A CONSTRAINED APPROXIMATE SEARCH SCENARIO FOR INTRUSION DETECTION IN HOSTS AND NETWORKS

Slobodan V. Petrović

Norwegian University of Science and Technology (NTNU),
Gjøvik, Norway

Abstract:
It is well known that most new attacks against computer systems and networks originate from the old ones. Namely, it is possible to change the old attack patterns in such a way that the modified patterns affect approximately the same targets on the victim system and pass undetected by signature-based Intrusion Detection Systems (IDS) or other detection tools. In this paper, we consider a scenario where an old attack pattern is changed by means of an automatic tool. The structure of changes must be kept under control in order for the attack to remain effective. For example, the number of changed symbols in an automatically crafted string in the attack pattern must be limited. Otherwise, this string would not affect the victim system in the same way as in the original attack. Under such an assumption, we describe the requirements for a search algorithm implemented in the detection tool (for example, an IDS) that would be capable of detecting the changes in the old attack signature. We present the basic structure of a generic search algorithm of this kind, describe some application scenarios and discuss the effectiveness of the algorithm under these scenarios.

Key words:
intrusion detection, misuse detection, non-deterministic finite automaton, simulation, approximate search.

1. INTRODUCTION

Intrusion detection in hosts and networks is automatic detection of security policy violations. Monitoring systems that perform this kind of activity are called Intrusion Detection Systems (IDS). Most IDS compare the incoming traffic (or security logs) with the records of attack signatures contained in a signature database. This process is called Misuse Detection. The attack signatures define what is abnormal for the defended computer system/network and every match is reported in the form of an alarm. The comparison is exact, which is a source of vulnerability since an attacker can easily modify the attack traffic in such a way that the signature present in the IDS does not match and the traffic launched towards the victim is still harmful. On the other hand, too many changes to the original attack traffic template performed by an attacker can make the resulting traffic behave unpredictably. Because of that, the number and distribution of possible changes to the old attack traffic pattern are limited. In addition, the attackers often use automatic tools to perform these changes since the amount of traffic to process is huge. These tools must be configured before the traffic is processed and configuration determines the number and distribution of changes, among other parameters.

Correspondence:
Slobodan V. Petrović

e-mail:
slobodan.petrovic@ntnu.no.

DOI: 10.15308/Sinteza-2016-118-123

In order to cope with the vulnerability of IDS that is a consequence of using exact search, it is possible to replace the exact search algorithm with an approximate search algorithm. In such a way, up to a tolerance level determined in advance, the IDS would be capable of detecting several variants of the same attack by keeping just one attack signature in its database (see for example [1]).

There are two challenges related to the use of approximate search in intrusion detection. The first one is related to the efficiency of operation. Namely, the approximate search algorithms are slower than their exact search counterparts and we have to make sure that an IDS implementing an approximate search algorithm is efficient enough to process the incoming traffic and/or log files in real time. The second challenge is a possibility of generating additional false positives and/or false negatives due to the fact that an approximate search algorithm without any limitation regarding the number and distribution of changes performed on the original attack traffic pattern would report a match even on the traffic patterns that could not be generated by small modifications of the original attack.

In this paper, we define the criteria for approximate search algorithms that they must satisfy in order to be applied in intrusion detection. We propose a generic algorithm that would satisfy these criteria and describe some scenarios of its application. The effectiveness of the algorithm regarding the speed and false positive/negative rate is also discussed.

The structure of the paper is the following. In Section II, we define the basic criteria that any approximate search algorithm has to satisfy in order to be applied in intrusion detection and present a generic approximate search algorithm for intrusion detection. In Section III, we describe the application scenarios for this algorithm and discuss its effectiveness. The conclusion is given in Section IV.

2. THE GENERIC SEARCH ALGORITHM

Search algorithm quality criteria

Basically, the challenges related to the use of approximate search algorithms in intrusion detection that we enumerated above determine the criteria of their quality. Regarding the speed, real time operation of an IDS is essential since the attack traffic has to be alerted upon before any harm is done. This is a challenge even for exact search algorithms because of the speed of today's network traffic. Thus, the computational overhead introduced by approximate search in an intrusion detection system must not significantly reduce its speed.

Regarding the false positive/negative rate, we have to reduce the probability of appearance of false positives and false negatives that are produced as a consequence of using approximate search since the false positives and false negatives are also produced in an IDS for other reasons and otherwise represent a great problem, reducing the trust of the users to the alarms generated by these systems.

To satisfy the quality criterion related to operation speed defined above, it is required that we use search algorithms that include parallel processing. The fastest known algorithms of this kind exploit so-called bit-parallelism phenomenon, *i.e.* the fact that it is possible to use computer words to simulate Non-deterministic Finite Automata (NFA) assigned to search patterns.

To satisfy the quality criterion related to the number of false positives/negatives, it is necessary to introduce some constraints in the approximate search. These constraints reflect the properties of the algorithm that the attacker uses in order to produce the new attack traffic patterns starting from the old ones. For this to be effective, we must know the probabilities of change/edit operations that the attacker uses in this process. The key fact here is that the changes on the old traffic patterns must be small. Thus, by studying the existing attack signatures, it is possible to predict what kind of changes are feasible on to the corresponding traffic and to define adequate constraints in search.

Satisfying the quality criterion related to the numbers of false positives/negatives is not independent of the operation speed criterion. The reason for this is the computational overhead that is introduced to the approximate search algorithm without constraints once the constraints are introduced in it. Because of that, we have to design the constrained approximate search algorithms for application in intrusion detection in such a way that we do not violate the satisfaction of the operation speed criterion. We have to achieve a trade-off between efficiency and (low) probability of false positives and/or negatives.

Bit-parallelism and search

We explain the bit-parallelism phenomenon by studying several examples. Consider the search pattern w="quick" of length m=5 and the search string S="The quick brown fox" of length n=19. A naïve search algorithm

would place the search pattern along the search string starting from the first position in the search string and compare the corresponding characters. An exact match would be reported if every character of the search pattern would be equal to its counterpart in the search string. Then the search pattern would be moved one position to the right and the whole procedure would be repeated. We would continue this process until the position *n-m+1* in the search string. If we have as many computers running in parallel as we wish, then we can run *n-m+1* character-by-character comparisons concurrently and finish the search in a single step. We can imagine that each time a new symbol from the search string appears, a new finite state machine performing the character-by-character comparison assigned to the search pattern is created and all the machines created before that point together with the newly created one perform the comparison of the current symbol from the search string with the current character of the search pattern. If any of the machines cannot match the current character from the search string it becomes *inactive* (*i.e.* it will not process more characters in the future), otherwise it remains active.

The problem with this kind of parallelism is that it is not binary (it involves character-by-character comparisons) and as such it can only be simulated on a digital (*i.e.* binary) computer. Consequently, simulation of parallel processing this way does not improve efficiency of the (exact) search algorithm.

Baeza-Yates and Gonnet [2] showed that it is sufficient to consider only the status (active or inactive) of the machines that were created while processing the characters of the search string *S*. This converts our parallel processing search algorithm into a binary one. Then we can use the binary operators (shift, OR, AND) on all the bits of the computer word of length *m* at the same time, which speeds up the search process by the factor *m* provided the search pattern is shorter than the length of the computer word (usually 32 or 64 bits).

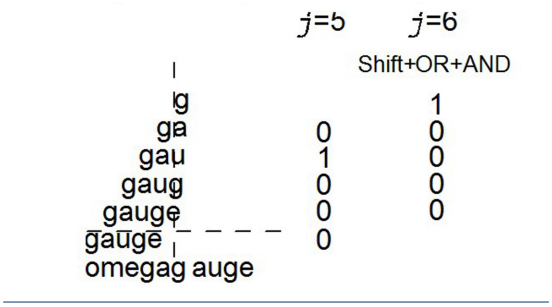

Fig. 1 – Bit-parallelism (see text)

Consider the following example. Let the search pattern be *w*="gauge", *m*=5 and the search string be *S*="omegagauge", *n*=10". We assign a finite state machine performing character-by-character comparison to the pattern *w*. We keep track of the status of each machine created during the search process in a computer word *D* of length *m*. We call this computer word the *search status word*. A zero in the status word means an inactive machine and a one means an active machine. Whenever a new machine is created (before we process the next symbol from *S*) its status is active. After processing *j* symbols from the search string *S*, some machines are active and some are inactive. We can only have *m* machines at a time, so before processing a new symbol from *S* we have to eliminate the oldest machine. We do this by shifting the status word one position to the left. The fact that the newly created machine that will start processing from the *j*-th symbol of *S* is active before processing that symbol is equivalent to OR-ing the status word with 1. We now notice that if a machine is active waiting for a certain character only that character will keep it active and this fact does not depend on the search string. Thus, we can define bit masks *B*[s] assigned to each character s of the search pattern w prior to processing the search string *S*. The bit mask *B*[s] has the value 1 at the positions where the search pattern has the character s, read from right to left. In our case, *B*[g]=01001, *B*[a]=00010, *B*[u]=00100, *B*[e]=10000. For the characters outside of the search pattern *w* the bit mask is zero. After shifting the status word D by one position to the left and OR-ing it with 1, we AND it with the bit mask of the current character. We call this process *updating the status word*. The updating formula is then [2]

$$D_j = ((D_{j-1} << 1)\ OR\ 1)\ AND\ B[S_j], \qquad (1)$$

where *Sj* is the *j*-th character of the search string S.

Thus, the search process is reduced to updating the status word *D* for each input symbol from the search string *S* and checking whether the most significant bit (MSB) of *D* is equal to 1. In that case, we have an *occurrence* of the search pattern *w* in the search string *S*. Note that the original "Shift-AND" search algorithm from [2] based on bit-parallelism status word updating formula (1) is an *exact search* algorithm.

Unconstrained approximate search

Suppose we allow up to *k* errors in the search process and the errors can be results of insertions, deletions or substitutions of characters. A common way to design a tolerant search algorithm of this kind based on bit-

parallelism is to use the so-called Row-wise Bit Parallelism (RBP) [3]. In such an algorithm, instead of having a single search status word D, we use a search status array R containing $k+1$ search status words of length m (the length of the search pattern w). The 0-th row (R_0) of the array corresponds to exact search, the 1-st row (R_1) corresponds to the search with 1 error, *etc.* The search status array updating formula is more complicated than in the exact search case. To understand it, we have to imagine a Non-deterministic Finite Automaton (NFA) array that is simulated by the RBP. It is an array of machines performing matching of a single character connected with transition arrows. A horizontal arrow means a match of the current input character from the search string S. A vertical arrow means an insertion of that character, a solid diagonal arrow means a substitution of the current input symbol and a dashed diagonal arrow means a deletion of the current input symbol.

The status word updating formula for the 0-th row of the array is the same as (1), *i.e.* the "Shift-AND" formula. For the rest of the rows, each row status word depends on the previous one. So, the final status word array updating formula for the unconstrained approximate search using RBP is [3,4]

$$R_0' = ((R_0 << 1)\ OR\ 1)\ AND\ B[S_j],$$
$$R_i' = ((R_i << 1)\ AND\ B[S_j])\ OR\ R_{i-1}\ OR\ (R_{i-1} << 1)\ OR\ (R_{i-1}' << 1), \quad (2)$$

where R' represents the new status of the array, after the updating. Note that the status updating formula for the rows other than 0 consists of 4 parts corresponding to match, insertion, substitution and deletion, respectively. The contributions of all the operations are superimposed, which is represented by OR-ing in the formula.

Constrained approximate search

To cope with false positives and false negatives that are produced as a consequence of using approximate search, we have to introduce certain constraints in the search process, as discussed above. The constraints are taken into account in the Row-based Bit Parallelism algorithm by assigning special counters or counter arrays to each bit of the status array. The role of the counters is to keep track of allowed edit operations from a state of the NFA array. These operations depend on the nature of the constraints. For example, if the constraints are on the total numbers of edit operations allowed in the search process, then each counter contains the remaining number of edit operations allowed from the corresponding state. The search status updating formula in the constrained case encompasses not only updating the values of the bits of the status array, but also updating the values of the constraint-related counters.

Generic constrained approximate search algorithm

After explaining bit-parallelism and the unconstrained and constrained bit-parallel approximate search, we are now ready to present a generic constrained approximate search algorithm. It contains counters that have to be updated after updating the status bits of the NFA array. To update these status bits, we first have to determine whether certain transitions are allowed from each state. This is done by introducing additional bit masks and AND-ing them with the status words corresponding to each row of the array. Such bit mask is always equal to 1m for the 0-th row of the status array. For other rows, the bit mask values depend on the values contained in the counters. If the values of the counters reach their allowed maxima (or minima, depending on the constraints), then the corresponding bit value in the bit mask will be 0. A zero bit in such a mask means that the corresponding transition will not be allowed at updating the status array.

The generic constrained approximate search algorithm is presented below. The concrete operations necessary to update the counters and produce bit masks depend on the constraints.

Algorithm 1 – Generic constrained approximate search

Input:

The search pattern $w=w_1w_2\ldots w_m$
The search string $S=s_1s_2\ldots s_n$
The number of allowed errors k; sometimes, k is determined by the values of the constraints
The constraints, usually in the form of an array of numbers C; the meaning of the numbers depends on the nature of the constraints.

Output:

A set of positions where w was found in S (can be empty).

1. Initialization – i least significant bits are set to 1 in the row i of the status array R, $i=0,\ldots,k$
2. For $j=1,\ldots,n$
   2.1. Update the row 0 of the status array
   2.2. For the rest of the rows of R:
      2.2.1. Check the values of the counters for each bit of the row

2.2.2. Generate the bit mask depending on the values of the counters
2.2.3. Update the status word for the row (equation, (2)) and AND it with the bit mask
2.2.4 Update the counter values for the row
2.3. Check the MSB of each row of the status array – if it is equal to 1 then we have an occurrence of *w* in *S*.

The model of the attacker behavior and the tool that the attacker uses to transform the known attack pattern to a new attack determine what constraints will be applied in the search algorithm. These constraints convert the generic approximate search algorithm into a concrete one.

3. APPLICATION SCENARIOS AND EFFECTIVENESS

We consider two scenarios for application of the generic constrained approximate search algorithm presented in the previous section. In the first scenario, the total number of edit operations (insertions, deletions and substitutions) represents the constraints. In the second scenario, the maximum lengths of runs of deletions and insertions represent the constraints.

If the total number of edit operations represents the constraint, then we associate the counter array C=[*i,e,s*] to each bit of the status array, where i denotes the number of insertions, *e* denotes the number of deletions (erasures) and *s* denotes the number of substitutions that are allowed for any transition originating from that status bit. If any of the numbers *i, e,* or *s* is equal to 0, the corresponding transition is not allowed. The initial values for the triples [i,e,s] are equal to [*I,E,S*] *i.e.* the constraints given in advance. Obviously, $I+E+S=k$ in this case. Each time a transition takes place, the corresponding number of allowed edit operations is reduced by 1. If an insertion takes place, i is reduced by 1 and copied to the corresponding constraint value of the destination status bit. If a deletion takes place, e is reduced by 1 and copied to the corresponding constraint value of the destination status bit. If a substitution takes place, s is reduced by 1 and copied to the corresponding constraint value of the destination status bit.

If the maximum lengths of runs of insertions and deletions represent the constraints, then we have to be aware of the fact that, in addition to the counter values of consecutive deletions and/or insertions, we have to keep track of other operations that, if they take place, reset these counters. For example, a deletion, a substitution or a match resets the counter associated to insertions and an insertion or a substitution or a match resets the counter associated to deletions. The bit masks in this case become more complicated and must be associated to insertions and deletions at updating the rows of the status array other than 0.

By effectiveness of a search method in this paper we assume time and space complexities combined with probabilities of false positives and false negatives obtained with the IDS implementing the method. It is obvious that space complexity in both application scenarios is increased since counters (*i.e.* additional memory) are involved. However, as explained earlier, we are considering only small changes of the original attack patterns and because of that, the increase in space complexity cannot be such that it could threaten memory capacity on a sensor implementing the IDS. The same holds for time complexity. Increased time complexity is a result of updating the counters, in any application scenario. However, once the counters have been updated, operating the additional bit masks is as fast as other operations already present in the unconstrained approximate search algorithm. In addition, since the assumed changes of the original attack traffic pattern are small, very often just up to 1 operation, the use of the counters does not contribute too much to the overall complexity of the search algorithm.

Regarding the false positive and false negative rates, assuming small changes to the original attack traffic pattern and modeling such changes by the constraints introduced in the approximate search algorithm automatically prevents alarming on the traffic that does not correspond to the model. This traffic would be alerted upon without the introduced constraints.

4. CONCLUSION

In this paper, we gave an overview of possibilities to handle the situations in misuse-based intrusion detection, in which an attacker deliberately changes the attack patterns known to the defended host/network (*i.e.* the one, whose signature is present in the misuse database) in order to pass unnoticed by the victim. To this end, the attackers usually use tools since the amount of traffic to process is huge. Such tools must be configured regarding the number of changes to introduce and this number must be small since otherwise the transformed attack traffic might behave unexpectedly. This fact can be exploited on the defended side. First, to detect the transformed traffic pattern, the misuse-based Intrusion Detection System (IDS) can use approximate search instead of exact search

(which is otherwise almost always used). Second, to reduce the number of false positives and false negatives, certain constraints can be introduced in the approximate search algorithm, which model the behavior of the attacker/the tool that he/she uses. A generic constrained approximate search algorithm is presented in the paper, which takes into account general constraints (of virtually any kind). Then, two application scenarios are described that model the behavior of the tool used to transform the old attack traffic pattern. Each scenario has its particular definition of constraints, which can be substituted in the generic search algorithm in the corresponding application. The computational overhead introduced by introducing the constraints must be kept under control in order for the IDS to continue operating in real time. This is ensured by the fact that the changes to the original attack traffic pattern that the attacker can introduce must be small. On the other hand, using constraints guarantees reduction of the false positive and false negative rates since the traffic that does not correspond to the model of the attacker's/tool behavior is automatically discarded.

REFERENCES

[1] J. Kuri and G. Navarro, "Fast multipattern search algorithms for intrusion detection," in String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on, 2000, pp. 169-180.

[2] R. Baeza-Yates and G. H. Gonnet, "A new approach to text searching," Commun. ACM, vol. 35, no. 10, Oct. 1992, pp. 74-82.

[3] S. Wu and U. Manber, "Fast text searching allowing errors," Commun. ACM, vol. 35, no. 10, Oct. 1992, pp. 83-91.

[4] G. Navarro and M. Raffinot, Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences. New York, NY, USA: Cambridge University Press, 2002.