Network traffic analysis optimization for signature-based intrusion detection systems

Dmitry S. Kazachkin, *Student, Computational systems lab at CMC MSU,* Dennis Y. Gamayunov, *scientific advisor, PhD, Computational systems lab at CMC MSU*

***Abstract* — In this paper we propose a method for signature matching optimization in the field of intrusion detection and prevention. Signature matching algorithm performance is one of the key factors in the overall quality of the IDS/IPS, especially in high-speed networks. Optimization method proposed in this paper relies on semantics of the signature matching task, typical for such systems as Snort. The method minimizes the number of patterns called by the detection system for each network packet, reducing the time of its processing.**

***Index Terms*— Security, Intrusion detection, Networks, Snort.**

I. Introduction

NETWORK intrusion detection and prevention systems (IDS/IPS), which are quite common nowadays, mostly use signature-based method for attacks recognition. The idea of this method is in comparison of real observed network traffic with a set of known attack descriptions. The number of signatures in a typical IDS database is usually several thousands (Snort system has about 7 thousand signatures [1]). A single signature can be imagined as an ordered set of tests of network packet header and payload content – values of some characteristic header fields, typical textual substrings, regular expressions, etc. Thus, "straight" testing of all the signatures set against a single network packet results in hundreds of thousands machine operations. This is one of the key factors that hinder wide spreading of those systems at high speed networks – 1Gbit/s and beyond. Furthermore, there are 2 important trends, observed from the beginning of computers and networks evolution, called Moor's law and Gilder's law. Moor's law reads that processor's throughput of a given cost doubles every 2 years; Gilder's law says that the network throughput trebles at the same period. So, the network node's computational power growth is left behind by the growth of volume of information transferred through the network, which continuously make performance requirements of the security systems' algorithms to grow higher as well, which refers to IDSs too.

This paper proposes a method for optimized testing process of a given signature set on a single packet. The basic model for signature matching, used is this paper for evaluating the optimized method, is consecutive checking of every signature until the first success (getting "true" value), which is in fact a brute-force search. Of course, modern IDSs do not use this full search for signature matching and most of them have some heuristics for search space reduction (Snort optimizations are shown below). The proposed method was implemented for an experimental IDS, designed at the Computational systems lab of Moscow State University's Faculty of Computational Math and Cybernetics [4], and Snort's signature base released at 2007 was used as an initial pack of signatures. Implementing the optimized signature matching algorithm would be a hard task, for it implies altering the Snort core engine. Yet, for experimental IDS this method implementation is reduced to signature base translation without meddling in the IDS engine.

Experimental IDS uses a specialized behavior description language called *R-lang*, which use finite automaton for describing network object behavior. A set of such descriptions, grouped by the object type (in our case – network packets), is called *module*. R-lang language is based on ideas of Eckmann's, Vigna's and Kemmerer's works and STATL language [5,6]. Implementation of signatures matching optimization method offers an optimizing translator from R-lang modules to R-lang modules, which minimize the total number of states and transitions in the module, and also the number of testing functions called for a single event.

Evaluation of the Snort's signature base with optimizing translator revealed its essential superfluity, that partially can be explained by the fact, that this base was formed by many independent developers, and seemingly some of them were using automatic signature generators, based on attack samples. In the end of this paper is shown how proposed signature matching optimization has helped to reduce this superfluity.

II. Signature formal definition

First, we will give a formal signature definition to use it afterwards. The object of analysis of a signature is a single network *packet* P, which consist of *header* and packet *payload*. *Header* is a vector of some fixed fields $H = \{H_1,H_2,...,H_n\}$, that belong to corresponding finite spaces. *Payload* — text line P of unrestricted length.

Also, there is a variable vector $C = \{C_1,C_2,...,C_k\}$, describing the signature-based analyzer inner state (so-called

Manuscript received March 31, 2008.

D. S. Kazachkin is with the Computational systems lab at CMC MSU, Moscow, Russia (e-mail: zok@lvk.cs.msu.su).

D. U. Gamayunov is with the Computational systems lab at CMC MSU, Moscow, Russia (e-mail: gamajun@lvk.cs.msu.su).

state vector). Variables C_i also have a finite value ranges.

Header condition $CondH_i(H)$ — is a logical predicate, which takes header fields H as its arguments.

A sample of such condition is a testing of the equality of source port (one of the header fields) to value 80.

Payload condition — is an array of functions:

- $CondP_i(P,C)$ — is a logical predicate, which takes packet payload and state vector as its arguments,
- $Effects_i = \{Effect_{i,j}(P,C)\}$ — each function in this set (for i=1...k) represents the side effect on variable C_k, performed during $CondP_i(P,C)$ checking.

It is said that payload condition evaluates successfully, if $CondP_i(P,C)$ returns 1. At the same time, all the variables ib C vector simultaneously get the new values $C'_j=Effect_{i,j}(P,C)$, based on payload text and their old values. When consecutive payload conditions are evaluated, each of them "feels" side effects from already evaluated conditions, and the result is their complex superposition. This means there is no way of caching conditions evaluation result in a general case.

A sample of such condition is a test of substring presence in payload P between markers C_1 and C_2 with a side effect of moving left marker C_1 to the end of found substring.

Reaction — is an element of some finite set of event classes.

Signature — is a triplet <SH,SP,R>, where SH — set of header conditions, SP — ordered set of payload conditions, R — reaction.

Signature evaluation result RES(<SH,SP,R>,H,P,C):

- {R}, if all the header conditions return 'true' and all the payload conditions successfully evaluate consecutive.
- $\varnothing$, else

III. Converted signatures

Snort's signatures are quite strictly described by suggested formal model. Let's examine Snort's signature structure and describe the way to build a corresponding formal signature.

Snort's signature consists of 4 sections:

- *action* – action performed on rule activation (usually, 'alert'). Corresponds to reaction function.

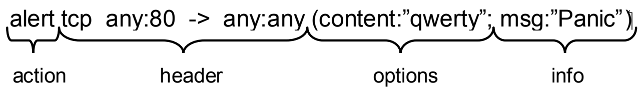

Fig. 1. Snort's signature structure.

- *header* – context-independent conditions on packet header: protocol, IP addresses and ports (if defined by protocol) of source and destination, direction. Corresponds to header conditions from the model.
- *options* – context-dependent conditions on packet payload. Consists of different tests of text, contained in payload after a single moving marker. Marker is moved depending on a condition type. Corresponds to payload conditions from the model.
- *info* – rule info and message, generated at rule activation. Corresponds to reaction function arguments.

Let's define vector H in the following way: H_1 — protocol type (TCP / UDP / ICMP / other IP), H_2 and H_3 — IP and port of packet source, H_4 — packet direction (from server/to server), H_5 and H_6 — IP and port of packet destination. Ports range is 0…65535, IPs range is 0.0.0.0 … 255.255.255.255. Regarding to so defined vector H, for any *header* field a set of header conditions $\{CondH_i(H)\}$ exists, having the same functionality.

Vector C is defined as a single-variable vector. C_1 — current state of the text marker, after which the pattern matching takes place. Regarding to so defined vector C, for any *options* chain a chain of payload conditions [<CondPi(P,C), Effecti(P,C)>] exists, having the same functionality, including both return value and side-effects.

Reactions space, which contain different classes of events we can define as a space of different possible values of *classtype* field, which is representing class of detected attack. The reaction of each signature is the value of *classtype* field. This definition is correct, because for each signature reaction exists and only one.

IV. R-lang language

R-lang is the language used in experimental IDS. It describes the behavior of observed system with a scenario – finite automaton with memory, each transition of which is marked by type, condition and body of transition.

There are 3 types of transitions – consuming (simply performs a transition), non-consuming (creates automaton exact copy and perform a transition there), unwinding (destroys current automaton copy). Condition – is a logical expression that should be true to perform transition. Body – a program instruction block, that executes during the transition.

In fact, automaton is an extension of structure concept, because it can contain functions (methods of scenario), and automaton's memory is a implemented in form of inner variables (fields of scenario), that are globally visible in all the transitions.

There is an important particularity of R-lang language: there is totally no dynamic memory in it, so the framework is safe from troubles caused by scenario errors. Also there are no global variable, visible in all the scenarios, so for a library function, designed as a function that is external to scenario, side-effect can be performed only over its direct arguments (structures are passed by reference). Functions, defined inside scenario can also perform side-effects on its fields, even if they are not passed as arguments.

V. Conversion to R-lang

A. Simple conversion

Structure similarity of R-lang and STATL languages allows using conversion ideas, described by S.T. Eckmann in [2], the paper about translating snort rules to STATL scenarios.

Network sensor of experimental IDS provides typified events to scenarios on getting every packet. Each event contains IP-addresses and ports of packet source and destination (IPSrc, PortSrc, IPDst and PortDst), its direction

(Direction), and also its direction (Payload).

Below is an algorithm of signature conversion, written in terms of formal model. The result is a R-lang scenario.

If protocol Pr (TCP, UDP, IP or ICMP) is granted by header conditions $\{CondH_i(H)\}$ (vector $H^*=\{Pr, IPSrc^*, PortSrc^*, Dir^*, IPDst^*, PortDst^*\}$ exists and all the conditions return true value on it), then the signature has a corresponding scenario, accepting events of corresponding packet type.

Each header condition has a corresponding logical expression in R-lang language, which tests network event fields, corresponding to H vector elements. This expression is a model of $CondH_i(H)$ in R-lang.

For example, corresponding code for CondH(H):(H_3=80) is “ev.tcpSrcPort==80”.

Each payload condition is assigned to a logical expression in R-lang language, which tests packet payload text P and condition vector C. This expression returns a Boolean value, simulating $CondP_i(P,C)$. Also during its evaluation condition vector C can be modified, simulating $Effect_i(P,C)$.

For example, CondP(P,C): true, if packet payload contains substring «qwerty» after the marker C_1, $EffectP_1(P,C)$ moves C_1 to the end of found substring. The code corresponding to payload condition <CondP, EffectP> is a call “content(“qwerty”, C, ev.Payload)”, where content – external function, modeling the described functionality, including side-effect on passed-by-reference vector C.

Each reaction is assigned to an alert sending code, which sends to IDS a message, containing found attack classification.

Here is the scheme of signature evaluation performing scenario in R-lang:

```
scenario sc(<event corresponding to packet type> ev)
{
   <Secondary variables definition>
   initial state st0;
   consuming transition st0->st0
   event <event corresponding to packet type> (
      <Header condition 1> && … && <Header condition N>
   ){
      if(<Payload condition 1>)
         …
            if(<Payload condition M>)
               <Reaction>;
   }
};
```

Before optimizations, described in this paper, Snort2R-lang converter worked this way.

B.Header-based optimization

Conditions alternative — a set of pairs $<SP_i,R_i>$, where SP_i – ordered set of payload conditions, R_i — reaction. Alternative-containing signature – a pair <SH, SA>, where SH — set header conditions, SA — condition alternative.

Set of signatures with the same header are converted to alternative-containing signature this trivial way:

$$\{<SH, SB_i, R_i>\} \rightarrow <SH, \{<SB_i, R_i>\}>$$

Alternative-containing signature evaluation result is defined as:

$$RES(<SH,SA>,H,P,C) = \bigcup_{i=1}^{N} RES(<SH,SP_i,R_i>,H,P,C)$$

Also, let’s define condition alternative evaluation result:

$$RES(SA,P,C) = \bigcup_{i=1}^{N} RES(<\phi,SP_i,R_i>,H,P,C)$$

Here is the structure of transition body at the R-lang model of alternative-containing signature (anything else is just as in simple signature scenario):

```
C1=C;    if(<Series of payload conditions 1>) <Reaction 1>;
C=C1;    if(<Series of payload conditions 2>) <Reaction 2>;
…
C=C1;    if(<Series of payload conditions N>) <Reaction N>;
```

Tests on Snort signature base revealed considerable economy on header condition evaluation provided by this optimization: there are only 519 different header condition sets for base of 6372 signatures.

The burden of this optimization is in replacement of excess header condition evaluations with context vector restoring operation, much simpler in the case of Snort’s signatures, where context vector contain one integer only.

Let a set of alternative-containing signatures is given, granting the same protocol. Because of context independency of header conditions, they could be evaluated in tree-style order that provides a good economy.

This additional optimization can be organized in R-lang by scenario combining with the use of nested “if”s:

```
consuming transition st0->st0
event <event corresponding to packet type>(true){
   if(<Header condition 1>
      if(<Header condition 2>)
         <predicate alternative for header granted by 1,2>
      if(<Header condition 3>)
         if(<Header condition 4>)
            <predicate alternative for header granted by 1,3,4>
}}
```

Snort analysis engine use that header optimization only, that does not allow achieving further speed-up on a fixed signature set.

C.Predicate tree

The computational complexity of payload condition evaluation usually much higher than the computational complexity of header condition evaluation. That’s why the task of optimizing conditions alternatives evaluation is urgent.

Condition chains contained in conditions alternative can have the same beginnings. A huge benefit can be achieved by using this fact.

Predicate tree ST – tree:

- the edges are marked with a payload condition
- the nodes are marked with a reaction set, possibly empty
- the leafs are marked with non-empty reaction sets

Algorithm of building a predicate tree based on a condition alternative is intuitive clear: when adding a next chain of payload conditions to a tree, a pointer moves from root node through the edges marked with corresponding payload conditions if they exist, otherwise a new branch, containing left conditions is created and linked to the current pointer position.

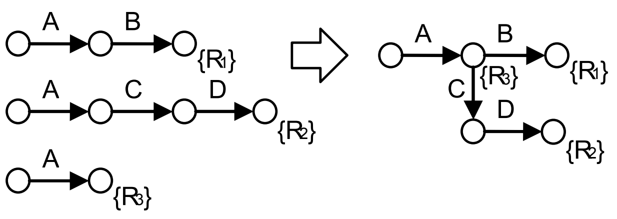

Fig. 2. Predicate tree building sketch.

Predicate tree evaluation – recursive traversal of tree from root node. Sub-trees are evaluated only if corresponding condition could be evaluated successfully in current context.

Predicate tree evaluation result RES(ST, P,C) – a set of all the reaction, that were achieved during the predicate tree evaluation.

Statement. Let predicate tree ST is converted from condition alternative SA. Then RES(ST,P,C)=RES(SA,P,C).

D.Static result analysis

Due to that optimization, a high benefit is achieved at some scenarios.

Here are defined two static characteristic of this optimization. Tree profit – is the difference between edges number in the tree and the total number of payload conditions in a conditions alternative that was the source of the predicate tree. Relational tree profit – is the ration between thee profit and total number of payload conditions in a conditions alternative.

For two signature groups in the Snort Base the tree profit exceeds 11000 for each group, and the relational tree profit for them is about 80%. These groups contain about 2000 signatures, i.e. about 1/3 off the whole base. On average, the relational tree profit is about 62%.

Also, while building those trees, 38 pair of identical signatures and 2 groups of six identical signatures were found.

The number of context restoring is not increased at all comparing to condition alternative usage. The number of context backups is increased by the number of additional branching, which is not more than the tree profit. Thus, considering heaviness of payload condition evaluation, burden of this optimization is insignificant in comparison with economy, achieved by the lowering the number of conditions.

E.Additional optimizations

Consider a set of leafs, that are all marked with same reaction R and being direct children of the same node.

Sometimes it is possible to think of a payload condition B, so RES(<∅,[B],R>,H,P,C) would be equal to evaluation result of a sub-tree that includes these given leafs and a parent node, and the computational costs of B evaluation is lower than costs for evaluation of A_1, A_2... A_n in total.

For example, payload conditions, which perform regular expression analysis (pcre) can be combined into a single predicate, that performs the search of first pattern, found of this set.

For implementing this optimization for R-lang scenarios,

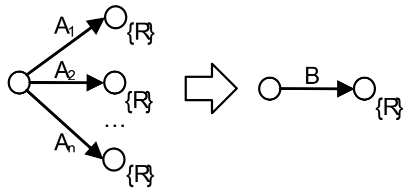

Fig. 3. Additional optimization scheme.

callout mechanism of PCRE library was used. It allows not only to stop the ongoing regular expressions set testing after the first match, but also returns id of matched pattern, which allows to inform IDS of matched signature id and classification text, not only classtype.

Also it is proposed to unite payload predicates, performing substring search (contest) using Aho-Corasick algorithms [3], which allows saving on substring search, performing all the patterns look-up at once.

These predicate combination methods of quite effective when performing conditions set evaluation until the first match. But when searching all the matches (that is necessary, if the branching node is not pre-leaf) at some strings, this method works even worthier, than separate matching. Nevertheless, they demonstrate a good performance in average, so their application for non-pre-leaf branching nodes needs further research.

VI. Conclusion

Using the proposed optimization method with the Snort signature base allowed reducing the number of testing functions calls by 60% (see table below). Considering that most of superfluous conditions were “heavy-weigh” functions such as substring search and regular expressions matching, the real gain could be even higher. A series of experiments using different types of traffic is planned to find out the numerical evaluation of that gain.

The ideas, used in optimizing translator are with minimal changes applicable for some R-lang constructions, not overviewed in this paper. Here are some examples of this constructions: several consequent transitions of the same type to the same state (condition of transition is a analogue of a payload conditions chain, because some side-effects are possible here, body of transitions is and analogue of reaction); several consequent if-blocks, etc. Implementing these ideas would help to develop a universal optimizing translator for this language.

Table I
Static characteristics of proposed optimization

Characteristic	The source signatures	Using proposed optimization
Number of header conditions sets	6372	519
Number of payload conditions	39299	15049

This table illustrates advantages of proposed method. The presented data was collected using Snort IDS signature base dated 28.01.2007.

REFERENCES

[1] Snort IDS, http://www.snort.org

[2] S.T. Eckmann, "Translating Snort rules to STATL scenarios" presented at the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001), Davis, CA, October 2001, LNCS 2212, pp. 69-84.

[3] M. Norton, "Optimizing Pattern Matching for Intrusion Detection," white paper, Sourcefire Inc., 2004 [Online] Avaliable: http:// docs.idsresearch.org/OptimizingPatternMatching/ForIDS.pdf.

[4] D.U. Gamayunov, “Network objects behavior analysis for detecting computer attacks” PhD thesis, Faculty of Computational Math and Cybernetics, Moscow State University, Moscow, 2007.

[5] S.T. Eckmann, G. Vigna, and R. A. Kemmerer. “STATL: An Attack Language for State-based Intrusion Detection” Dept. of Computer Science, University of California, Santa Barbara, 2000.

[6] G. Vigna, R. Kemmerer, "NetSTAT: A Network-based Intrusion Detection Approach." Proceedings of the 14th Annual Computer Security Application Conference, Scottsdale, Arizona, December 1998.

[7] M. Roesch. "Writing Snort Rules: How To write Snort rules and keep your sanity" [Online] Avaliable: http://www.snort.org.