AN EFFICIENT SYSTEM FOR HUNTING REGULAR EXPRESSION

[1]*TATIREDDY HARATHI*, [2]*G.S. PRAVEEN KUMAR*
[1]PG Scholar, Associate Professor, CSE Department,
Aurora Scientific Technological and Research Academy,
Hyderabad, India. Email: harti7475@gmail.com.
[2]Associate Professor, M.Tech (PhD), CSE Department,
Aurora Scientific Technological and Research Academy
Hyderabad, India.

ABSTRACT- *A Regular expression is a sequence of text characters, some of which are understood to be meta characters with symbolic meaning, and some of which have their literal meaning, that together can automatically identify textual material of a given pattern, or process a number of instances of it that can vary from a precise equality to a very general similarity of the pattern. The pattern sequence itself is an expression that is a statement in a language designed specifically to represent prescribed targets in the most concise and flexible way to direct the automation of text processing of general text files, specific textual forms, or of random input strings .Regular expressions (RE) are getting popular still under developed stage of their inherent complexity that limits the total number of RE that can be known by using a single chip. This limit on the number of RE doesn't pair with the scalability of present RE detection systems. Existing schemes is limited in the old detection paradigm based on per-character-state working and also state transition detection. Keeps concentrate on optimizing the count of states and the need transitions, but not on concept of optimizing the suboptimal character-based detection method .The advantages of allowing out-of-sequence detection, rather than detecting components of a RE in order of appearance, have not been explored. LaFA needs less memory due to these three aspects providing specialized and optimized detection modules, systematically reordering the RE detection sequence and sharing states among automata for different RE's.*

Index Terms—Deep packet inspection, DPI, LaFA, Look-ahead Finite Automata, network intrusion detection and prevention system, NIDPS, regular expressions.

1. INTRUDUCTION

Flexibility in complex string patterns in large number of applications ranging from network intrusion detection and also prevention systems, compilers and DNA multiple sequence alignment was provided by RE's. NIDPS's uses RE to gives attacking signatures or packet classifiers. A RE detection system need few requirements for NIDPS high-speed networks scalability and max throughput.

A scalable detection system capable of giving Look ahead Finite Automata supports the current and expected future RE sets with less memory requirements. Finding at line speed is another important requirement of NIDPS. Our compact data structure is capable by using very high-speed on-chip memory even for large RE sets for bulky amount with high speed. Finite automata (FA) are the de facto tools to mention the RE detection problem. For RE detection on an input, the FA starts at an initial state. And each character in the input, the FA makes a transition to the next state, which is determined by the last state and the present input character. If the resulting state is unique, the FA is termed a deterministic finite automaton (DFA); or else, it is termed a nondeterministic finite automaton (NFA). These both represent two extreme cases. DFA has permanent time complexity which makes DFA the preferred approach executed quickly on commodity CPUs. NFA needs huge parallelism, making it harder to implement in software. NFA allows multiple simultaneous state transitions, leading to a higher time complexity. RE detection systems and, we believe, that limit the scalability of these systems. RE share same components. Normally FA approaches, a small state machine is used to detect a component in a RE. This state machine is changed since the similar component may appear multiple times in different RE's. Furthermore, most of the time, the RE's sharing this component cannot appear at the same time in the input. As last, the repetition of the same state machine for various RE's introduces redundancy and boundaries the scalability of the RE's detection system. Based on the above three observations, we introduce LaFA, a novel detection method that resolves scalability issues of the current RE detection paradigm. The scalable and compact LaFA data structure requires a small memory footprint and makes it feasible to implement large RE sets using only very fast, small on-chip memory.

2. RELATED WORK

An intrusion detection system (IDS) can be a key component of security incident response within organizations. Traditionally, intrusion detection research has focused on improving the accuracy of IDSs, but recent work has recognized the need to support the security practitioners who receive the IDS alarms and investigate suspected incidents. To examine the challenges associated with deploying and maintaining an IDS, we analyzed 9 interviews with IT security practitioners who have worked with IDSs and performed participatory observations in an organization deploying a network IDS. We had three main research questions: (1) What do security practitioners expect from an IDS?; (2) What difficulties do they encounter when installing and configuring an IDS and (3) How can the usability of an IDS be improved Our analysis reveals both positive and negative perceptions that security practitioners have for IDSs, as well as several issues encountered during the initial stages of IDS deployment. In particular, practitioners found it difficult to decide where to place the IDS and how to best configure it for use within a distributed environment with multiple stakeholders. We provide recommendations for tool support to help mitigate these challenges and reduce the effort of introducing an IDS within an organization. Exact string matching was the principal method used in early DPI systems and has been studied extensively. To keep up with new and highly sophisticated attacks, software-based NIDPSs started using RegEx-based signatures to express these attacks more flexibly. The introduction of RegEx-based signatures increased the complexity of the DPI, limiting the scalability of Software NIDPS or packet classifiers to high speeds. Current state-of-the-art hardware architectures are either space-efficient (NFA) or high-speed (DFA), but not both. Pure NFA implementations will be too slow in software, but hardware implementations can be feasible with a high level of parallelism. The implementation in uses solely logic gates. Although such schemes achieve high-speed RegEx detection, the inflexibility of implementing signatures on logic gates limits the updatability and scalability of NFA implementations. Mitra et al. proposed a compiler to automatically convert PCRE opcodes into VHDL code to generate NFA on FPGA.The memory requirement of DFA implementation can be very high for certain types of RegExes. Some approaches aim to reduce the memory requirement by reducing the number of states. These approaches replace the DFA for these problematic RegExes with NFA or other architectures to minimize memory consumption, while using DFA to implement the rest of the RegExes. Hybrid FA keeps the problematic DFA states as NFAs (other parts use DFA). In, the authors propose history-based finite automata (H-FA), which remember the transition history to avoid creating unnecessary states, and history-based counting finite automata (H-cFA) which add counters to reduce the number of states. The XFA proposed in formalizes and generalizes the DFA state explosion problem and shows a reduction of number of states. In, the authors introduce a DFA-based FPGA solution. Multiple microcontrollers, each of which independently computes highly complex DFA operations, are used to avoid DFA state explosions. Reference is similar to in that it compiles RE's into simple operation codes so that multiple, specifically designed micro engines (microcontroller) work in parallel. However, even after replacing the problematic DFA states with more memory-efficient structures in the above schemes, the memory consumption of the rest of the DFA is still significantly high. Other approaches propose to reduce DFA memory by reducing the number of transitions. For instance, D FA merges multiple common transitions, called default transitions, to reduce the total number of transitions. However, this approach may require a large number of transitions for some cases, leading to an increase in the number of memory accesses per input byte. In addition, D FA construction is complex and requires significant resources. Several researchers follow up on the D FA idea. CD FA and Merge DFA resolve the shortcomings of D FA by proposing multiple state transitions per character. Merge DFA bounds the number of worst-case transitions to, where is the length of the input string. Although bounded, Merge DFA still requires a relatively large number of memory accesses. CD FA can achieve one transition per input character. However, it requires a perfect hash function to do so. Although these schemes successfully address some issues in the D FA, they still cannot achieve satisfactory results for all three design objectives—namely flexibility for adding new signatures, efficient resource usage, and high-speed detection. The authors focus on optimization at the RE level before the FA is generated. They propose rule rewriting for particular RE patterns that cause state explosions. They also suggested grouping (splitting) the DFA into multiple groups to reduce the number of states.

3. PROBLEM STATEMENT

- IDS find the Intrusion using known attack patterns called signatures.
- Every IDS will have more number of signatures (more than 5000)
- If Pattern matching algorithm is slow, the IDS attack response time will be very high.
- The existing efficient algorithms such as Boyer Moore (BM), Aho-Coarasick (AC) does not improve the throughput of IDS.
- The proposed system is an Implementation of Scalable look-ahead Regular Expression Detection System.
- Works based on look-ahead Finite Automata Machine.
- Improves the detection speed or attack-response time.
- The proposed system should be capable of processing more number of signatures with more Number of Complex Regular Expressions on every packet payload.
- The attack response time should be less when Compared with Deterministic Finite Automatic (DFA) Pattern Matching Procedures(aho-coarasick).
- Should Provide pattern matching with Assertions (back References, look-ahead, look-back, and Conditional sub-patterns).
- Should use less memory (Space complexity is low)

4. SYSTEM DEVELOPMENT

1) Buffered Lookup Modules: The buffered lookup modules are core detection modules in the La FA architecture. The look ahead operation can be known by these modules. These modules use history-stored in buffers to check past activity. Next, we explain various detection modules based on the buffered lookup approach.

Timestamps Lookup Module: TLM stores the incoming character to its time of arrival. This module can detect non repetition types of variable strings such as VA, VB, and VC. TLM incorporates an input buffer, which stores characters recently received from a packet in chronological order. Using this buffer, TLM can answer queries such as Does the character at time belongs to a character class C? Let us consider the detection process of RegEx1 as an example. For the detection of RegEx1, a lowercase alphabetical character must be detected between simple strings “abc ” and “ op” Let us assume for the time being that detection sequence and detection timing is already verified by the correlation block. The last portion that needs to be verified is whether the input character between “abc” and “op” was a lowercase alphabetical character or not. In the

example, since at time 4, a lowercase letter, appears in the input, RegEx1 is detected.
RE1: abc [a-z] op| S1 V1 S2

Input: a b c x o p

Time: 1 2 3 4 5 6

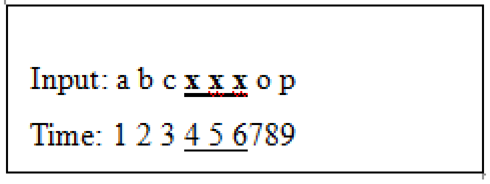
In a formal notation, let TT L M =(T1,T2,T3…..Tl) be an ordered set of the TLM contents, while is the length of the character buffer. In the example, variable string is [a-z] located at time 4. Receiving character in time 4 is verified. The query is in a statement in the form of "[a, z] □ T4 ≠□" to check for its validity.

2) Character Lookup Module (CLM):

CLM is responsible for the detection of VH-type variable strings. This is one type of variable string that drags traditional FAs down to impractical architectures. Let us follow an example to see the detection procedure of the CLM using RegEx5. We assume the detection order and timing are already verified at the correlation block. The CLM has multiple memory locations to store detection timestamps of each character. For example, the first character "a" is detected at time 1, so timestamp 1 is stored for character "a." Timestamp 2 will be stored for character "b" and so on. To match RegEx5, "x" should not appear from time 4 to time 7. Let us call this time period the inviolable period. To verify that information, the CLM uses timestamps stored previously .In the example, "x" was detected at time 7, and the time is stored at character timestamp memory in the CLM. From this information, we can conclude that the example input does not match. RegEx5 because "x" was detected during the inviolable period (at time 7)
RE 5: abc [xΛ] {3,5}op| S1 V2 S2

Input: a b c d e f x o p
Time: 1 2 3 4 5 6 789

In a formal notation of the ASCII values, let CCLM ={C0 C1….Ci} (can be 0 to 255) be an unordered set. Some components can be empty(Cx=□) . Each element Cx stores time stamps (e.g., C120={t6} shows component C120 (lowercase " x") appeared at time 6). The query is in a statement in the form of "Is [t4,t6]□C120=□" to check for its validity. For some RegExes, more than one timestamp needs to be stored per ASCII character. To clarify this need, consider following example:
REGEX 6: abc [xΛ] {3}xyz| S1 V4 S3


Input: a b c x x x o p

Time: 1 2 3 4 5 6789

This example illustrates a situation that requires more than one timestamp entry. Character "x" appears four times at times 4, 5, 6, and 7. CLM stores the timestamps. However, time stamps 4–6 are overwritten, and only timestamp 7 is stored because only one timestamp entry is assigned. In this situation, RegEx element [x]{3}cannot be verified correctly. At least two time stamp entries must be reserved in CLM for this example RegEx.1 The simulation result in Section VII shows CLM only needs to store a small number of time stamps per each of the 256 ASCII characters regardless of the input traffic. This gives us the brief description about the outline look up modules comes to in line look up modules it differs.

3) In-Line Lookup Modules:

Repetition Detection Module (RDM):

RDM is responsible for detecting repetitions that are not detected by CLM (variable strings of type VD, VE, VF, and VG).More formally, RDM detects components in the form base {x,y} by accepting consecutive repetitions of the base ,x to times in the input. Here, base can be a single character, a character class, or a simple string .The {x,y} shows a range of repetition, where is the minimum and is the maximum repetitions. The RDM is the only in-line detection module. The RDM consists of several identical sub modules, and each sub module can detect character classes and negated character classes with repetitions. These sub modules operate in an on-demand manner. Assume a RegEx detection process is in progress and the following component to be inspected is a character class or negated character class repetition. The correlation block sends a request to the RDM for the detection of this component. The request consists of the base ID and the minimum and maximum repetition boundaries, where the base is represented by a pattern ID. The RDM assigns one of the available sub modules for detecting this component. The assigned sub module then inspects the corresponding input characters to see whether they all belong to the base range and if the number of repetitions is between the minimum and maximum repetition boundaries. Once the number of repetitions reaches the minimum repetition Value, a next simple string is activated. The next simple string inactivates when the number of repetitions reaches to the maximum repetition value.

5. OVERVIEW OF SYSTEM ARCHITECTURE

- Packet capturing modules receives every packet.
- Payload extraction module, extracts the application layer packet.
- Using time stamps module (TLM) each incoming character is cross checked against non-repetition types of variable strings.
- Character look up module (CLM) is responsible for identifying frequently access character strings.
- Repetition Detection module is responsible for identifying repetition that are not detected by CLM
- Frequently appearing repetition module(FRM) it Reduces Resource usage by creating opportunity for sharing Effort of Frequent bases.

6. RELATED WORK

An intrusion detection system (IDS) can be a key component of security incident response within organizations. Traditionally, intrusion detection research has focused on improving the accuracy of IDSs, but recent work has recognized the need to support the security practitioners who receive the IDS alarms and investigate suspected incidents. To examine the challenges associated with deploying and maintaining an IDS, we analyzed 9 interviews with IT security practitioners who have worked with IDSs and performed participatory observations in an organization deploying a network IDS. We had three main research questions: (1) What do security practitioners expect from an IDS?; (2) What difficulties do they encounter when installing and configuring an IDS and (3) How can the usability of an IDS be improved Our analysis reveals both positive and negative perceptions that security practitioners have for IDSs, as well as several issues encountered during the initial stages of IDS deployment. In particular, practitioners found it difficult to decide where to place the IDS and how to best configure it for use within a distributed environment with multiple stakeholders. We provide recommendations for tool support to help mitigate these challenges and reduce the effort of introducing an IDS within an organization. Exact string matching was the principal method used in early DPI systems and has been studied extensively. To keep up with new and highly sophisticated attacks, software-based NIDPSs started using RegEx-based signatures to express these attacks more flexibly. The introduction of RegEx-based signatures increased the complexity of the DPI, limiting the scalability of Software NIDPS or packet classifiers to high speeds. Current state-of-the-art hardware architectures are either space-efficient (NFA) or high-speed (DFA), but not both. Pure NFA implementations will be too slow in software, but hardware implementations can be feasible with a high level of parallelism. The implementation in uses solely logic gates. Although such schemes achieve high-speed RegEx detection, the inflexibility of implementing signatures on logic gates limits the updatability and scalability of NFA implementations. Mitra et al. proposed a compiler to automatically convert PCRE opcodes into VHDL code to generate NFA on FPGA.The memory requirement of DFA implementation can be very high for certain types of RegExes. Some approaches aim to reduce the memory requirement by reducing the number of states. These approaches replace the DFA for these problematic RegExes with NFA or other architectures to minimize memory consumption, while using DFA to implement the rest of the RegExes. Hybrid FA keeps the problematic DFA states as NFAs (other parts use DFA). In, the authors propose history-based finite automata (H-FA), which remember the transition history to avoid creating unnecessary states, and history-based counting finite automata (H-cFA) which add counters to reduce the number of states. The XFA proposed in formalizes and generalizes the DFA state explosion problem and shows a reduction of number of states. In, the authors introduce a DFA-based FPGA solution. Multiple microcontrollers, each of which independently computes highly complex DFA operations, are used to avoid DFA state explosions. Reference is similar to in that it compiles RE's into simple operation codes so that multiple, specifically designed micro engines (microcontroller) work in parallel. However, even after replacing the problematic DFA states with more memory-efficient structures in the above schemes, the memory consumption of the rest of the DFA is still significantly high. Other approaches propose to reduce DFA memory by reducing the number of transitions. For instance, D FA merges multiple common transitions, called default transitions, to reduce the total number of transitions. However, this approach may require a large number of transitions for some cases, leading to an increase in the number of memory accesses per input byte. In addition, D FA construction is complex and requires significant resources. Several researchers follow up on the D FA idea. CD FA and Merge DFA resolve the shortcomings of D FA by proposing multiple state transitions per character. Merge DFA bounds the number of worst-case transitions to, where is the length of the input string. Although bounded, Merge DFA still requires a relatively large number of memory accesses. CD FA can achieve one transition per input character. However, it requires a perfect hash function to do so. Although these schemes successfully address some issues in the D FA, they still cannot achieve satisfactory results for all three design objectives—namely flexibility for adding new signatures, efficient resource usage, and high-speed detection. The authors focus on optimization at the RE level before the FA is generated. They propose rule rewriting for particular RE patterns that cause state explosions. They also suggested grouping (splitting) the DFA into multiple groups to reduce the number of states.

7. CONCLUSION

We explored LaFA, an on-chip RE detection system that is highly scalable. The scalability of present schemes is normally limited by the traditional per-character state processing and state transition detection paradigm. The key research existing schemes is on optimizing the number of states and need transitions, not on the suboptimal character-based detection format. In future, the potential benefits of accessing out-of-sequence detection optional of detecting components of a RE in sequence of looking have not been explored. We selected perfect separate detection operations from state transitions, allowing opportunities to another optimize traditional FAs. LaFA employs is look ahead technique to re arrange the sequence of pattern detections technique that needs less operations and can announce a mismatch before exploring complex patterns. Replacing variable strings are arranged by independent detection modules. By this solution was evaluated to the scalability problem of traditional FAs and improvement in memory efficiency of LaFA. In Comparison with state-of-the-art RE detection system, LaFA needs an order of magnitude less memory.

8.REFERENCES

[1] M. Fisk and G. Varghese, "An analysis of fast string matching applied to content-based forwarding and intrusion detection," Tech. Rep.CS2001-0670 (updated version), 2002.

[2] "Port80," Port80 Software, San Diego, CA [Online]. Available: http://www.port80software.com/surveys/top1000compression.

[3] “Website Optimization, LLC,” Website Optimization, LLC, Ann Arbor, MI [Online]. Available: http://www.websiteoptimization.com.

[4] P.Deutsch, “Gzip file format specification,” RFC 1952, May1996[Online].Available: http://www.ietf.org/rfc/rfc1952.txt.

[5] P. Deutsch, “Deflate compressed data format specification,” RFC 1951, May 1996 [Online]. Available: http://www.ietf.org/rfc/rfc1951.txt.

[6] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” IEEE Trans. Inf. Theory, vol. IT-23, no. 3, pp. 337–343, May 1977.

[7] D. Huffman, “A method for the construction of minimum-redundancy codes,” Proc. IRE, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[8] “Zlib,” [Online]. Available: http://www.zlib.net

[9] A. Aho and M. Corasick, “Efficient string matching: An aid to bibliographic search,” Commun. ACM, vol. 18, pp. 333–340, Jun. 1975.

[10] R. Boyer and J. Moore, “A fast string searching algorithm,” Commun.ACM, vol. 20, no. 10, pp. 762–772, Oct. 1977.