Exscind: Fast Pattern Matching for Intrusion Detection Using Exclusion and Inclusion Filters

Monther Aldwairi and Duaa Alansari
Department of Network Engineering and Security
Jordan University of Science and Technology
Irbid, Jordan
munzer@just.edu.jo, dwalansari08@cit.just.edu.jo

***Abstract*—The need for efficient intrusion detection systems increases every day to protect network traffic against emerging attacks. Unfortunately, increasing network speeds and number of signatures makes it harder for the existing signature-based intrusion detection systems to keep up. This makes those systems the weak link and the bottleneck which decreases the overall network performance. Researchers found that 30%-60% of the overall processing time of signature-based intrusion detection systems is spent on pattern matching operations [1]. In this paper, we present a novel and fast software-based pattern matching algorithm to reduce the number of times to perform pattern matching. This new algorithm introduces an exclusion-inclusion filter programmed only with signatures prefixes. It filters out the clean traffic without requiring pattern matching and weeds out suspicious packets to be searched using a specially modified Wu-Manber pattern matching algorithm. The exclusion-inclusion filter is a modified Bloom filter that produces a list of probable matching signatures for each suspect packet. The remaining few suspicious packets are searched only for the probable matches. Compared to the Wu-Manber algorithm used in intrusion detection systems, the experimental results indicate a speed up of 3.4 times on average, 5.5 times for regular traffic, and 1.6 times for worst case traffic. The memory overhead added by the algorithm was limited to 0.11%.**

Keywords-intrusion detection; network security; pattern matching; Snort; Bloom filters

I. INTRODUCTION

The Internet is integrated in all kinds of personal and business activities. With more and more services turning online and with the growing Internet connectivity and speed, the risk of putting private data at jeopardy increases. The need for faster, accurate and smart protection systems is urgent. Intrusion Detection Systems (IDSs) are popular in protecting network traffic against intruders. IDSs collect and analyze ingress and egress packets looking for suspicious contents or behaviors and alert the network security administrator. They are classified depending on the detection technique into anomaly-based and misuse-based. Anomaly-based IDS uses machine learning techniques to profile the normal network behavior and classify the incoming traffic into either normal or abnormal. A major advantage of anomaly-based IDS is the ability to detect new attacks. However, they suffer from slow speeds and high false positives. On the other hand, misuse-based often referred to as signature-based IDSs employ exact pattern matching algorithms to look for specific patterns, called attack signatures, within a packet stream. Signature-based IDSs are the preferred protection technique because they are faster, more accurate and have low false positives. But they suffer from the inability to detect emerging attacks that do not have signatures yet. In addition, signatures are drafted manually, making the IDS as accurate as the security threat analyst who authored the signatures. None the less, signature-based IDSs remain the most popular and widely deployed.

At the core of the signature-based IDSs is the pattern matching algorithm which matches the incoming packets to the attack signatures database. Research has shown that between 30%-60% of total signature-based IDS processing time is spent on pattern matching, making it the bottleneck and most computationally extensive task of intrusion detection [1]. In addition, new attacks pop up daily and therefore the number of signatures increases making the IDS task even harder. The number of Snort rules containing signatures increased from 1,542 rules in 2003 [2] to 9,945 rules in 2011 [3]. To make matters worse, the Internet speed is ought to double every eighteen months according to Moore's law and the Internet traffic is doubling every six months [4]. This makes the window for performing pattern matching smaller and smaller. Unfortunately, the existing signature-based IDSs cannot meet the speed demands imposed by both high network speeds and increasing number of signatures.

To remedy that, we propose a new fast and memory-efficient software-based pattern matching algorithm to speed up signature-based IDS. We call it Exscind which means to exclude from the union. The contributions of this paper are twofold: a new exclusion-inclusion filter and a modified pattern matching algorithm. This algorithm programs and queries the filter to determine if an incoming packet is benign or suspicious. This helps exclude and skip the search of all benign packets. For the remaining suspicious packets, the filter reports probable matching signatures to be included in the search process. In addition, the filter marks the location of the first probable matching signature in the packet. Exscind modifies the Wu-Manber pattern matching algorithm in a novel manner to minimize the number of patterns to be searched. The new algorithm searches every suspicious packet for only the probable signatures reported

978-1-4577-1127-5/11/$26.00 ©2011 IEEE

by the filter. Moreover, the modified Wu-Manber skips much of the packet and starts the search at the the position of the first probable match. Numerous experiments are performed to evaluate and compare Exscind to Wu-Manber and other algorithms in the state of the art literature.

The rest of the paper is organized as follows. Section II explains the necessary background. Section III briefly surveys the related work. Section IV explains Exscind algorithm. Section V presents the traffic traces analysis, defines the performance metrics, and discusses the results of speed and memory simulations.

II. BACKGROUND

Before presenting Exscind, the following Subsections provide the necessary background to understand pattern matching for IDS and basic Bloom filters theory.

A. Snort

Snort has been used by most researchers in the literature for evaluation purposes. It comes with a database of thousands of rules most of them contain attack signatures [5]. Fig. 1 shows a snippet of an actual Snort rule where "alert" is the action to be taken if a packet matches the rule. "TCP" is the protocol and "$EXTERNAL_NET" is an environment variable representing the source IP. Other protocols supported include UDP, ICMP and IP. "Any" means any port number or range. The arrow separates the source from destination {IP, port} duple and indicates incoming or outgoing packet flows. "$HTTP_SERVERS" is a variable for the destination IP address and "80" is the destination port number. “msg: "WEB-IIS CodeRed v2 root.exe access"” is the message to appear in logs if the rule is matched. “Content:"/root.exe"” is the signature to search the packet payload for. Special pipe operators “|10|” allows enclosing HEX characters within the contents string. "Sid:1256" is the rule identifier indicating the signature ID (SID).The rule in Fig. 1 is read as follows. Alert the administrator if an incoming packet is directed to one of the local HTTP servers and tries to execute “ /root.exe”.

B. Pattern Matching for IDS

The process of matching the rules contents or signatures to packet payload boils down to pure pattern matching. Snort uses three pattern matching algorithms: modified version of Boyer-Moore (BM) [6], Aho-Corasick (AC) [7] and Wu-Manber (WM) [8]. The algorithms can be categorized into: single and multiple pattern matching. Single pattern matching searches for only one pattern at a time while multiple pattern matching searches the packet for all patterns at the same time. The main drawback of the single pattern matching is that the packet has to be scanned once for every string, which is very time consuming.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS
80 (msg:"WEB-IIS CodeRed v2 root.exe access";
        content:"/root.exe"; sid:1256)
```

Figure 1. Snort rule example

C. Wu-Manber

The Wu-Manber algorithm is the fastest and most efficient multiple pattern matching algorithm introduced by Udi Manber and Sun Wu in 1994. It extends the bad character heuristic of BM algorithm to a bad character block. Moreover, it outperforms the AC algorithm by adopting hash tables as opposed to finite automata which consumes more memory and time. WM algorithm consists of two phases: preprocessing and searching.

1) Preprocessing Phase

In this phase, WM computes the minimum length of the patterns (m). Then, WM works on blocks of size B and builds three hash tables: SHIFT, HASH, and PREFIX. The SHIFT table is a hash table that maps each substring of size B characters into a shift value representing the number of characters to skip on a mismatch and it is constructed as follows. Let X be a block of characters, then there are couple of scenarios: if X does not appear in any pattern at all, then the shift value will be the default value of $SHIFT[i] = m - B + 1$ characters. The second scenario is if X appears in one or more patterns, then the shift value will be $SHIFT[i] = m - q$, where q is rightmost position that X occurs in any pattern. WM calculates the shift values by mapping each substring of any pattern P_i of size B into the SHIFT table. The HASH table is indexed by the same hash function used for the SHIFT table for faster access and it aims to prevent comparing a substring to all patterns in the pattern list. For each character block with zero shift value the HASH table lists all signatures containing that block. The PREFIX table is used to speed up the HASH table search and contains the hash values for the prefixes of the patterns in the HASH table [8].

2) Searching Phase

In the search phase, WM divides the incoming packet in a sliding window fashion and hashes the first block. Next, it checks the SHIFT table to find a corresponding shift value. There are two possibilities: the shift value is greater than zero, then the sliding window is shifted by that value and a new hash value is computed for the new block. The second possibility is zero shift value indicating a match possibility, then both HASH and PREFIX tables are used to verify if there is an actual match.

D. Wu-Manber Example

Table I shows the SHIFT table for the signatures shown in Table II. The signatures are extracted from Snort rules database version 2.9.0.4 [9]. The block size B is *3* and the minimum pattern length m is *6*. The SHIFT table is initially filled with the default value which equals $m - B + 1 = 6 - 3 + 1 = 4$. For substring “log” which exists in signature number 162 “logged in”, the shift value is $m\text{-}q = 6 - 3 = 3$. The SHIFT table only includes substrings which exist in signatures. To keep the table size under control, all other substring combinations that are not part of signatures are summed up under “others”. Fig. 2 shows the corresponding HASH and PREFIX tables. The HASH table includes lists of signatures containing substrings that have a shift value of

zero {ge/, ged, DIR}. While the PREFIX includes hash values for signature prefixes to speed up the matching.

Fig. 3 shows the steps to scan the input text "ztimage/lkSYSDIRo" for the patterns in Table II. In step one, start with the sliding window of {ztimag}. The last *B* characters {mag} are hashed and the SHIFT table is accessed to retrieve the shift value of two. The window is shifted by two to become {image/}. In step two, the last *B* characters {ge/} have a shift value of zero indicating a possible match. Consequently, check the HASH and PREFIX tables to verify the exact pattern match and obtain the matched pattern {image/} as a result. The next step slides the window by one to become {mage/l}. The last *B* characters are {e/l} and have a shift value of four making the next window to scan {/lkSYS}. In step four, the last *B* characters {SYS} have a shift value of three making the next window to scan {SYSDIR}. In step five, the last *B* characters {DIR} have a shift value of zero indicating a possible match. Therefore, check the HASH and PREFIX tables to verify the exact match and retrieve the pattern {SYSDIR}.

E. Bloom Filters Theory

Bloom filters were presented in 1970 by Burton Bloom [10]. The Bloom filter creates a hash vector representation of strings which can easily exclude negative matches. The filter preprocesses the strings by computing *k* hash values ranging from *1* to *m* for each string. The bits corresponding to the hash values computed for all strings are set in an *m-bit* long vector. To search a packet for the strings the filter is checked by computing the same *k* hash values on the packet in a sliding window fashion. Then, the corresponding bits of the vector are examined to determine whether the given item exists or not. If at least one bit is not set, it means that the item is not a member of the filter with 100% certainty, i.e. Bloom filters have no false negatives. On the other hand, if all the bits are set in the vector, then the item belongs to that string set with a certain probability. Exact pattern matching must be used to verify if the item actually belongs to the given set or not.

III. RELATED WORK

The literature is rich with research to accelerate signature-based IDS. The majority are hardware architectures and algorithms which are expensive, complex and suffer from cost and configurability issues. Configurability is a very important issue because of the constant need to add new signatures. In 2005 Haoyu Song et al. [11] proposed a new lookup algorithm which speeds up the HASH tables where they modified Bloom filters to provide exact matches. The Speedup comes from cutting the number of hash collisions and using a small external memory. The main disadvantage is the need for an expensive cache like on-chip memory. Sarang Dharmapurikar et al. [12] presented a new hardware pattern matching algorithm in 2006 using a combination of Bloom filters and classic AC algorithm. They implemented Bloom filters using embedded on-chip memory blocks in FPGA. Deepti Chaudhary et al. [13] proposed a new hardware parallel architecture in 2010 based on Bloom filters. It can test input strings simultaneously to detect possible attacks with less delay. It computes hash functions concurrently on all test strings and can AND bit location values in the look up array separately for different hashes of different strings.

On the other hand, software-based techniques are cheaper and easy to reconfigure. However, they are unable to match the increasing network speeds and they require a lot more memory. Kostas Anagnostakis et al. [14] proposed a new exclusion filter in 2003 for IDS called E^2xB. It is an exclusion-based filter that preprocesses the packet by using a 256 cell map to mark the existence of each character of patterns within the packet. If at least one character of a pattern is not marked, it means that this pattern does not exist in that packet. Zhongqiang Chen et al. [15] proposed a new system in 2009 to accelerate pattern matching for IDS by combining both fingerprinting and pattern matching techniques. In the programming phase of the fingerprinting, the system generates a short digest for each pattern. Then, it computes the digests of the incoming packets to match them against those of the patterns. BM algorithm is then used to perform exact pattern matching. Ramakrishnan Kandhan et al. [16] proposed an efficient and scalable system in 2010 called sigMatch to improve multi-pattern matching algorithms. It preprocesses all the patterns to organize them

TABLE I. WM SHIFT TABLE

Block	Shift	Block	Shift
mai	3	WIN	3
mag	2	IND	2
age	1	NDI	1
ge/	0	DIR	0
log	3	SYS	3
ogg	2	YSD	2
gge	1	SDI	1
ged	0	others	4

TABLE II. SNORT SIGNATURES EXAMPLE

Pattern	SID
image/	2706
logged in	162
imagedata	12280
WINDIR	3010
SYSDIR	3011

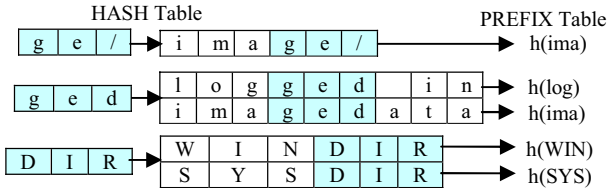

Figure 2. WM HASH and PREFIX tables

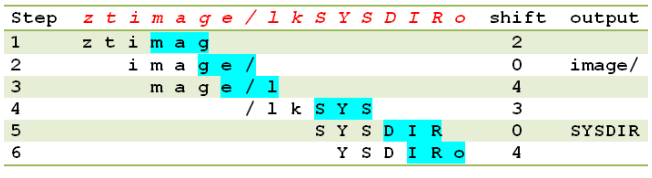

Figure 3. WM search

in a q-gram index structure called sigTree according to common sub-patterns. The sigTree combines the features of both faster tries and Bloom filters which consumes less memory. The idea is to match all the patterns against the sigTree to discard the unmatched ones and the remaining patterns are sent to a verification unit to check their presence.

There exist several modified versions of WM not necessarily for IDS. Yang Dong Hong et al. [17] introduced a Quick Search improved WM algorithm (QWM) in 2006. QWM adds a HEAD table which contains first two characters of the patterns to help determine if the first two characters in the matching window are the prefix of any pattern. Chen Zhen et al. [18] presented an improved WM (IWM) in 2008. IWM added a second SHIFT table to increase the possibility of shift at each comparison and to reduce the number of accesses to the HASH table. Baojun Zhang et al. [19] proposed an Address Filtering Based WM Multiple Patterns Matching Algorithm (AFWM) in 2009. It optimizes the access to the PREFIX table to accelerate searching the linked lists by sorting the patterns into ascending order according to the address pointers.

IV. EXSCIND

We propose Exscind: a new, fast and memory-efficient software-based matching system. It introduces an exclusion-inclusion filter that excludes clean traffic without the need to perform costly pattern matching. For that purpose a Bloom filter is modified to provide probable matches to further reduce the number of pattern matching operations required for suspicious packets. The filter is programmed with only the prefix (first 4-grams) of all Snort signatures in order to keep the filter processing overhead to minimum and speed up matching by skipping clean packets. The incoming packet is hashed and queried for those prefixes. If the query is negative then the packet is clean and can safely be skipped. If the query is positive then the packet probably contains an attack signature prefix and requires further matching. The filter produces a set of probable matches that are used by the probable match modified WM (PWM) to determine if there is a full signature. That is, the suspicious packets are searched for just a subset of signatures as opposed to all Snort signatures. In addition, the filter indicates the position within the packet where the first probable match was found, enabling the PWM to search part of the packet starting from that position as opposed to searching the whole packet. Because the majority of traffic is benign, Exscind accelerates pattern matching and improves the overall performance at the expense of adding minimal processing and memory overhead attributed to the filter.

Exscind works at packet level using one sliding window, one buffer for storing signature IDs and one signatures Bloom vector. The following Subsections illustrate by example the different stages of the algorithm.

A. Exscind Example

The first step is to initialize the modified Wu-Manber with all Snort signatures. Next, Exscind creates and programs the signatures Bloom vector with just the 4-gram prefixes of all Snort signatures. The programming works by computing two hash functions for each signature prefix and setting the corresponding bit in the vector. The signature identification numbers associated with those bits are stored in a buffer organized as a hash table. Fig. 4 explains how the programming of a 21-bit signatures Bloom vector is carried out for the first 4-grams of the following Snort signatures *{"dba_tables", "user_tablespace", "sys.all_users"}*, with SIDs {1687, 1688, 1689}. The figure shows the signatures with the corresponding hash bits to be set in addition to the Bloom vector after programming. For {*"dba_tables"*}, the prefix "*dba_*" is hashed with two functions to result in 5 and 16 which indicate the indices for the bits to be set in the Bloom filter. The signature ID, 1687, is recorded to be used later in generating the probable matches list.

For each incoming packet, Exscind starts a four character sliding window throughout the packet and computes the same two hash functions for each window. Fig. 5 shows how the sliding window moves through packet number 48897 of trace 1 of DEFCON17 traces [20]. For each hashed sliding window, Exscind checks the corresponding two hash bits in the signatures Bloom vector. If they are set, this means there is a probable match. Therefore, Exscind uses the signatures SIDs buffer to retrieve the associated signatures and marks them in the probable signatures list. The algorithm also records the index of the sliding windows that caused the match. On the other hand, if any of the two hash bits is not set, the sliding window continues until the end of the packet is reached. Fig. 6 shows how the bits are checked for the last sliding window "*user*".

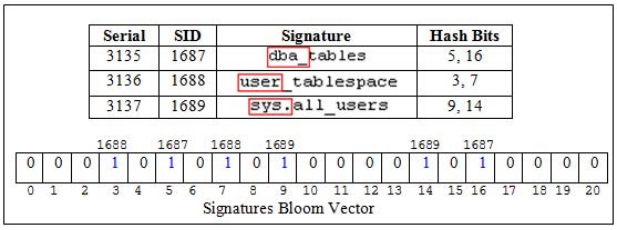

Figure 4. Signatures Bloom vector programming

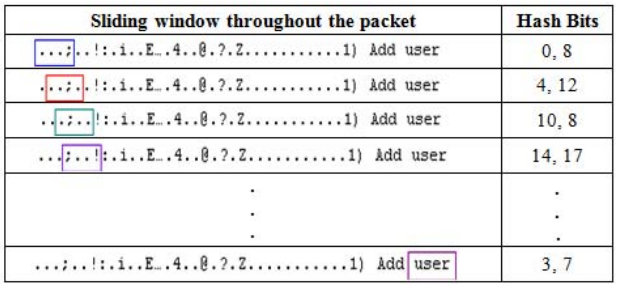

Figure 5. Packet sliding window

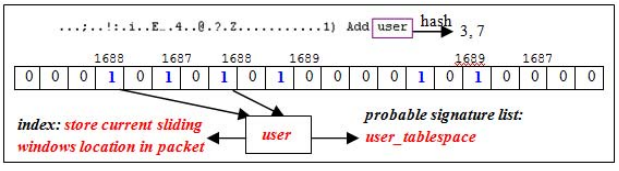

Figure 6. Bloom vector query

Finally, the algorithm checks the resulting probable signatures list. If it is empty, this means that the packet is clean and with 100% certainty it does not contain any signatures. Therefore, there is no need to perform any pattern matching on that packet. Otherwise, the packet is suspicious and may contain attack signatures. Therefore, Exscind uses the PWM to search the packet only for the probable matches starting from the index of the first probable match.

B. Probable Match Modified WM

PWM rewrites the original WM C++ code [21] using C#.NET in order to support all signature elements especially the HEX bytes between the pipes. PWM extends WM to support all special, control and non-printable ASCII characters as well as the NULL character. In addition, it searches the packet for just the highly probable signatures starting from a certain index provided by the filter. In the preprocessing phase the PWM is initialized once with all signatures. Only the search function is modified with the probable matches list as an argument. PWM associates each signature in the HASH table lists with a Boolean flag indicating whether that signature belongs to the probable matches or not. When the SHIFT table indicates a zero shift and the HASH table is to be searched, PWM checks that flag. If set, the WM computes the prefix hash for the packet substring, and then checks its equality with the signature prefix stored in the PREFIX table. If they are equal, exact character matching is performed between the signature and the packet substring until reaching the end of that signature. If the signature and the packet substring match, PWM declares the packet malicious and records the number of match occurrences along with the index of each match. Otherwise, if the flag is not set, this means that the signature does not belong to the probable matches. Therefore, it is skipped safely without performing any matching and the next string in the HASH table list is checked. In addition, PWM supports searching the packet starting from the position of the first probable match instead of searching the whole packet. This is done by using the index of the sliding window which caused that match provided by the filter.

V. Evaluation and Analysis

Experimental time and memory measurements are carried out using actual traffic traces. The experiments are designed to compare Exscind with the regular Wu-Manber algorithm used in Snort. The simulations are performed on a PC with a 3 GHz Intel Core 2 processor, with 4 GB of main memory running Microsoft Windows 7. Both packets and signatures are extracted and read offline from local files. Each experiment is performed 20 times and then average time and memory are scored. To measure the execution time, we use *ExecutionStopWatch* class [22] and to measure the memory usage, we use the *Garbage Collection* class [23].

A. Traffic Analysis

To evaluate Exscind, we use both normal and malicious traffic traces classified into good, bad and ugly. The classification is based on the maliciousness of packets measured by the percentage of Snort signatures found in the trace. The good traces are from the SourceForge.net publically available PCAP files which list packet capture repositories [24]. The four traces shown in the lower section of Table III, with percentage of malicious content ranging from 1 to 4% represent the good traces. Fragmented packets were ignored during the analysis.The good traces include:

1) SIP_Eyebeam2Eyebeam_Video_Audio (VA) is a video and audio streaming from TechTraces repository [25].

2) Good-Download (GD) is a file download from Laura's Lab Kit v.8 repository [26].

3) Live-Chat (LC) is a live chat application from [26].

4) WebHotmail (HM) is a mail trace from TkuIM mail repository [27].

For the bad and ugly traces, we use packet traces from Capture the Flag (CTF) game held at DEFCON17 hacker conference released in August of 2009 [19]. CTF is a virtual warfare between the best hackers in the world to capture each other's machines. The highly malicious traces are collected and made available publically for research purposes. The bad and ugly traces analysis is shown in the upper section of Table III. Out of total of 78 CTF traces analyzed, we only use eight traces, those with the highest and lowest percentages of malicious packets. Traces number 1, 22, 0 and 2 have the least percentage of malicious packets, ranging between 15.9% and 20.4%, and represent the bad traces. Traces number 8, 57, 51 and 58 have the highest percentage of malicious packets, ranging between 38.5% and 44.6%, and represent the ugly traces. It is also important to mention that a packet might contain more than one match. For example although trace 58 has 119,627 malicious packets out of 268,000 in total, but the same trace has 3,107,078 attack signature matches averaging almost 26 matches per malicious packet.

B. Traffic and Signatures Extraction

We use Wireshark Network Protocol Analyzer v1.4.4 to open and extract the packets traces [28]. A C# utility reads the trace files in binary, extracts the packet payloads in HEX and writes each packet contents on a separate line.

For the signatures, we use Snort 2.9.0.4 rules database released in March of 2011 which contains 56 rule files [9]. Exscind extracts the SID, content and uricontent parts from all Snort rules. Multiple content and/or uricontent parts of the same rule are concatenated with zero space to form one signature. The total number of extracted signatures is 9,945.

C. Execution Time and Speedup

Table III shows the execution time of Exscind compared to the regular WM for the good, bad and ugly traces using four characters prefix in the filter. The ugly traces represent the worst case scenario while the good traces represent the best or normal scenario. The table shows significant speedup of 3.4 on average for all traces. The worst case speedup for ugly traces is between 1.5 to 1.7 times, with an average of 1.61 times. The speedup for the bad traces ranges between 2.9 and 3.3 times, with average of 3.17 times, while the speedup for the good traces ranges between 5 and 6.2 times, with an average of 5.5 times.

It is noticeable that the longest execution time belongs to the bad and not the ugly traces as one might expect. The time depends on the total number of packets, and in a great degree, on the number of actual signatures found. On average the bad traffic has more packets than the ugly making their execution time longer. In addition, the more signatures matched the more the filter will produce a sizable probable matches list and the more pattern matching the algorithm will have to execute. The bad traces have on average 31,874,200 signature matches while the ugly traces have only 5,820,262 signature matches. The large number of matches simply translates into larger execution time. We chose the ugly traces based on the percentage of malicious packets which is indicative of how many packets we can skip using the filter. Because we work at the packet level and ugly traces have more malicious packets, we report lower speedup for the ugly traces as opposed to the bad traces despite the fact that bad traces have more matches. The speedup is best represented by Fig. 7 with the lowest value of 1.53 times in the case of the most malicious trace and highest of 6.18 times in the case of the good VA trace.

It is difficult to compare to the related work due to the use of different datasets and testing environments. QWM used the Bible and Chinese text for instance. But comparing the speedup numbers as opposed to the original WM would give us a taste. For 2000 signatures QWM, AFWM and IWM reported speedups of 1.21, 1.55 and 1.67 respectively. The average speedup for Exscind for 2000 signatures was 2.98. The best speed up was 5.43 and the worst was 1.26.

D. Memory Usage

To evaluate the memory overhead added by the exclusion-inclusion filter, we measure the total memory consumption of both Exscind and WM for the ugly traces. The total memory consumed by WM is 537,178 KBs and by Exscind is 537,766.1 KBs with an increase of only 588.1 KBs representing 0.1095% worst-case overhead. That is a very small price to pay for an average speedup of 3.4. The overhead is attributed to the signatures Bloom vector, the SIDs buffer and the probable matched signatures list.

E. Performance Scaling

To evaluate how Exscind performance scales with increasing workload, we vary the number of signatures while measuring both the execution time and memory usage. Fig. 8 shows Exscind speedup for the good, bad and ugly traces against increasing number of signatures. The signatures are varied between 1000 and 9000, with 1000 intervals. In order to do that, we randomly divided Snort signatures into groups of 1000 and conducted the experiments while adding 1000 signatures each time. The chart shows that the smallest speedup is achieved when we have the smallest number of signatures that is 1000. This is expected, because after careful inspection the first group of 1000 signatures, it turned out that they are short and averaging 4.5 characters in length while longer strings appear more in the signatures set as we move toward the full set of 9000 signatures. The short signatures are easily found in packets and therefore the filter is not able to skip that many packets. On the other hand, the longer strings penalize the WM which has to perform matching every time while Exscind benefits more from skipping such long strings. Therefore, WM runtime increases while Exscind time gets shorter which explains the slight increase in speedup as the number of signatures increase.The important conclusion out this experiment is that the speed up achieved by the algorithm is nearly independent from the number of strings if they were completely even in size. This is a great performance scaling compared to decreasing performance, exponentially in AC and linearly in WM. Fig. 9 shows Exscind worst case memory usage measured for the

TABLE III. EXECUTION TIME FOR THE GOOD, BAD AND UGLY TRACES

Ugly Trace	Total Packets	Malicious Packets	Malicious Packets (%)	WM Time	Exscind Time	Speed up
8	671116	258556	38.5	25.74	15.09	1.71
57	209188	86129	41.2	15.29	9.53	1.60
51	299713	129763	43.3	15.02	9.50	1.58
58	268000	119627	44.6	11.37	7.42	1.53
Bad Trace	**Total Packets**	**Malicious Packets**	**Malicious Packets (%)**	**WM Time**	**Exscind Time**	**Speed up**
1	688158	109311	15.9	75.44	23.00	3.28
22	659365	110363	16.7	76.22	22.90	3.33
0	771382	139866	18.1	66.18	20.87	3.17
2	642091	130901	20.4	55.04	19.11	2.88
Good Trace	**Total Packets**	**Malicious Packets**	**Malicious Packets (%)**	**WM Time**	**Exscind Time**	**Speed up**
VA	3606	54	1	105	17	6.18
GD	9354	175	2	345	63	5.48
LC	29474	1031	3	698	130	5.37
HM	2817	126	4	90	18	5.00

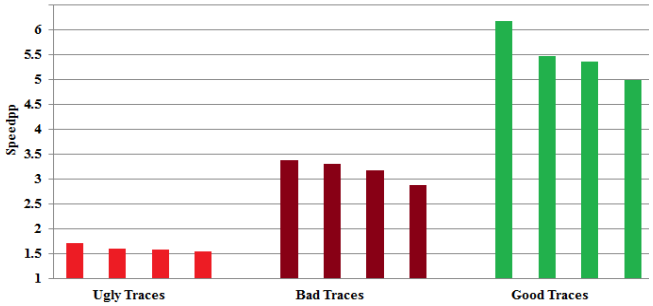

Figure 7. Speedup for good, bad and ugly traces

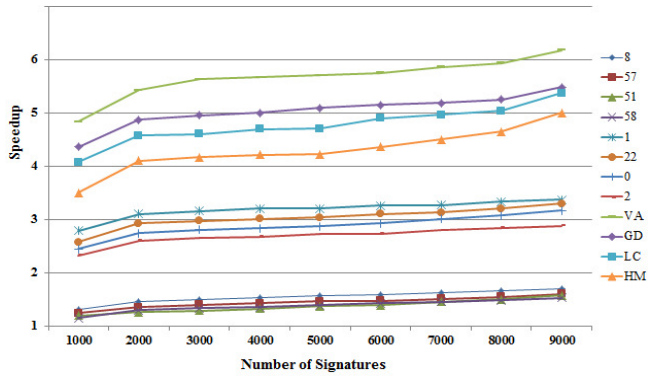

Figure 8. Speedup versus number of signatures

TABLE IV. MEMORY USAGE FOR A PREFIX OF 4 CHARACTERS IN KBS

WM Memory	Our Memory	Added Overhead	Percentage overhead (%)
537,178	537,766.1	588.1	1.095

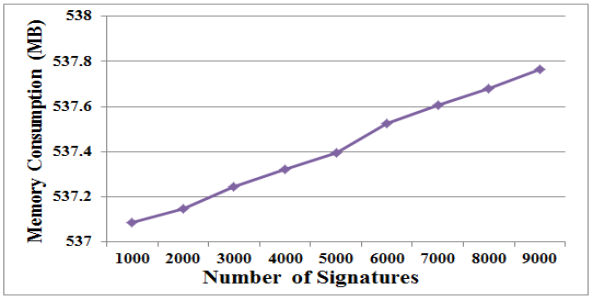

Figure 9. Meomory usage versus number of signatures

ugly traces versus increasing number of signatures. The memory usage increases linearly with the number of signatures. This is typical of WM based algorithms as opposed to exponential in AC.

VI. CONCLUSIONS

We introduce Exscind: a new and fast software-based pattern matching algorithm to accelerate signature-based IDS. It excludes clean traffic without the need to do pattern matching by using a lightweight exclusion-inclusion filter programmed only with the signatures prefix. In addition, the filter includes only a subset of the signatures in the search process performed by a specially modified WM algorithm. The filter establishes the PWM algorithm with a starting position for the first probable match as well as a small list of probable matches. Exscind is thoroughly evaluated and compared to the state of the art. Exscind achieves an average speedup of 3.4 times and a normal traffic speedup of more than 6 times. The overhead incurred is limited to 0.11% increase in memory usage. The algorithm scales very well with increasing number of signatures. The speedup is almost constant and undiminished with increasing number of signatures while memory usage increases linearly. The filter is easily reconfigured with new signatures to support emerging attacks.

REFERENCES

[1] R. Proudfoot, K. Kent, E. Aubanel, N. Chen, “High performance software-hardware network intrusion detection system. In Proceedings of the International Conference Field-Programmable Technology (ICFPT), 2007:309-312.

[2] M. Aldwairi, T. Conte, P. Franzon, “Configurable string matching hardware for speeding up intrusion detection. In ACM SIGARCH Computer Architecture News 2005, 33(1):99-107.

[3] D. Alansari, “Fast pattern matching for intrusion detection using exclusion and inclusion filters”, MS thesis, Jordan University of Science and Technology, Irbid, Jordan, 2011, Print.

[4] L. Roberts, “Internet growth trends”, In Internet Watch column of IEEE Computer Magazine, January 2000.

[5] R. Rehman, “Intrusion detection with SNORT: advanced IDS techniques using SNORT, Apache, MySQL, PHP, and ACID”, Pearson Education Inc., Prentice Hall, 2003:75-129.

[6] R. Boyer and J. Moore, "A fast string searching algorithm". In Communications of the ACM vol. 20 no. 10, pp762–772, 1977, doi:10.1145/359842.359859.

[7] A. Aho, M. Corasick, "Efficient string matching: an aid to bibliographic search", in the Communications of the ACM vol. 18, no 6, pp333–340, June 1975. doi:10.1145/360825.360855,

[8] S. Wu and U. Manber, “A fast algorithm for multi-pattern searching”, Technical Report TR94-17, University of Arizona, 1994.

[9] Snort rules, http://www.snort.org/snort-rules/, last access in April, 2011.

[10] B. Bloom, “Space/time trade-offs in hash coding with allowable errors”, In the Communications of the ACM 1970, 13(7):422-426.

[11] H. Song, S. Dharmapurikar, J. Turner and J. Lockwood, “Fast hash table lookup using extended Bloom filter: an aid to network processing”, In SIGCOMM Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications 2005, 35(4):181-192.

[12] S. Dharmapurikar and J. Lockwood, “Fast and scalable pattern matching for network intrusion detection systems”, In Selected Areas in Communications IEEE, 2006, 24(10):1781-1792.

[13] D. Chaudhary, “Parallel processing of Bloom filter”, In International Journal of Electronic Engineering Research 2010, 2(1):35-40.

[14] K. Anagnostakis, S. Antonatos, E. Markatos and M. Polychronakis, “E2xB: A domain-specific string matching algorithm for intrusion detection”, In the Proceedings 18th IFIP International Information Security Conference (SEC) 2003:217-228.

[15] Z. Chen, Y. Zhang, Z. Chen and A. Delis, ”A digest and pattern matching-based intrusion detection engine”, In the Computer Journal 2009, 52(6):699-723.

[16] K. Ramakrishnan, T. Nikhil and M. Jignesh, ”SigMatch: fast and scalable multi-pattern matching”, In 36th International Conference on Very Large Data Bases (PVLDB) 2010, 3(1):1173-1184.

[17] H. Yang, K. Xu and Y. Cui, "An improved Wu-Manber multiple patterns matching algorithm," In 25th IEEE International Performance, Computing, and Communications Conference (IPCCC), 2006: 680-688.

[18] Z. Chen and D. Wu, "Improving Wu-Manber: a multi-pattern matching algorithm", In IEEE International Conference on Networking, Sensing and Control (ICNSC), 2008: 812-817.

[19] Z. Baojun,C. Xiaoping, P Lingdi and W. Zhaohui, “Address Filtering Based Wu-Manber Multiple Patterns Matching Algorithm”, In Proceedings of the 2009 Second International Workshop on Computer Science and Engineering (IWCSE), IEEE Computer Society Washington 2009, (1):408-412.

[20] DEFCON17 traffic, Last access in Apr, 2011, http://www.defcon.org.

[21] One unified, last access in Sept, 2010, http://www.oneunified.net/blog/2008/03/23/.

[22] CodeProject ExecutionStopwatch, last access in February, 2011, http://www.codeproject.com/KB/dotnet/ExecutionStopwatch.aspx.

[23] MSDN Microsoft GC, last access in February, 2011, http://msdn.microsoft.com/en-us/library/system.gc.aspx.

[24] SourceForge publically availabe PCAP files, last access in April, 2011,http://sourceforge.net/apps/mediawiki/networkminer/index.php?title=Publicly_available_PCAP_files.

[25] TechTraces sample captures, last access in April, 2011, http://techtraces.com/sample_captures/.

[26] Laura's Lab Kit v.8, last access in April, 2011, http://demeter.uni-regensburg.de/Lauras_Lab_Kit_v8/AutoPlay/trace_files_llk8/.

[27] TkuIM mail, last access in April, 2011, http://mail.im.tku.edu.tw/~miller.lai/pcap/pcapList.php.

[28] Wireshark v1.4.4, last access in February, 2011, http://www.wireshark.org/download.htm.