 4.4.1 and Section 4.4.2. Figure 7 shows the CFG-based Generator construction of the web-filter NSF. It is constructed based on the NSF-Facing Interface Data Model in [nsf-facing-inf-dm]. According to Figure 7, the security manager can express productions for each clause as in following CFG:

1. [cont_name] -> <rule-name>[cont_name_data]</rule-name>

2. [cont_name_data] -> block_web

3. [cont_ipv4] -> [cont_ipv4][cont_ipv4] (Allow duplication)

4. [cont_ipv4] -> <ipv4>[cont_ipv4_data]</ipv4>

5. [cont_ipv4_data] -> 10.0.0.1 | 10.0.0.3

6. [cont_url] -> [cont_url][cont_url] (Allow duplication)

7. [cont_url] -> <url>[cont_url_data]</url>

8. [cont_url_data] -> harm.com | illegal.com

9. [cont_action] -> <action>[cont_action_data]</action>

10. [cont_action_data] -> drop

11. [struct_packet] -> <packet>[cont_ipv4]</packet>

12. [struct_payload] -> <payload>[cont_url]</payload>

13. [struct_cond] -> <condition>[struct_packet][struct_payload]</condition>

14. [struct_rules] -> <rules>[struct_cond][cont_action]</rules>

15. [struct_i2nsf] -> <I2NSF>[cont_name][struct_rules]</I2NSF>

Then, Generator generates a low-level policy by using the above CFG. The low-level policy is generated by the following process:

1. Start: [struct_i2nsf]

2. Production 15: <I2NSF>[cont_name][struct_rules]</I2NSF>

3. Production 1: <I2NSF><rule-name>[cont_name_data]</rule-name>[struct_rules]</I2NSF>

4. Production 2: <I2NSF><rule-name>block_web</rule-name>[struct_rules]</I2NSF>

5. Production 14: <I2NSF><rule-name>block_web</rule-name><rules>[struct_cond][cont_action]</rules></I2NSF>

6. Production 13: <I2NSF><rule-name>block_web</rule-name><rules><condition>[struct_packet][struct_payload]</condition>[cont_action]</rules></I2NSF>

7. Production 11: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet>[cont_ipv4]</packet>[struct_payload]</condition>[cont_action]</rules></I2NSF>

8. Production 3: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet>[cont_ipv4][cont_ipv4]</packet>[struct_payload]</condition>[cont_action]</rules></I2NSF>

9. Production 4: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet><ipv4>[cont_ipv4_data]</ipv4><ipv4>[cont_ipv4_data]</ipv4></packet>[struct_payload]</condition>[cont_action]</rules></I2NSF>

10. Production 5: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet><ipv4>10.0.0.1</ipv4><ipv4>10.0.0.3</ipv4></packet>[struct_payload]</condition>[cont_action]</rules></I2NSF>

11. Production 12: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet><ipv4>10.0.0.1</ipv4><ipv4>10.0.0.3</ipv4></packet><payload>[cont_url]</payload></condition>[cont_action]</rules></I2NSF>

12. Production 6: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet><ipv4>10.0.0.1</ipv4><ipv4>10.0.0.3</ipv4></packet><payload>[cont_url][cont_url]</payload></condition>[cont_action]</rules></I2NSF>

13. Production 7: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet><ipv4>10.0.0.1</ipv4><ipv4>10.0.0.3</ipv4></packet><payload><url>[cont_url_data]</url><url>[cont_url_data]</url></payload></condition>[cont_action]</rules></I2NSF>

14. Production 8: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet><ipv4>10.0.0.1</ipv4><ipv4>10.0.0.3</ipv4></packet><payload><url>harm.com</url><url>illegal.com</url></payload></condition>[cont_action]</rules></I2NSF>

15. Production 9: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet><ipv4>10.0.0.1</ipv4><ipv4>10.0.0.3</ipv4></packet><payload><url>harm.com</url><url>illegal.com</url></payload></condition><action>[cont_action_data]</action></rules></I2NSF>

16. Production 10: <I2NSF><rule-name>block_web</rule-name><rules><condition><packet><ipv4>10.0.0.1</ipv4><ipv4>10.0.0.3</ipv4></packet><payload><url>harm.com</url><url>illegal.com</url></payload></condition><action>drop</action></rules></I2NSF>

The last production has no non-terminal state, and the low-level policy is completely generated. Figure 8 shows the generated low-level policy where tab characters and newline characters are added.

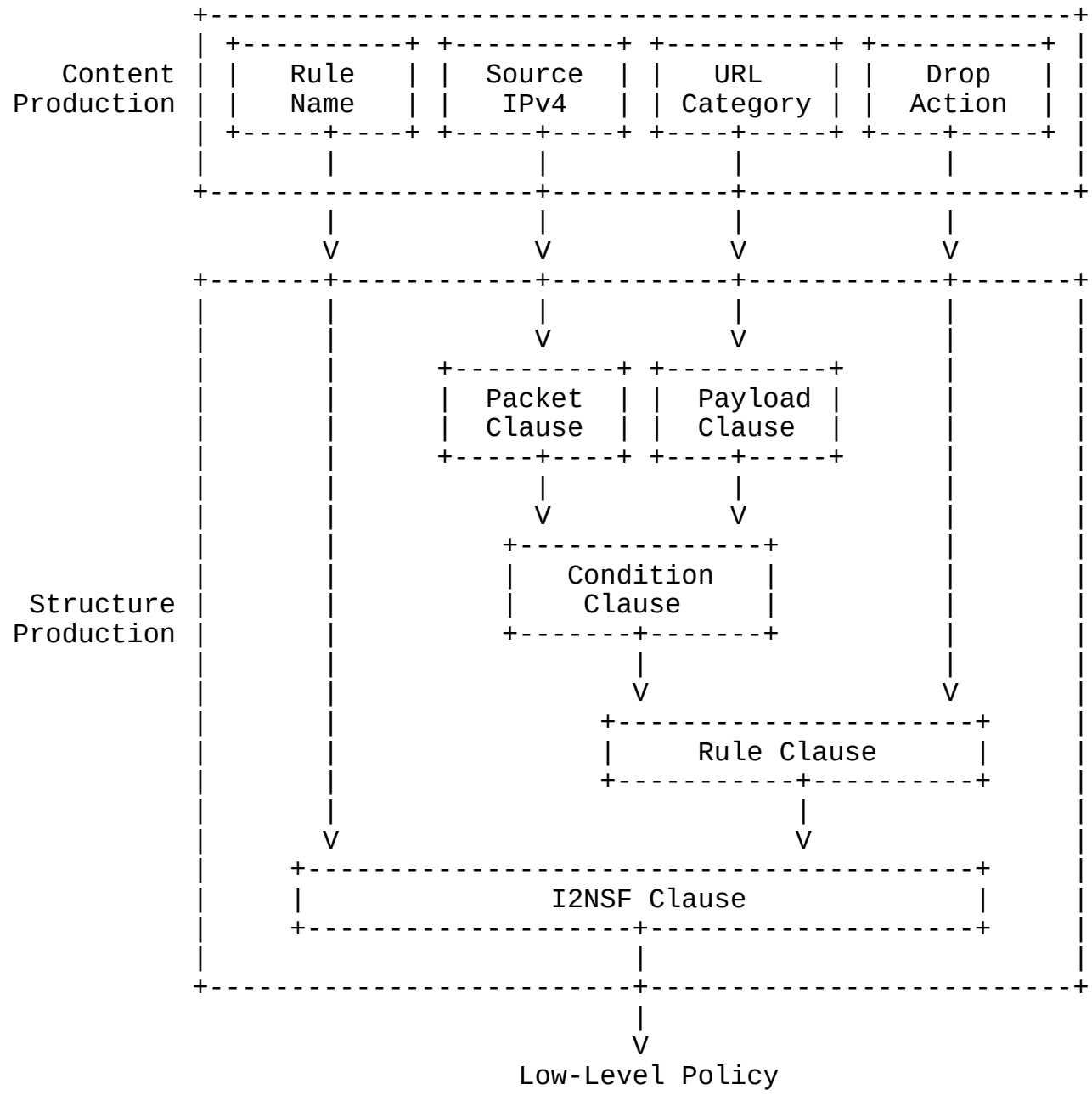

Figure 7: Generator Construction for Web-Filter NSF

```
<I2NSF>
    <rule-name>block_web</rule-name>
    <rules>
        <condition>
            <packet>
                <ipv4>10.0.0.1</ipv4>
                <ipv4>10.0.0.3</ipv4>
            </packet>
            <payload>
                <url>harm.com</url>
                <url>illegal.com</url>
            </payload>
        </condition>
        <action>drop</action>
    </rules>
</I2NSF>
```

Figure 8: Example of Low-Level Policy

5. Implementation Considerations

The implementation considerations in this document include the following three: "data model auto-adaptation", "data conversion", and "policy provisioning".

5.1. Data Model Auto-adaptation

Security Controller which acts as the intermediary MUST process the data according to the data model of the connected interfaces. However, the data model can be changed flexibly depending on the situation, and Security Controller may adapt to the change. Therefore, Security Controller can be implemented for convenience so that the security policy translator can easily adapt to the change of the data model.

The translator constructs and uses the DFA to adapt to Consumer-Facing Interface Data Model. In addition, the CFG is constructed and used to adapt to NSF-Facing Interface Data Model. Both the DFA and the CFG follow the same tree structure of YANG Data Model.

The DFA starts at the a node and expands operations by changing the state according to the input. Based on the YANG Data Model, a container node is defined as a middle state and a leaf node is defined as an extractor node. After that, if the nodes are connected in the same way as the hierarchical structure of the data model, Security Controller can automatically construct the DFA. The DFA can be conveniently built by investigating the link structure using the stack, starting with the root node.

The CFG starts at the leaf nodes and is grouped into clauses until all the nodes are merged into one node. A leaf node is defined as the content production, and a container node is defined as the structure production. After that, if the nodes are connected in the same way as the hierarchy of the data model, Security Controller can automatically construct the CFG. The CFG can be conveniently constructed by investigating the link structure using the priority queue data, starting with the leaf nodes.

5.2. Data Conversion

Security Controller requires the ability to materialize the abstract data in the high-level security policy and forward it to NSFs. Security Controller can receive endpoint information as keywords through the high-level security policy. At this time, if the endpoint information corresponding to the keyword is mapped and the query is transmitted to the NSF Database, the NSF Database can be conveniently registered with necessary information for data conversion. When a policy tries to establish a policy through the keyword, Security Controller searches the details corresponding to the keyword registered in the NSF Database and converts the keywords into the appropriate and specified data.

5.3. Policy Provisioning

This document stated that policy provisioning function is necessary to enable users without expert security knowledge to create policies. Policy provisioning is determined by the capability of the NSF. If the NSF has information about the capability in the policy, the probability of selection increases.

Most importantly, selected NSFs may be able to performe all capabilities in the security policy. This document recommends a study of policy provisioning algorithms that are highly efficient and can satisfy all capabilities in the security policy.

6. Features of Policy Translator Design

First, by showing a visualized translator structure, the security manager can handle various policy changes. Translator can be shown by visualizing DFA and Context-free Grammar so that the manager can easily understand the structure of Policy Translator.

Second, if I2NSF User only keeps the hierarchy of the data model, I2NSF User can freely create high-level policies. In the case of DFA, data extraction can be performed in the same way even if the order of input is changed. The design of the policy translator is

more flexible than the existing method that works by keeping the tag ’s position and order exactly.

Third, the structure of Policy Translator can be updated even while Policy Translator is operating. Because Policy Translator is modularized, the translator can adapt to changes in the NSF capability while the I2NSF framework is running. The function of changing the translator’s structure can be provided through Registration Interface.

7. Security Considerations

There is no security concern in the proposed security policy translator as long as the I2NSF interfaces (i.e., Consumer-Facing Interface, NSF-Facing Interface, and Registration Interface) are protected by secure communication channels.

8. Acknowledgments

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea MSIT (Ministry of Science and ICT) (R-20160222-002755, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning).

This work was supported in part by the MSIT under the ITRC (Information Technology Research Center) support program (IITP-2018-2017-0-01633) supervised by the IITP.

9. References

9.1. Normative References

[Automata] Peter, L., "Formal Languages and Automata, 6th Edition", January 2016.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, January 2017.

[RFC8329] Lopez, D., Lopez, E., Dunbar, L., Strassner, J., and R. Kumar, "Framework for Interface to Network Security Functions", RFC 8329, February 2018.

[XML] W3C, "On Views and XML (Extensible Markup Language)", June 1999.

9.2. Informative References

[consumer-facing-inf-dm]
Jeong, J., Kim, E., Ahn, T., Kumar, R., and S. Hares, "I2NSF Consumer-Facing Interface YANG Data Model", draft-ietf-i2nsf-consumer-facing-interface-dm-03 (work in progress), March 2019.

[i2nsf-terminology]
Hares, S., Strassner, J., Lopez, D., Xia, L., and H. Birkholz, "Interface to Network Security Functions (I2NSF) Terminology", draft-ietf-i2nsf-terminology-07 (work in progress), July 2019.

[nsf-facing-inf-dm]
Kim, J., Jeong, J., Park, J., Hares, S., and Q. Lin, "I2NSF Network Security Function-Facing Interface YANG Data Model", draft-ietf-i2nsf-nsf-facing-interface-dm-03 (work in progress), March 2019.

[registration-inf-dm]
Hyun, S., Jeong, J., Roh, T., Wi, S., and J. Park, "I2NSF Registration Interface YANG Data Model", draft-ietf-i2nsf-registration-interface-dm-02 (work in progress), March 2019.

[XSLT] W3C, "Extensible Stylesheet Language Transformations (XSLT) Version 1.0", November 1999.

Appendix A. Changes from draft-yang-i2nsf-security-policy-translation-02

The following changes are made from draft-yang-i2nsf-security-policy-translation-02:

- Section 4.3.2 is added for describing 'NSF Database'. This section reinforces the ambiguous description of the NSF Database.
- Section 5 is added for describing 'Implementation Considerations'. This section provides guidelines for a convenient implementation of security policy translator.

Authors' Addresses

Jinhyuk Yang
Department of Computer Engineering
Sungkyunkwan University
2066 Seobu-Ro, Jangan-Gu
Suwon, Gyeonggi-Do 16419
Republic of Korea

Phone: +82 10 8520 8039
EMail: jin.hyuk@skku.edu

Jaehoon Paul Jeong
Department of Software
Sungkyunkwan University
2066 Seobu-Ro, Jangan-Gu
Suwon, Gyeonggi-Do 16419
Republic of Korea

Phone: +82 31 299 4957
Fax: +82 31 290 7996
EMail: pauljeong@skku.edu
URI: http://iotlab.skku.edu/people-jaehoon-jeong.php

Jinyong Tim Kim
Department of Computer Engineering
Sungkyunkwan University
2066 Seobu-Ro, Jangan-Gu
Suwon, Gyeonggi-Do 16419
Republic of Korea

Phone: +82 10 8273 0930
EMail: timkim@skku.edu