Reducing HTTP Collapse with Usage of Pattern Methods

Abdul Mannan Virani1, Somesh Kumar Dewangan2
1M.Tech, 2M.Tech Associate Professor, CSE Dept, DIMAT, Chhattisgarh

Abstract--- Security is the major concern in today’s scenario a vigorous security mechanism is needed to secure the network from unwanted traffic. There are lots of technologies developed to provide a secure currently one of them is by utilizing “signature-based” detection can’t bare stressed traffic, where marketing value is incrementing. Signature predicated detection which is being used current security implements do not handle compressed traffic. GZIP compression is the scheme used to Compress HTTP traffic which requires decompression afore matching. This paper developed on the theme of compressed HTTP traffic. HTTP uses GZIP compression. Decompression phase is utilized by HTTP afore performing a string matching. In this a algorithm, Aho–Corasick-predicated algorithm for Compressed HTTP (ACCH) was habituated to provide more advantage than the commonly used Aho–Corasick pattern-matching algorithm. The advantage of this is that capitalizes on information amassed by the decompression phase in order to expedite, we explore that it is more expeditious to perform pattern matching on the compressed data, with the defect of decompression than on conventional traffic. We are the initial one that analyzes the quandary of “on-the-fly” multipattern matching on compressed HTTP traffic and solves it.

Keywords-- GZIP, Aho-Corasick (ACCH) algorithm, pattern-matching,LZ77 compression.

I. INTRODUCTION

Technologies for security, such as Network Intrusion Detection System (NIDS) or Web Application Firewall (WAF). This deals with signature-predicated detection techniques to identify attacks. Now a days, security implements is judged by the speed of the underlying string-matching algorithms that detect these signatures .HTTP compression nothing but content encoding is openly available method to compress textual content transferred from Web servers to browsers. Lots of websites and convivial sites are utilizing this HTTP compression. As per research 25% + industries are utilizing HTTP compression, and incrementing. This compressed content is built into HTTP 1.1 and was fortified by most browsers.

Most current security implements either ignore scanning compressed traffic, which causes of security apertures, or incapacitate the option for compressed traffic by re-engendering the “client-to” HTTPHeader to shows that compression is not backed by the client’s browser thus decays the consummate performance and bandwidth.

Less security implements HTTP compressed traffic by decompressing the entire page on the proxy and doing a signature scan on the decompressed page afore passing to the client. This option doesn’t applicable for security implements that performs at a high speed or when performing adscititious delay is not an option. In this paper, we explore a novel algorithm, Aho–Corasick predicated algorithm on Compressed HTTP (ACCH).

ACCH decompresses the traffic and then utilizes the data from the decompression phase to expedite the pattern matching. Categorically, the GZIP compression algorithm works by evading repetitions of strings utilizing back-references (pointers) to the reiterated strings. Our key insight is to store information engendered by the pattern-matching algorithm for the already-scanned decompressed traffic, and if any case of a pointer, utilize this information to get if it contains a match or one can securely expunges scanning bytes in it. ACCH can skip up to 84% of the data and boost the performance of the multi pattern-matching algorithm by up to 74%.

1.1 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 Automatic 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).
II. PROPOSED SYSTEM

1. Packet Capturing
   - The module opens the network interface card
   - Reads every packet that are received by the Network Interface Card (NIC).
   - Queues all the packets in buffer

2. Application Payload Extraction
   - The buffered packets will be in raw packet format.
   - The Module identifies the headers and payloads at each layer of TCP/IP.
   - Decodes the payload according to their header formats.
   - The Application Payload will be buffered or stored for the next level.

3. HTTP Encoding Header Identification
   - If the packet payload is in the HTTP Protocol format, the module checks for the HTTP Header
   - Accept-Encoding: gzip or
   - Accept-Encoding: deflate
   - Accept-Encoding: chunked Etc…. 
   - Presence of Accept-Encoding HTTP Header confirms that the payload is in the encoding format.
   - If the Header is “Accept-Encoding: gzip” then the payload is in the compressed format.

4. Decompression of Payload
   - If the payload is in the gzip compression, the payloads has to be buffered for all the incoming packets
   - The buffered payload is processed for gzip decompression.
   - The decompressed data is passed on to the Pattern Matching for attack identification.

5. Alert Verification in SNORT
   - The module confirms that, the signatures are not hitting for the compressed data and hitting for the decompressed data.

*Multi pattern Matching:*

Pattern matching has been a topic of intensive research resulting in several approaches; the two fundamental approaches are based on the Aho–Corasick (AC) and the Boyer–Moore algorithms. In this paper, we illustrate our technique using the AC algorithm. The basic AC algorithm constructs a deterministic finite automaton (DFA) for detecting all occurrences of given patterns by processing the input in a single pass. The input is inspected symbol by symbol (usually each symbol is a byte), such that each symbol results in a state transition. Thus, the AC algorithm has deterministic performance, which does not depend on the input, and therefore is not vulnerable to various attacks, making it very attractive to NIDS systems. Each arrow indicates a DFA transition made by a single byte scan.

![Aho–Corasick DFA for patterns matching](image)

The label of the destination state indicates the scanned byte. If there is no adequate destination state for the scanned byte, the next state is set to root. For readability, transitions to root were omitted. Note that this common encoding requires a large matrix of size, where is the set of ASCII symbols and is the number of states) with one entry per DFA edge. In the typical case, the number of edges, and thus the number of entries, is $256^s$. 

---

- Should Provide pattern matching with Assertions (back References, look-ahead, look-back, and Conditional sub-patterns).
- Should use less memory (Space complexity is low).
Challenges faced in multi pattern matching

1) Remove the HTTP header and store the Huffman dictionary of the specific session in memory. Note that different HTTP sessions would have different Huffman dictionaries.
2) Decode the Huffman mapping of each symbol to the original byte or pointer representation using the specific Huffman dictionary table.
3) Decode the LZ77 part.
4) Perform multi-pattern matching on the decompressed traffic.

Space: One of the problems of decompression is its memory requirement. The straightforward approach requires 32KB sliding window for each HTTP session. Note that this requirement is difficult to avoid since the back-reference pointer can refer to any point within the sliding window and the pointers may be recursive unlimitedly (i.e., pointer may point to area with a pointer). Indeed the distribution of pointers on the real-life data set (see Section VII for details on the data set) is spread across the entire window. On the other hand, pattern matching of non-compressed traffic requires storing only one or two packets (to handle cross-packet data), where the maximum size of a TCP packet is 1.5KB. Hence, dealing with compressed traffic poses a higher memory requirement by a factor of 10. Thus, mid-range firewall, which handles 30K concurrent sessions, requires 1GB memory, while a high-end firewall with 300K concurrent sessions requires 10GB.

Time: Recall that pattern matching is a dominant factor in the performance of security tools, while performing decompression further increases the overall time penalty. Therefore, security tools tend to ignore compressed traffic. This paper focuses on reducing the time requirement by using the information gathered by the compression phase. We note that pattern matching with the AC algorithm requires significantly more time than decompression since decompression is based on consecutive memory reading from the sliding window, hence it has low read-per-byte cost. On the other hand, the AC algorithm employs a very large DFA that is accessed with random memory reads, which typically does not fit in cache, thus requiring main memory accesses. Appendix A introduces a model that compares the time requirements of the decompression and the AC algorithm.

Experiments on real data show that decompression takes only a negligible 3.5% of the time it takes to run the AC algorithm. For that reason, we focus on improving AC performance. We show that we can reduce the AC time by skipping more than 70% of the DFA scans and hence reduce the total time requirement for handling pattern matching in compressed traffic by more than 60%.

2.1 Proposed Algorithm

Algorithm Native Decompression with Aho Corasick pattern matching

$Trf$ – the input, compressed traffic (after Huffman decompression)

$SWin_n$ – the sliding window of LZ77, where $SWin_i$ is the information about the uncompressed byte which is located $j$ bytes before current byte

$FSM(state, byte)$ – AC FSM receives state and byte and returns the next state, where $startStateFSM$ is the initial FSM state

$Match(state)$ – if state is “match state” it stores information about the matched pattern, otherwise NULL

1: state = function scanAC(state)
2: state = FSM (state, byte)
3: if $Match (state)$ / NULL then
4:    act according to $Match (state)$
5: end if
6: return state
7: procedure ZIPDecompressPlusAC($Trf_1, \ldots, Trf_n$)
8: state = $startStateFSM$
\begin{verbatim}
9: for i =1 to n do
10:   if Trf \_i is pointer (dist, len) then
11:     for j =0 to length -1 do
12:       state scanAC (state, SWin_{distj})
13:     end for
14:     update SWin with bytes SWin_{dist_{distlen}}
15:   else
16:     state=scanAC(state, Trf \_i)
17:     update SWin with the byte Trf \_i
18:   end if
19:end
\end{verbatim}

III. RESULTS AND DISCUSSIONS

In this section, we evaluate the performance benefit of ACCH algorithm and find the optimal CDepth, a key parameter of our algorithm, using real-life traffic.

A. Data Set

We use two data sets, one of the traffic and the other of the patterns. For the traffic, we use two data sources: traffic that was captured by a corporate firewall for 15 min and a list of the most popular Web pages taken from the Alexa Web site, which maintains Web traffic metrics and top sites lists. Table I shows the characteristics of both traces. No significant difference was found between the results over those two data sets, therefore we refer to the Alexa data-set parameters. We use two signature data sets: one of ModSecurity an open-source Web application firewall, and the other of Snort , an open-source network intrusion prevention and detection system.

B. ACCH Performance

In this section, we compare the performance benefit using the ACCH algorithm. We define as the scanned character ratio, is a dominant factor for performance improvement for ACCH over the Naive algorithm as shown later. The tradeoff of choosing different values for, as discussed in previous Section , can be shown in the graph. Snort patterns result in a significant amount of matches within the traffic (near 2% of the total number of bytes are marked with), resulting in more Match Segments, which in turn cause a significantly higher. In order to check the influence of the amount of matches on , we synthesized a reduced Snort pattern-set by removing 88 of the most frequent patterns. The data contain 234 448 patterns from the reduced pattern-set (instead of 14 M). Thus, match ratio has stronger influence on value than the number of patterns.

\begin{table}[h]
\centering
\caption{Parameters Analyzed Using Real-Life Data For Snort And Modsecurity.}
\begin{tabular}{|c|c|c|}
\hline
Description & Snort & ModSecurity \\
\hline $A_l$ & 1.007 & 0.684 \\
\hline $A_r$ & 1.61 & 1.159 \\
\hline $A_m$ & 2.55 & 4.824 \\
\hline $A_c$ & 0.017 & $7.73 \times 10^{-7}$ \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{Normalized Throughput Of Acch (With Optimizations) As Compared To The Two-Dimensional-Array Dfa Implementation Using Snort Pattern-Set}
\begin{tabular}{|c|c|c|c|}
\hline
& Memory & Plain & ACCH \\
\hline Two-Dimensional Array & 74.49 MB & 1 & 0.3 \\
\hline Sparse & 1.79 MB & 0.99 & 0.29 \\
\hline Banded & 2.02 MB & 0.97 & 0.28 \\
\hline Sparse-Banded & 1.89 MB & 0.98 & 0.28 \\
\hline
\end{tabular}
\end{table}

IV. CONCLUSION

Now a day’s almost each and every one modern security tool is a pattern matching algorithm. Web traffic is completely based on HTTP compression. Normally security tools ignore this traffic and leave security holes. In another case it neglects the parameters of the connection, it leads dangerous situation to the performance and bandwidth of client side and server side .Our algorithm eliminates up to 84% of data scan based on information stored in the compressed data. Unexpectedly, it is faster to perform pattern matching on compressed data with the effect of e compression, rather than pattern matching on uncompressed traffic. We have to observe that ACCH is not intrusive for the AC algorithm, so all the methods that improve AC DFA are orthogonal to ACCH and are applicable. We are the first paper that analyzes the problem of “on-the-fly” multi pattern-matching algorithm on compressed HTTP traffic and suggests a solution.
REFERENCES


Authors Profile

First Author: Abdul Mannan Virani
received his B.E. (CSE) degree from RCET Bhilai, Pandit Ravi Shankar Shukla University (Pt.RSU) Raipur, in 2006. From 2006 to 2010 he worked in various multinational companies as Consultant and Customer support. He is currently an M.Tech student in the Computer Science Engineering from DIMAT Raipur, Chhattisgarh Swami Vivekananda University Bhilai. His Research interests are in the areas of Wireless and Network Security, with current focus on secure data services in cloud computing and secure computation outsourcing.

Second Author: Somesh Kumar Dewangan
received his M. Tech in Computer Science and Engineering from RCET Bhilai, Chhattisgarh Swami Vivekananda University Bhilai, in 2009. Before that the MCA Degree in Computer Application from MPBO University, Bhopal, India, in 2005. He is lecturer, Assistant Professor, Associate professor, Disha Institute of Management and Technology, Chhattisgarh Swami Vivekananda Technical University Bhilai, India, in 2005 and 2008 respectively. His research interests include digital signal processing and image processing, Natural Language Processing, Neural Network, Artificial Intelligence, Information and Network Security, Mobile Networking and Cryptography and Android based Application.