Compiling Natural Language expressions to extended BPF programs for stateful Network Policy Enforcement

Mohammad Firas Sada, Nik Sultana
Department of Computer Science, Illinois Institute of Technology

Abstract

In this research, we address the challenge of configuring and securing distributed systems using natural language. This is important because it allows for easier and more intuitive configuration of network packets, which can improve security and efficiency in distributed systems. To address this problem, we propose a transpiler written in Java that converts natural language input (NRL) into BPF rules that can be applied to network packets. The transpiler includes a parser and code generator, and can handle stateful rules and manual packet manipulation using BPF. It has been evaluated using multiple PCAP files and NRL rule files, and a toolchain in C was used to apply the generated BPF rules. The transpiler provides a solution for configuring and securing distributed systems using natural language, and has been thoroughly tested to ensure its effectiveness.

1 Introduction

Networked systems require careful configuration to function and be protected against malicious attacks. However, traditional methods of specifying rules for these systems often involve low-level programming languages or complex configuration files, which can be difficult for non-technical users to understand, and lead to mistakes being made.

This research develops a transpiler that allows users to specify rules for distributed systems using a natural language called NRL (natural rule language). The transpiler takes NRL as input and generates BPF (Berkeley Packet Filter) programs, which can be directly applied to network packets. The BPF output is also consistent with any packet definition provided by a DFDL (Data Format Description Language) schema.

Using NRL to specify rules has several benefits. First, it allows non-technical users to easily understand and specify rules for distributed systems. This is particularly useful for organizations that have a diverse set of users with varying technical expertise, as it enables all users to contribute to the configuration and security of the system. Second, it reduces the risk of errors introduced by manual configuration or programming. Third, it enables the reuse of rules across different systems, as the transpiler can generate BPF rules for any system that uses BPF and a DFDL schema.

In addition to its technical benefits, this transpiler also has the potential to improve the efficiency and productivity of organizations that use distributed systems. By enabling non-technical users to easily specify rules, it reduces the reliance on technical staff for configuration and security tasks, freeing them up to focus on more complex tasks.

Overall, this transpiler allows for the easy configuration and securing of distributed systems using natural language, improving the usability, reliability, and efficiency of these systems for both technical and non-technical users.

2 Example

An example of transpiling NRL rules to BPF might involve the following steps: The user writes a set of rules in NRL, using natural language statements to describe the desired behavior of the networked system. For example, the user might write rules that identify TCP flows into BPF involves converting natural language statements about TCP flows into low-level programming language code that can be applied to network packets. For example, consider the following NRL rule:

```
If Packet.TCPFlow then IPSrc
must be '192.168.1.1'
```

To transpile this rule into BPF, the transpiler would need to parse the NRL rule and generate code that checks the protocol and destination port of each packet, and then determines whether it is part of an established flow or a new connection attempt.

One way to do this would be to track the state of each TCP flow using information such as the source and destination IP addresses and ports, as well as the sequence numbers and acknowledgement numbers of the packets. Based on this information, the transpiler could generate BPF code similar to the following (truncated):

```
rule111: ldh [12]
rule112: jeq #2048 , rule113 , rule116
rule113: ldb [23]
```

```
rule114: jeq #6 , rule115 , keep
rule115: st M[0]
.
.
```

This code checks the protocol and destination port of each packet, and then uses the flow state to determine whether to allow or drop the packet. The flow state could be set using additional BPF code that tracks the state of each TCP flow based on the packet information. By transpiling NRL rules that identify TCP flows into BPF, it is possible to implement complex behavior for handling packets within TCP flows, while still allowing non-technical users to specify the rules in a natural language format.

3 Design

Our transpiler is designed to take NRL as input and generate BPF rules as output for configuring and securing distributed systems. To ensure that the BPF output is consistent with any packet definition provided by a DFDL schema, the transpiler includes its own XML parsing component for parsing the DFDL schema. This component generates a set of templates that the code generator uses to generate the BPF code.

The transpiler also allows for manual writing into packets using BPF, enabling users to specify custom rules for packet manipulation. In addition, the transpiler supports stateful rules that can identify TCP flows and target packets in established flows, enabling more fine-grained control over network traffic.

Overall, the design of our transpiler allows for the easy specification of rules using NRL, the generation of efficient and accurate BPF code for distributed systems, and the ability to customize packet manipulation and apply stateful rules.

$$\begin{tikzcd}
\text{NRL Rules} \arrow[dr] & & \\
& \text{Transpiler} \arrow[r] & \text{BPF Output} \\
\text{DFDL Schema} \arrow[ur] & &
\end{tikzcd}$$

4 Evaluation

To evaluate the effectiveness and efficiency of our transpiler, we conducted a series of experiments using multiple PCAP files and rule files written in NRL. One rule file included 136 various NRL rules between statefule and non-stateful, targeting various packet fields and configurations, as well as the packet's status in a TCP flow.

To test the transpiler, we wrote a toolchain in C that applied the outputted BPF rules to the PCAP files that include 10,000+ packets. This allowed us to compare the performance of the BPF rules generated by the transpiler to those written manually in C.

Our results show that the BPF rules generated by the transpiler have a minimal impact on performance and provide equivalent security to manually written rules. This demonstrates the effectiveness and efficiency of our transpiler in generating accurate and efficient BPF rules for configuring and securing distributed systems.

In addition to testing the performance of the BPF rules, we also evaluated the usability of the transpiler by asking non-technical users to specify rules using NRL. The results showed that the non-technical users were able to easily understand and specify rules using NRL, demonstrating the utility of our transpiler in enabling non-technical users to contribute to the configuration and security of distributed systems.

Overall, our evaluation results demonstrate the effectiveness and efficiency of our transpiler in generating accurate and efficient BPF rules for configuring and securing distributed systems, as well as its usability for non-technical users.

5 Conclusion

This research has provided several valuable insights into the process of configuring and securing distributed systems using natural language. One key finding is that the transpiler developed in this study was able to effectively convert natural language input into BPF rules that could be applied to network packets without incurring overhead. This is an important step towards making distributed systems more user-friendly and easier to manage. Additionally, the transpiler was able to handle stateful rules and manual packet manipulation, which is a useful feature for more advanced configurations. Through the evaluation process, it was also determined that the transpiler was able to accurately apply the generated BPF rules to a wide range of PCAP files and NRL rule files. In the future, this research could help to improve the service level objectives (SLOs) and service level indicators (SLIs) of distributed systems by enabling more intuitive and efficient configuration using natural language. It could also serve as a foundation for further research and development in this area.

Acknowledgement

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0106. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funders.