Predicting and Detecting Emerging Cyberattack Patterns Using StreamWorks

George Chin Jr., Sutanay Choudhury, John Feo
Pacific Northwest National Laboratory
P.O. Box 999
Richland, WA 99352
{George.Chin, Sutanay.Choudhury, John.Feo}@pnnl.gov

Lawrence Holder
School of Electrical Engineering & Computer Science
Washington State University
Pullman, WA 99164
holder@wsu.edu

ABSTRACT

The number and sophistication of cyberattacks on industries and governments have dramatically grown in recent years. To counter this movement, new advanced tools and techniques are needed to detect cyberattacks in their early stages such that defensive actions may be taken to avert or mitigate potential damage. From a cybersecurity analysis perspective, detecting cyberattacks may be cast as a problem of identifying patterns in computer network traffic. Logically and intuitively, these patterns may take on the form of a directed graph that conveys how an attack or intrusion propagates through the computers of a network.

We are researching and developing graph-centric approaches and algorithms for dynamic cyberattack detection and packaging them into a streaming network analysis framework we call StreamWorks. With StreamWorks, a scientist or analyst may detect and identify precursor events and patterns as they emerge in complex networks. This analysis framework is intended to be used in a dynamic environment where network data is streamed in and is appended to a large-scale dynamic graph. Specific graphical query patterns are decomposed and collected into a graph query library. The individual decomposed subpatterns in the library are continuously and efficiently matched against the dynamic graph as it evolves to identify and detect early, partial subgraph patterns.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory – *graph algorithms, network problems, trees.*
I.2.8 [**Computing Methodologies**]: Problem Solving, Control Methods, and Search – *graph and tree search strategies.*

General Terms

Algorithms

Keywords

Cyberattack detection, subgraph pattern matching, emerging subgraph patterns, dynamic networks, subgraph join tree.

CISR '14, Apr 08-10 2014, Oak Ridge, TN, USA
ACM 978-1-4503-2812-8/14/04.
http://dx.doi.org/10.1145/2602087.2602111

1. INTRODUCTION

Challenges exist in employing graphs for cyberattack detection, which have severely limited their practical use in cybersecurity applications. First and foremost, identifying cyberattack graph patterns from within a larger graph of a computer network is a classic subgraph isomorphism problem which is known to be computationally expensive and NP-complete. Another complexity is the requirement to conduct partial matching of the cyberattack graph pattern, such that one can detect the pattern before it is fully instantiated. In addition, the larger computer network graph would be dynamic or ever-changing with message patterns and host machines statuses constantly transitioning over time.

To address these challenges, we have been researching and developing graph-centric approaches and algorithms for dynamic cyberattack detection and packaging them into a streaming graph analysis framework we call *StreamWorks*. StreamWorks is used to identify emerging cyberattacks and cyberthreats in streaming computer network traffic. Computer network data may be modeled as a dynamic graph with nodes representing host machines and edges representing messages between host machines. The set of edges in the graph convey all the communications among host machines within a specific time window. With a time window, the size of the dynamic graph will continually grow and shrink as new nodes and edges are feed into the graph, while older nodes and edges are aged out.

2. DYNAMIC SUBGRAPH MATCHING AND CYBERATTACK DETECTION

Investigation of subgraph isomorphism for dynamic graphs introduces new algorithmic challenges because one cannot afford to index a dynamic graph frequently enough for applications with real-time constraints. In fact this is a problem with searches on large static graphs as well. There are two alternatives in that direction. One can search for a pattern repeatedly or one can adopt an incremental approach. The work by Fan et al. [1] presents incremental algorithms for graph pattern matching. However, their solution to subgraph isomorphism is based on a repeated search strategy. Chen et al. [2] proposes a feature structure called the node-neighbor tree to search multiple graph streams using a vector space approach. They relax the exact match requirement but require significant pre-processing on the graph stream.

In the cybersecurity context, some limited research has been conducted on using directed graphs to model cyberattack patterns [3, 4]. Regarding dealing with dynamic computer network data, our research is aligned with streaming algorithms for anomaly detection or intrusion detection. Distributed event monitoring and minimizing the amount of false positives are the major challenges for these systems. As an example, a DDoS attack is often hard to separate from a flash crowd event. Ganguly et al. [5] present a streaming algorithm to monitor the distinct source frequencies to distinguish between benign and malicious activities. Venkatraman et al. [6] present an algorithm to detect sources that connect to a large number of distinct connections in a streaming setting with specified accuracy and memory requirements.

3. CYBERATTACK GRAPH PATTERNS

To enable subgraph pattern matching, various types of cyberattacks may be depicted as temporal, multi-dimensional, directed multi-graphs. In most cases, the graphical patterns of cyberattacks have repeating internal structures. In Fig. 1, we show a few illustrative cyberattack graph queries as described below.

Witty worm – The Witty worm is an Internet worm that targets a buffer overflow vulnerability in Internet Security Systems products. It is known to attack port 4000 of Windows machines with packets of sizes between 796 and 1,307 bytes. As shown in Fig. 1a, the associated query graph looks to detect infected machines that are sending out packets with Witty worm characteristics to at least five other machines and a path of at least three machines that have been infected in chronological order. In the diagram, the chronological order of the messages is indicated by edge color transitioning from light to dark blue.

Smurf distributed denial-of-service (DDoS) – DDoS attacks typically involve a hacker sending messages to intermediate host machines with the spoofed source address of the victim machine. In the case of the Smurf DDoS attack of Fig. 1b, the hacker sends an "ICMP Echo Request" message to a broadcast IP address that appears to come from the victim. A router will pick up the message and broadcast it to intermediate host machines. In response, the intermediate host machines then floods the victim machine with "ICMP Echo Reply" messages.

Fraggle DDoS – As shown in Fig. 1c, a Fraggle DDoS attack is the UDP version of a Smurf DDoS attack and has a similar graphical structure. In the Fraggle attack, a "UDP Echo Request" message is broadcast to port 19 of intermediate host machines, which in turn, sends the "UDP Echo Response" message to port 7 of the victim machine. The UDP version may be devastating because it may initiate a repetitive echo request-response loop between the intermediate host machines and the victim.

Domain Name System (DNS) amplification DDoS – In a DNS amplification DDoS attack, zombies or agents generate a large number of DNS queries with a spoofed source address and send these queries to various DNS servers. As shown in Fig. 1d, the DNS requests appear to come from the victim machine. The DNS servers respond with three different possible types of messages back to the victim machine, which are the "DNS Standard Query Response," "ICMP Destination Unreachable," and "Fragmented IP Address" messages. Such attacks are particularly effective because DNS response packets may be significantly larger in size in comparison to the initiating DNS request packets.

4. SUBGRAPH JOIN TREE

To manage and track a set of precursor subpatterns associated with a query graph, we introduce the concept of a subgraph join tree (SJT), which decomposes a query graph into smaller search subgraph patterns. These smaller subpatterns signify precursor events that emerge early before the full query pattern is complete. As precursor events are detected in data streams, they are matched to the nodes of a SJT and join to other partial matches that have previously occurred to signify larger matches. Matching that occurs higher within the internal nodes of the SJT indicates a higher probability that an attack is occurring. A query graph matching score may be computed based on where the matching is occurring in the SJT through different formulas including using the proportion of edges in the search subgraph pattern to those in the complete query graph pattern as shown in Fig. 2a

The SJT is a binary tree that successively divides a graph into two children subgraphs with specific vertices as join points. A SJT for a simple query graph pattern is shown in Fig. 2a. The query graph pattern illustrates host machines attacking a DNS server and a Web server. The root node of the SJT represents the full query graph pattern, while each descending level decomposes a section of the query graph. Decomposition continues until a primitive search subgraph pattern is reached, which should be small and discriminative enough to be efficiently found in the dynamic graph as an exact subgraph match. By limiting exact subgraph matching to only small search patterns and using the SJT to incrementally enlarge matches, we are able contain the computational cost of subgraph matching.

Fig. 2b-d illustrates how dynamic subgraph matching occurs with a query graph decomposed into a SJT. Subgraph matching occurs in the context of a dynamic graph that is continually updated as data batches are fed in through data streams. As new nodes and edges are inserted into the dynamic graph from the batch updates, the SJT is matched against the dynamic graph to initiate new partial matches and extend previous ones. In Fig. 2b, the first primitive subgraph pattern is found in the dynamic graph, which is matched to the subgraph leaf at the bottom-left of the SJT. This partial match is saved and tracked.

As data continues to arrive, the dynamic graph evolves and additional partial patterns are found. Fig. 2c illustrates the case when a newly-found partial pattern is a sibling to a previously-found

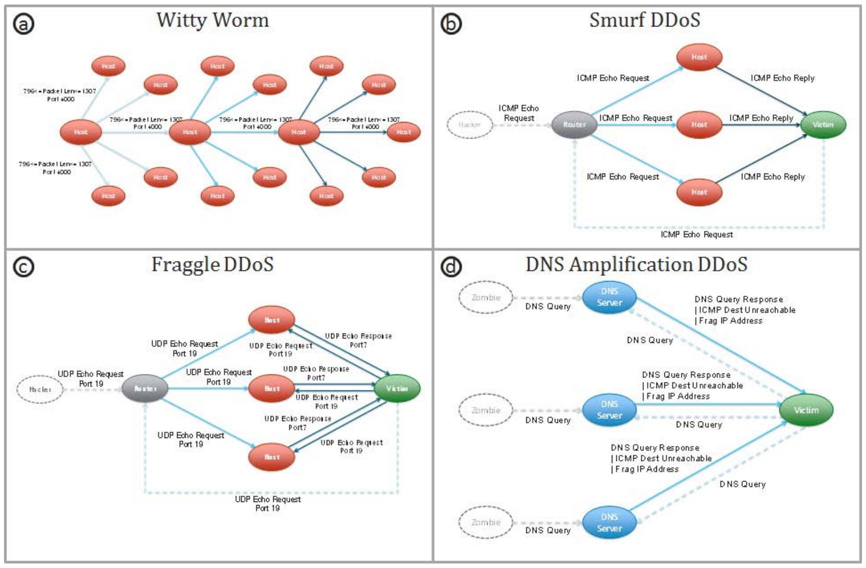

Figure 1. Cyberattack graph queries for a) Witty worm, b) Smurf DDoS, c) Fraggle DDoS, and d) DNS Amplifications DDoS cyberattack.

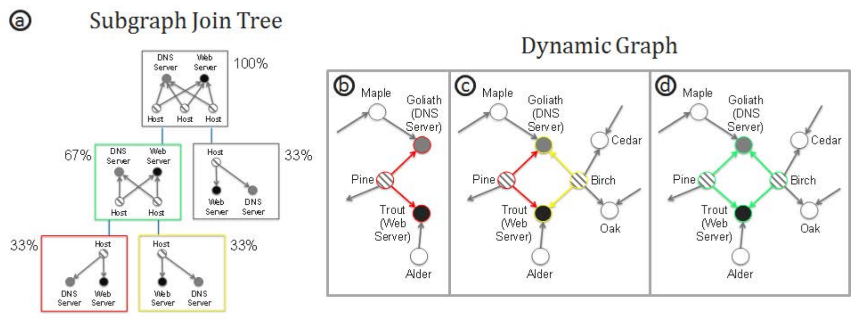

Figure 2. a) An example SJT for a simple cyberattack query graph, and b-d) the subgraph matching and joining process in a dynamic graph.

partial pattern in the SJT. In order for the two sibling patterns to combine to form a larger partial match and move up the join tree, however, the sibling patterns must have matching "DNS Server" and "Web Server" nodes, which are their join points. If these dependencies are satisfied, the two sibling partial patterns connect at their join points to construct the larger partial match (see Fig. 2d), which is stored and tracked as a new partial match. The initial sibling patterns also continue to be stored and tracked since they along with the larger partial match are all subject to additional joining. Over time, the joining of partial matches up the join tree may lead to the detection of the full query pattern.

5. JOIN TREE GENERATION

We are developing an interactive graph query construction tool that will allow an analyst to build a query graph similar to those depicted in Fig. 1. With this construction tool, an analyst builds a query graph by drawing nodes that represent machines and edges that represent message flow. The query attributes of the nodes and edges are also defined using this tool. For nodes, query attributes include the label, hostname, machine or address type, IP address, and port number. For edges, query attributes include protocol, message type, packet length, timestamp, and order.

Once a query graph is constructed, the conversion to a SJT would be mostly automatic. A join tree generation tool would use the graph diagram, node and edge attribute specifications, and the ordering details to generate a temporal layout of the query graph. The generation tool would then traverse the temporal layout in either a depth-first or breadth-first fashion based on user preference to extract graph partitions and identify join points across partitions. Figure 4 shows both a depth-first and breadth-first SJT for the Smurf DDoS attack pattern.

In Fig. 3a, the depth-first SJT is organized along individual circular paths from the victim to the router to the host machine and back to the victim as seen in the network traffic. Each path has temporal constraints which fixes the message order that must take place for the pattern to match. As shown, every relative edge in a join tree partition has a unique id (R1, R2, R3, …), which is used in specifying temporal constraints. For each path, the router is distinguished from other types of machines by having the attribute of being able to multiplex messages that have been sent to a broadcast address. Since all the primitive search subgraph patterns of the depth-first SJT is of the same exact structure, the SJT may be collapsed into a more efficient equivalent form as shown in Fig. 3b, which has the same join flow as the corresponding binary tree of Fig. 3a, but should simplify the storage and tracking of partial matches.

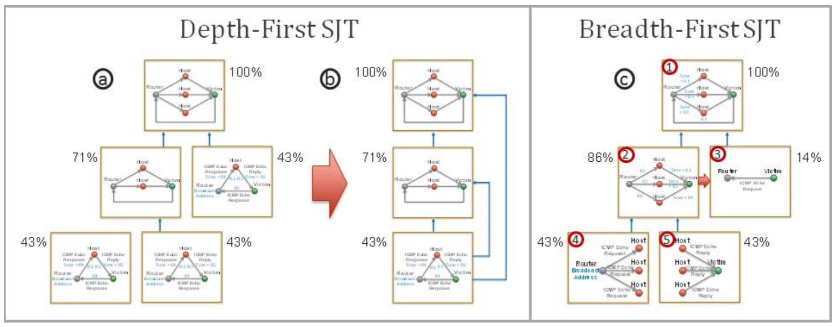

Figure 3. a) Depth-first SJT for Smurf DDoS query, b) optimized depth-first SJT for Smurf DDoS query, and c) breadth-first SJT for Smurf DDoS query.

The breadth-first SJT of Fig. 3c is organized along the expected temporal ordering of messages through the query graph pattern. For the Smurf DDoS attack, the spoofed message from the victim to the router occurs first, followed by "ICMP Echo Request" messages to different host machines, and then completed by "ICMP Echo Reply" messages to the victim. In a breadth-first SJT, these temporal stages would be used to partition the query graph into subgraph patterns for partial matching. With the Smurf DDoS query pattern, however, the very first temporal primitive subgraph pattern would be a single edge representing the initial spoofed "ICMP Echo Request" message, which is likely to find an enormous number of matches in the dynamic graph.

We are developing capabilities that will allow analysts to test a SJT against a data stream or dataset and that can automatically optimize a SJT based on subgraph matching characteristics and frequencies. The speed of subgraph pattern matching may be accelerated by collecting and utilizing node and edge frequency information to optimize search paths through a massive dynamic network. In the case of the Smurf DDoS breadth-first SJT, the joint tree generation tool would move the high-frequency single-edge primitive subgraph pattern up the SJT, to be searched for only in the case when a larger, more discriminative match has already been found. This specific logic and dependency is denoted by the red arrow in the SJT of Fig. 3c.

In the breadth-first SJT, the edges of the primitive subgraphs will generally occur within the same timeframe. Thus, any temporal constraints among query graph edges are likely to occur across sibling subgraphs of the SJT. In the case of Fig. 3b, many partial matches of the two primitive subgraph patterns lowest in the SJT (partitions 4 and 5) may occur, but a pair of primitive partial matches will only be joined when the temporal dependencies of the parent partition (partition 2) are satisfied. The temporal dependencies specified at the root node of the SJT (partition 1) are handled similarly. In joining across partitions, we reference edge ids (E1, E2, E3, …) in the context of the full query graph rather than relative to just the subgraph partition.

For the Smurf DDoS, the breadth-first SJT should provide faster detection of precursor subpatterns since it specifically looks for subpatterns that will occur early in time, while the depth-first SJT will require one full cycle of interactions, infections, or intrusions to occur before any breadth-first SJT subpatterns are detected. The breadth-first SJT is most effective when specific subgraph patterns in the query graph occur very early on such as in DDoS attacks, or when the node degrees of the query graph are relatively high such as in virus and worm attacks. The depth-first SJT is most useful when the attacker is predestined to follow specific attack vectors in hopes of reaching particular critical resources. It is also better suited to detect surgical, human-in-the-loop cyberattacks where actions are more planned and deliberate.

6. VISUALIZING PATTERNS

We programmed the Gephi graph visualization software [7] to read in and display a dynamic computer network graph and partial subgraph matches that are emerging in the graph. Fig. 4 shows snapshots of emerging subgraph patterns in a computer network graph that are identified and tracked using both the depth-first and breadth-first SJTs. The colors of the subgraph patterns in the snapshots correspond to partitions in the associated SJT, which indicate the degree of partial matching to the full query graph. Matching percentages are shown for each SJT.

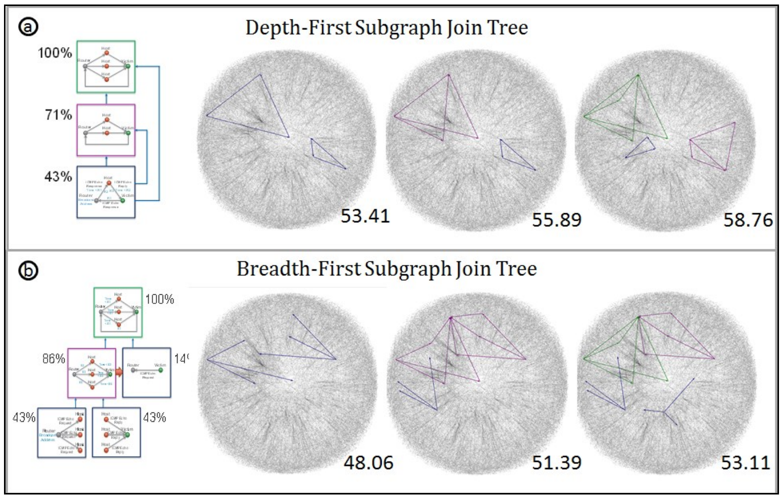
a
Depth-First Subgraph Join Tree
100%
71%
43%
53.41
55.89
58.76
b
Breadth-First Subgraph Join Tree
100%
86%
14%
43%
43%
48.06
51.39
53.11

Figure 4. Emerging Smurf DDoS subgraph patterns in a dynamic computer network graph using a) depth-first SJT, and b) breadth-first SJT.

In examining the partial matches stemming from the depth-first SJT of Fig. 4a, we see the first primitive subgraph patterns (blue subgraphs) emerging at 53.41 seconds into the data stream. Later, at 55.89 seconds, two primitive subgraph patterns join to signify a larger partial match (magenta subgraph) as we move up the SJT. Finally, at 58.76 seconds, the larger subgraph pattern joins with another primitive subgraph pattern to form an instance of the full query graph (green subgraph).

When using the breadth-first SJT of Fig. 4b, we see that the first primitive subgraph patterns (blue subgraphs) are found earlier at 48.06 seconds. A larger subgraph pattern, however, is not found until 51.39 seconds into the data stream. The full query graph is found at 53.11 seconds. As previously discussed, the breadth-first SJT may provide faster detection of precursor subgraph patterns than the depth-first SJT for certain types of cyberattacks because the subgraph partitions are temporally ordered and do not require a path traversal through a strand of the attack.

As shown in Fig. 4, the full Smurf DDoS attack query graph pattern only takes a few seconds to emerge from its initial subpatterns. For this cyberattack, the amount of time available to take preventive or defensive actions would be very limited. StreamWorks capabilities would be most useful in cases where the cyberattack proceeds over a longer period of time or when there are other network activities apart from the actual attack that can foreshadow the impending attack. In the case of the Smurf DDoS attack, for instance, the precursor pattern of the attacker scanning for misconfigured routers to identify usable broadcast addresses could have been added as part of the query graph pattern.

7. CONCLUSION

The theses of our research are that graph-based representations of cyberattack patterns may serve as powerful and effective conceptual models for interactive cybersecurity analysis, and a graph-based approach and framework may be developed to detect cyberattack graph patterns in a dynamic computer network graph before those patterns are fully realized. Towards these goals, we have developed a SJT-based subgraph pattern matching approach, which partitions a query graph into smaller subgraphs that may be used to identify precursor or emerging patterns or events.

Beyond computer networks, we have also applied StreamWorks to other forms of networks including citation and social networks. A concerted effort is currently underway to validate StreamWorks with cybersecurity analysts on authentic streaming large-scale network flow data such that we may better evaluate and tune its effectiveness, accuracy, and performance.

8. ACKNOWLEDGMENTS

This work was funded by the Center for Adaptive Supercomputing Software (CASS) at the U.S. Department of Energy's Pacific Northwest National Laboratory, which is operated by Battelle Memorial Institute under Contract DE-ACO6-76RL01830.

9. REFERENCES

[1] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu, "Incremental Graph Pattern Matching," *Proc. 2011 ACM SIGMOD International Conference on Management of Data*, ACM Press, 2011, pp. 925-936.

[2] L.Chen and C. Wang, "Continuous Subgraph Pattern Search Over Certain and Uncertain Graph Streams," *IEEE Trans. on Know. and Data Eng.,* vol. 22, no. 8, 2010, pp. 1093–1109.

[3] A. Godiyal, M. Garland, and J.C. Hart, "Enhancing Network Traffic Visualization by Graph Pattern Analysis," 2010, https://agora.cs.illinois.edu/download/attachments/18744303/netflowpatterngraphs.pdf.

[4] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "GrIDS a Graph Based Intrusion Detection System for Large Networks," *Proc. 19th National Information Systems Security Conference*, 1996, pp. 1-10.

[5] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani, "Streaming Algorithms for Robust, Real-Time Detection of DDoS Attacks*," Proc. 27th International Conference on Distributed Computing Systems*, IEEE Press, 2007, pp. 1-4.

[6] S. Venkataraman, D. Song, Phillip B. Gibbons, and A. Blum, "New Streaming Algorithms for Fast Detection of Superspreaders," *Proc. 12th ISOC Symposium on Network and Distributed System Security Symposium (SNDSS),* IEEE Press, 2005, pp. 21-30.

[7] Gephi, an Open Source Graph Visualization and Manipulation Software, http://www.gephi.org/.