May 25, 2018

Detecting Evasive Malicious URL Using Graph Algorithm

Mu Lin

Sanjay d'Abreu Noronha

Kan Yan

Zhifeng Cai

Kevin Hayes

DETECTING EVASIVE MALICIOUS URL USING GRAPH ALGORITHM

Introduction

The present disclosure provides systems and methods to enable inspecting of byte streams using a bipartite match algorithm to detect the use of evasive techniques to defeat Uniform Resource Locator (URL) filtering or filtering of other Uniform Resource Identifiers (URI). Generally, URL filtering is one of the primary methods used to detect cyber threats. However, the URL specification (e.g., as set forth in RFC 3986) is quite flexible and can allow malicious actors to use evasive techniques to defeat algorithms used in URL filtering technologies. The conventional way to inspect byte streams, in particular for web traffic, often uses automata (e.g., Deterministic Finite Automaton (DFA), Nondeterministic Finite Automaton (NFA)) or regular expression (regex)). These methods generally work well when the regex is relatively "simple," but when evasive techniques are used, automata is less effective. The systems and methods of the present disclosure can provide for inspecting byte streams using a bipartite match algorithm. Effectively, the token can be kept simple, and the complexity of the pattern (formed using the token) can be expressed using bipartite match terminology.

Summary

According to an aspect of the present disclosure, a two stage dynamic programming algorithm is provided to enable inspecting of byte streams to detect use of evasive techniques to defeat Uniform Resource (URL) filtering. This algorithm matches signatures once and only once just like DFA in the first stage, and uses polynomial runtime to select a matching pattern in the second stage.

Detailed Description

The systems and methods of the present disclosure can provide for inspecting byte streams using a bipartite match algorithm. According to an aspect of the present disclosure, the token can be kept simple, and the complexity of the pattern (formed using the token) can be expressed using bipartite match terminology. The pattern is a sequence of tokens. (Note that a token can match to multiple signatures.)

The tokens in the pattern are adjacent and their order of appearance is significant, these two factors are implicitly inconvenient for describing collections of signatures where order or continuity (e.g., a pattern "abc" where a, b, and c are tokens and there are no other tokens in the pattern) or both are not the matching criteria. The above can be difficult and/or expensive to execute, even not possible, in DFA, as well as NFA when compounded with signature polymorphism. From the first principle, this is a combinatorial pattern matching problem.

According to an aspect of the present disclosure, a two stage dynamic programming algorithm and related data structure is provided as an alternative to DFA. This algorithm matches signatures once and only once just like DFA in the first stage, and uses polynomial runtime to select a matching pattern in the second stage.

According to an example implementation of the present disclosure, a two stage dynamic programming algorithm comprises the following operations. Conceptually, in compile time, the patterns are compiled into an $n*m$ binary matrix plus one additional column to describe the modifiers, where n is the number of patterns and m is the number of unique tokens used in the patterns. The tokens are compiled into DFA as usual (e.g., partitioned per URL host + path using a trie data structure) such that $e(i,j)=1$ if and only if the ith pattern has the jth token. In the first stage of runtime, first an $n*m$ binary matrix is built where n is the number of tokens

matched against the DFA (e.g., tokens are obtained from protocol/URL parsing phase) and *m* is the number of tokens such that *e*(*i*,*j*)=true if and only if the *i*th token matches the *j*th token. Note that the *i*th row can have multiple columns set to true. In the second stage of runtime, the sub runtime *n***m* matrix is obtained for a particular pattern such that *e*(*i*,*j*)=true if and only if the *i*th token matches the *j*th tokens in the pattern. A bipartite graph is equivalent to a matrix, therefore, a bipartite graph can be built where there are two sets of vertex, *n* and *m*, where *n* is token and *m* is signature. There is a weight 1 edge between an *i*th vertex in *n* and an *n*th vertex in *m* if *e*(*i*,*j*) in the matrix is true.

The necessary condition for a pattern match is the bipartite graph's maximum bipartite matching or max flow equals to *m*, the number of tokens in the pattern. This can be proven using the definition of maximum bipartite matching/max flow, which will be called Raven Pattern Matching Theorem L herein. It can be further proven that Theorem L is not only the necessary but also the sufficient condition for patterns who have both orderfree and continuityfree modifiers.

When patterns have only the orderfree modifier, there is a need to additionally check that it is continuous max flow. The FordFulkerson algorithm (FFA) and its variants can be used to determine the max flow number in O(Ef).

Function FFA

for each edge (u,v) in E(G)

do f[u, v] = 0 f[v, u] = 0 while there is a path p from s to t in the residual network Gf

do m = min{c(u, v)-f[u, v]: (u, v) is on p} for each edge (u, v) on p

do f[u, v] = f[u, v] + m f[v, u] = - f[u, v]

FFA can be overkill since it deals with general graphs, whereas the HopcroftKarp algorithm (HKA) is an example of a max flow algorithm for bipartite graphs. It runs in O(E*sqrt(V)), and for random graphs, it runs in near linear time. For sparse graphs, HKA can provide better results in worst case performance.

Function HKA

for each u in U

Pair_U[u] = NIL for each v in V

Pair_V[v] = NIL matching = 0 while BFS() == true

for each u in U

if Pair_U[u] == NIL

if DFS(u) == true

matching = matching + 1

return matching

It can be proven that if and only if the diagonals of the sub square *m*m* matrix are all true, then it matches a pattern who has neither orderfree nor continuityfree. A naive algorithm can determine the above in O(nm). A KnuthMorrisPratt style algorithm can be used to speed this up.

Finally, it can be proven that the runtime is O(n) for patterns that have only continuityfree.

All four combinations of orderfree and continuityfree therefore have polynomial runtime complexity or better.

In conclusion, just like DFA is the mathematical model for regex, bipartite graph is a good mathematical model to represent pattern.

For combinatorial analysis, the problem space is roughly P(k, n) * C(k, m) for k = 1…n O(Ef) where E is the number of edges, f is max flow. When orderfree and continuityfree are both specified in the pattern, it is better to specify the max number of parameters in order to have the algorithm be bounded, the same for matching the signatures in the first stage.

Figure 1 depicts an example system 100 according to an implementation of the present disclosure. Figure 1 illustrates one example computing system that can be used to implement the present disclosure. Other computing systems can be used as well. The system 100 may comprise one or more user computing devices, such as user computing device 102, one or more firewalls, such as firewall 130, one or more filtering server computing systems, such as filtering server computing system 140, and one or more web servers, such as web server(s) 160, coupled over one or more networks, such as network 180.

The user computing device 102 can include one or more processors 104 and a memory 106. The one or more processors 104 can be any suitable processing device and can be one processor or a plurality of processors that are operatively connected. The memory 106 can include one or more non-transitory computer-readable storage mediums, such as RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. The memory 106 can store data 108 and instructions 110 which are executed by the processor 104 to cause the first computing device 102 to perform operations.

The user computing device 102 can also include one or more input/output interface(s) 116. One or more input/output interface(s) 116 can include, for example, devices for receiving information from or providing information to a user, such as a display device, touch screen, touch pad, mouse, data entry keys, an audio output device such as one or more speakers, a microphone, haptic feedback device, etc. The user computing device 102 can also include one or

more communication/network interface(s) 118 used to communicate with one or more systems or devices, including systems or devices that are remotely located from the user computing device 102.

According to an aspect of the present disclosure, the user computing device 102 can send requests to and receive responses from one or more web servers, such as web server(s) 160. The requests and responses can be processed through one or more firewalls, such as firewall 130, for example, to protect the user computing device 102 from malicious actors. The firewall 130 can communicate with filtering server computing system 140 to perform URL filtering as discussed herein.

The filtering server computing device 140 can include one or more processors 142 and a memory 144. The one or more processors 142 can be any suitable processing device and can be one processor or a plurality of processors that are operatively connected. The memory 144 can include one or more non-transitory computer-readable storage mediums, such as RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. The memory 144 can store data 146 and instructions 148 which are executed by the processor 142 to cause the filtering server computing device 140 to perform operations, for example, to implement operations as discussed herein. The filtering server computing device 140 may include one or more URL filtering systems 150 that can assist in identifying and/or filtering invalid and/or malicious URL requests and/or responses. The URL filtering systems 150 can include a bipartite matching subsystem 152 which can provide operations for malicious URL filtering as discussed herein.

Figure 2 depicts a flowchart illustrating example operations 200 for inspecting of byte streams using a bipartite match algorithm in accordance with aspects of the present disclosure.

Although operations 200 are shown and described in a particular order for purposes of illustration and discussion, the operations are not limited to the particularly illustrated order or arrangement and certain operations can be performed in different orders or simultaneously.

The operations begin at block 202 where patterns are compiled into an $n*m$ binary matrix plus one additional column to describe the modifiers, where n is the number of patterns and m is the number of unique tokens used in the patterns.

At block 204, the signatures are compiled into DFA as usual (e.g., partitioned per URL host + path using a trie data structure) such that $e(i,j)$=1 if and only if the ith pattern has the jth token.

At block 206, in the first stage of runtime, an $n*m$ binary matrix is built where n is the number of tokens matched against the DFA and m is the number of signatures such that $e(i,j)$=true if and only if the ith token matches the jth token.

At block 208, in the second stage of runtime, the sub runtime $n*m$ matrix is obtained for a particular pattern such that $e(i,j)$=true if and only if the ith token matches the jth tokens in the pattern.

At block 210, a bipartite graph is built having two sets of vertex, n and m, where n is token and m is signature. There is a weight 1 edge between an ith vertex in n and an nth vertex in m if $e(i,j)$ in the matrix is true.

At block 212, indication(s) can be provided for malicious URL(s) that are identified such that appropriate response measures can be performed.

Figures

Figure 1

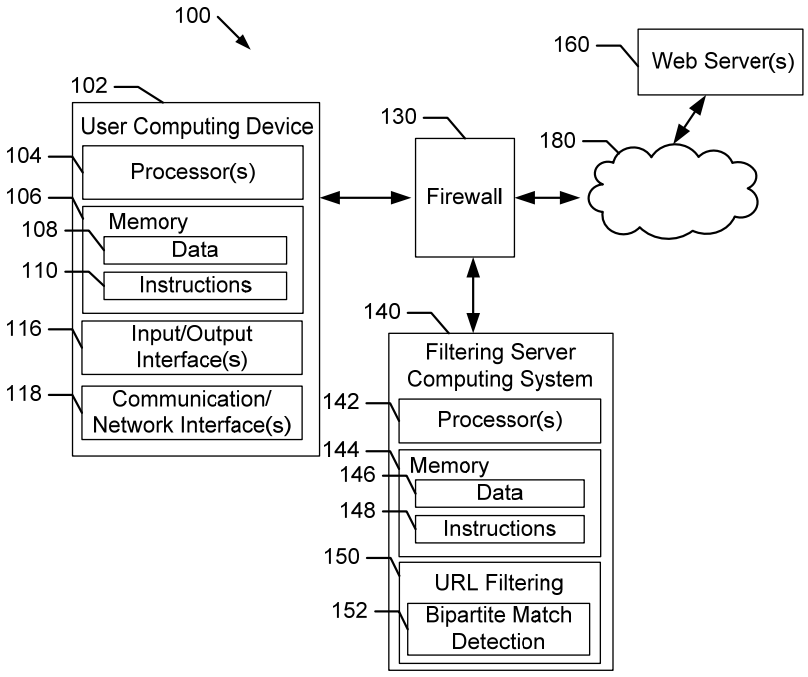
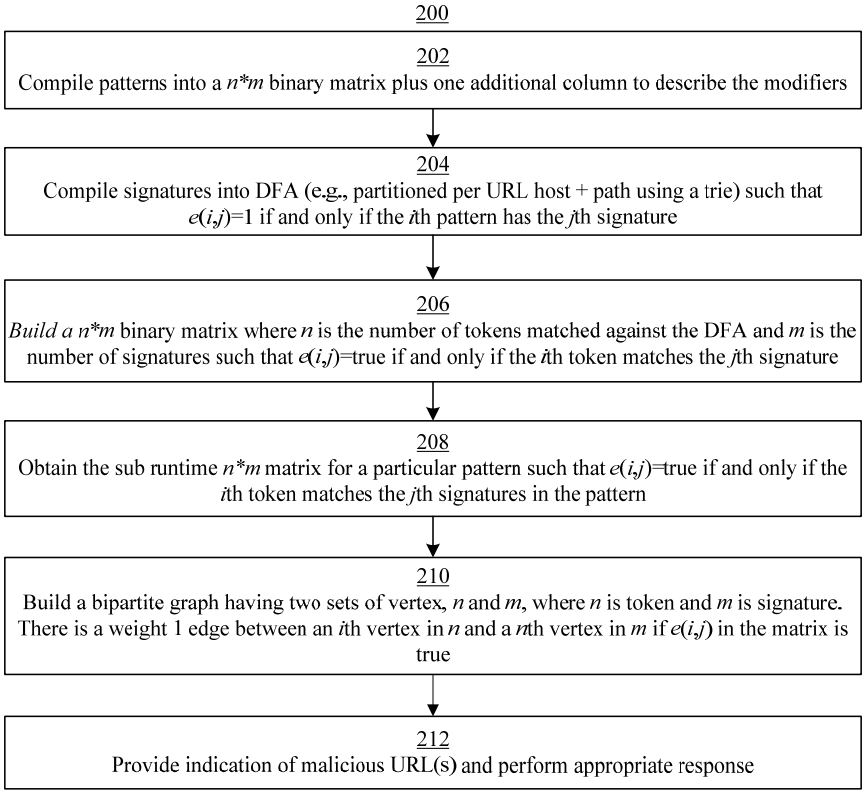


Figure 2



Abstract

The present disclosure describes systems and methods to enable inspecting of byte streams using a bipartite match algorithm to detect use of evasive techniques to defeat Uniform Resource (URL) filtering. According to an aspect of the present disclosure, a two stage dynamic programming algorithm is provided that matches signatures once and only once just like DFA in the first stage, and uses polynomial runtime to select a matching pattern in the second stage. According to an example implementation of the present disclosure, the patterns are compiled into an $n*m$ binary matrix plus one additional column to describe the modifiers, where n is the number of patterns and m is the number of unique signatures used in the patterns. In the first stage of runtime, an $n*m$ binary matrix is built where n is the number of tokens matched against the DFA and m is the number of signatures such that $e(i,j)$=true if and only if the ith token matches the jth signature. In the second stage of runtime, the sub runtime $n*m$ matrix is obtained for a particular pattern such that $e(i,j)$=true if and only if the ith token matches the jth signatures in the pattern. A bipartite graph can be built where there are two sets of vertex, n and m, where n is token and m is signature. There is a weight 1 edge between an ith vertex in n and an nth vertex in m if $e(i,j)$ in the matrix is true.