Cyberthreat analytics using graph analysis

Eric Dull
Cray, Inc
Seattle, WA
edull@cray.com

Abstract— **In cyber security analysis, everything is growing at prodigious rates: data feeds, numbers of computers and phones on the network, number of users, and the number of threats that security and information technology professionals must evaluate, prioritize, and mitigate. The one thing that isn't growing in enterprises is the number of security professionals available to perform this analysis. Without enough expert staff, every enterprise only gets further and further behind the curve with the threats. Enterprises need a different analytic approach and strategy to make every one of their personnel more effective. Cyber-threat analytics is that approach. Cyber-threat analytics focuses the analyst time available in the enterprise to the most important threats present within the enterprise by combining threat data with computer network data using graph algorithms.**

Graph algorithms have been applied to computer network data in multiple contexts. Cray, through engagements with its customers and partners, has utilized graph analytics in performing computer network analysis and identifing threats and risks to enterprise-scale networks. One of these networks is SCinet. SCinet is the high-bandwidth network that supports the SC technical conference and exhibit hall. SC14 had the largest network to date, with 1.2 terabits per second reaching the show floor and 11,000 devices using SCinet. SCinet has a /17 subnet (32,766 IP addresses) of publicly routable IPv4 space and is designed, constructed, operated, and dismantled by 300 volunteers.

SCinet's scale of data (18 billion triples from 5 days of data), time-to-first solution (analytics need to be developed in minutes to an hour or two), and time-to-solution (answers need to be generated in seconds to minutes to be useful) requirements make it a great analogue for enterprise networks and analytic challenges.

This paper describes computer network data, cyber-threat analytics, graph algorithms, and a use case where these have been applied together to perform analysis on an enterprise-scale network.

Keywords: Graph algorithms, computer network security, SCinet, XMT2, Urika-GD, SPARQL, RDF

I. INTRODUCTION

Computer network data analysis and computer security analysis are areas that continue to grow in complexity, scale and maturity. Existing analytic approaches are signature based analytic approaches that look for known malicious traffic. These approaches are too inflexible to adapt to new behaviors or attacks in a dynamic analytic environment such as an enterprise-scale network. More flexible approaches give analysts more freedom to change analytics as the threats change but suffer from poor performance when faced with enterprise scale data sets. Cray has used graph analysis techniques and its Urika-GD graph discovery appliance [1] to successfully perform flexible, large-scale computer network analysis. This paper discusses Cray's approach and analytic techniques.

II. BACKGROUND

Understanding Cray's use of graph algorithms to analyze computer network data requires a base of knowledge in computer network data, graph algorithms, challenges faced in performing analysis with computer network data, a definition of cyberthreat analysis, and Cray's Urika-GD graph discovery appliance.

A. Computer network data

Computer network data consists of at least two types of data: transactional and enrichment. Transactional data is voluminous and represents actions and activities across the network. Enrichment data is smaller is scale than transactional data and provides context about the meaning of actors generating transactions. In computer network data, actors include computers and users. Transactional and enrichment data are used together to produce meaningful analytic results. For example, a common transactional data type, Netflow [2], contains information about two IP [38] (Internet Protocol) addresses that have communicated with each other and the nature of that communication. Enrichment data, such as how those IP addresses are assigned and the common usages of the ports used in the communication, allow analysts to identify common and unusual activities on a given enterprise's network when combined with the transactional data.

Transactional computer network data is generated by network sensors observing computers using networks at many different levels of the OSI (Open Systems Interconnection) network model [36], from layer 2 ARP [37] (Address Resolution Protocol) requests through layer 3 IP traffic, layer 4 TCP [34] (Transmission Control Protocol) and UDP [35] (User Datagram Protocol) traffic, layer 5 sessionized TCP traffic, and layer 7 applications such as HTTP [3], Domain Name Service [4], FTP [5], and Telnet [6]. Common transactional data types include network flow activity, Domain Name Server requests and responses, emails, Application indicators, and HTTP requests and responses.

Enrichment data includes enterprise IP role (server, client, desktop, wireless client, mobile device, etc) and assignment, WHOIS [7] database information, common network port assignment, and Autonomous System information. Enrichment data are static data sets, smaller in scale and volume, and change more slowly than transactional data.

B. Challenges encountered performing computer network data analysis

Analyzing computer network data encounters many challenges. These challenges include managing and interacting with the millions to billions of records in transactional datasets, merging the transactional datasets with enrichment data sets, and generating results in seconds to hours. Transactional data sets are growing larger as enterprises allow more and more devices to access their networks. Published attacks on enterprises are also increasing, adding to the amount of analysis required on these datasets to protect enterprises' core business practices. However, the computer security field is understaffed. There is currently unmet demand for 1 million computer security professionals [8].

Enterprises do not have enough qualified analytic staff to perform the computer security analysis to protect their business practices and their analytic tools and approaches fail to overcome these challenges. Current analytic approaches produce incomplete or inadequate findings in too long an amount of time to allow enterprises to overcome these challenges and perform timely, meaningful computer security analysis. In fact, 99% of Point of Sale / credit card payment system breaches are discovered by someone other than the victim enterprise [9].

C. Applying graph analytics to computer network data analysis

Graph analysis is an approach to performing computer network data analysis that can overcome these challenges. To perform graph analysis, the data to be analyzed is represented as a graph. A graph is a computer science data structure that consists of nodes and edges. Nodes represent objects: people, places, computers, time, or space. Edges represent interactions or relationships between those objects. Graphs can represent nearly any type of structured data.

Computer network data is structured and easily represented in graph data structures. This graph representation addresses one of the challenges present in current analytic systems: merging multiple data sets. The nodes in the graph can represent IP addresses, hostnames, and other entities present in computer network traffic. The edges in the graph can represent the presence of these entities in different types of data. When structured data sets are transformed into graphs, they can merge.

Computer network data can be transformed into a graph by defining a mapping of entities present in computer networks, such as IP addresses, hostnames, and autonomous systems, to nodes and activities found on the network, such as one IP address connecting to another via TCP sessions or a hostname is assigned to an IP address through DNS, to edges. Each graph format requires a different mapping process, and an example mapping of Netflow to RDF [23] (Resource Description Framework) is [14]. Example conversion code is available at [33].

Once the computer network data sets are represented as a graph, they can be analyzed using graphs algorithms. Graph algorithms have been applied to computer network data to address multiple analytic challenges, including botnet identification [10], unknown tradecraft-unknown threat identification [11] and high-speed network analysis [12].

D. Graph algorithms

There are a number of graph algorithms that are of interest when performing computer network data analysis. These algorithms include:

- Subgraph isomorphism/pattern-matching [13],
- Jaccard similarity coefficient [14],
- Community detection [15],
- Shortest path [16],
- Betweenness centrality [17], and
- Badness propagation [18].

Subgraph isomorphism or pattern matching is a simple graph algorithm that is analogous to a SELECT statement in SQL [19]. This algorithm accepts a user-input pattern, much like Boolean logic and returns all subgraphs, portions from the main graph, that match the analyst-provided pattern. Subgraph isomorphisms are used to select computer network activity of interest to the analyst. The analyst then analyzes these subgraphs. An example isomorphism is "Find all network flow records where the source and destination port at greater than 50000 and occurred between 2 and 3 am."

Jaccard similarity coefficient is a graph algorithm that finds nodes in the graph that have a high degree of connectivity to an analyst-provided node. This algorithm accepts an analyst-provided seed node (V1) and outputs a ratio as defined in (1) for the similarity to other nodes in the graph, such as V2. The numerator in (1) is the number of nodes connected to both V1 and V2, and the denominator in (1) is the number of nodes connected to V1 or V2. Figure 1 visually illustrates (1). The numerator for Figure 1 is 5, and the denominator is 11. The resulting J for V1 and V2 is 5/11.

The analyst uses the list of coefficients to determine which of the candidate V2 nodes is most similar to the provided V1 seed node. An example application is botnet command-and-control channel identification. The analyst provides a seed port, from an intrusion detection system alert about a malware download to an infected system. The Jaccard similarity coefficient would be applied to the graph of the network flow information to determine which other ports are frequently observed being used by the same systems as the port used for the malware download.

$$J_{V1\_V2} = |V1 \cap V2| \;/\; |V1 \cup V2| \tag{1}$$

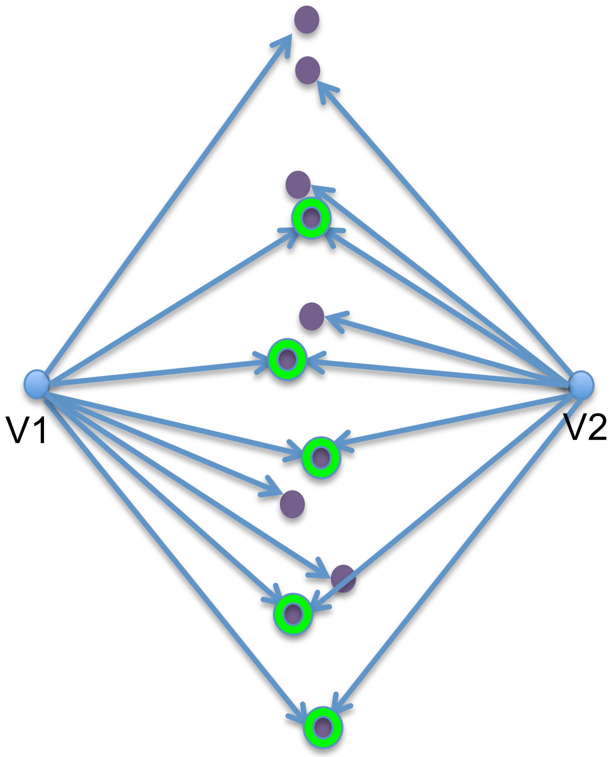


Figure 1. Example graph. V1 is the analyst-provided seed node. V2 is a candidate node considered for its similarity. Green nodes are in contact with both V1 and V2, and blue nodes are in contact with V1 or V2.

Community detection algorithms [20] identify groups of nodes that are related to each other, using a variety of measures of "related" including structural elements or maximizing connectivity within groups while minimizing connectivity outside of groups. This algorithm accepts an input graph and outputs a list of nodes and the group or "community" to which each node belongs. These types of algorithms should be used with particular care paid to the composition of the input graphs, as these algorithms analyze the topology of the input graph without consideration as to the semantic meaning of the nodes and edges. For example, building an input graph that consists of network flow and user logins mixes two types of behavior (computer network activity and user login activity) into one graph. Running a community detection algorithm across this graph produces meaningless communities as the input graph conflates different behaviors.

An example application of a community detection algorithm is building a graph of email communications and then identifying communities of users based on their communication patterns. This graph would help security analysts identify email accounts that start sending emails across communities in an unusual manner.

Shortest path algorithms identify the shortest paths between two nodes or groups of analyst-supplied nodes. The shortest paths are the ones that involve the fewest number of edges to connect a node from each analyst-supplied group. The algorithm accepts an input of two sets of nodes and outputs the shortest paths that connect the two sets. The analyst then considers the results to identify the paths that are most likely to answer the analytic question at hand. An example application of shortest path to a computer security analysis problem is having two sets of computers infected with different pieces of malware. Shortest path analysis would reveal users or network activity connecting the two sets, aiding root-cause analysis of the infections.

Centrality algorithms identify the most-central nodes within a graph, focusing analyst time and attention on the identified nodes. There are many types of centrality algorithms, including degree, eigenvector, eigenvalue, and betweenness. Betweenness centrality, also known all-pairs shortest-path) identifies the most-central nodes or edges through the method outlined in the pseudo-code below:

- From every node, compute the shortest path(s) to every other node
- For every node, count the number of shortest paths that go through it
- For every edge, count the number of shortest paths that go through it
- Divide the shortest path counts by the total number of shortest paths to generate centrality scores

Betweenness centrality requires an input graph and outputs a list of scores for the nodes and the edges as to the fraction of shortest paths that pass through a given node or edge. An example application of betweenness centrality in computer network security is vulnerability analysis. An analyst would represent an enterprise's network configuration as a graph and apply betweenness centrality to that graph to identify which nodes and edges are most responsible for efficient communication across the enterprise's network. If these nodes or edges are lost, the enterprise will suffer the greatest degradation of network performance.

BadRank is an algorithm that takes known bad actors or elements in a graph and identifies other actors or elements that are likely also bad. Conceptually, it is an extension of "guilt-by-reference" or "guilt-by-association." An example application in computer network security is the propagation of badness from known spam websites to other websites that are referred to by the known spam websites.

These graph algorithms address another challenge present in current analytic tools: the maturity of workflows and analytic approaches. Graph algorithms represent a rich, mature set of techniques that analysts can use to produce richer, more meaningful analytic results.

E. *Cyberthreat analysis*

Cyber security, itself a term of nebulous meaning, is growing field with emerging terms, techniques, and tools, and cyberthreat analysis is another emerging term that needs definition. Lockheed Martin has published a “Cyber Kill Chain” that describes the anatomy of a computer exploitation event [21]. This anatomy describes the reconnaissance, weaponization, delivery, exploitation, installation, command-and-control, and adversary action on objectives steps that occur during the infiltration of an enterprise’s network. Adversary activity should be identified as early in the “Kill Chain” as possible to give enterprises the largest amount of time to craft and realize a response to prevent or mitigate adversary activity. Cyberthreat analysis is where a number of enrichment and transactional datasets are combined to examine possible reconnaissance activities that would be the start of an adversary action chain.

F. Cray’s Graph Analytic Appliance

The final challenge that needs to be overcome is that posed by the sheer scale of transactional datasets (billions) and the performance requirement to deliver human-interactive time (1-100 seconds) analytic results on algorithms run over the transactional data sets. Cray’s Urika-GD graph discovery appliance satisfies these requirements and enables enterprises to overcome the challenges and meet their computer security analysis needs.

Urika-GD provides responses to individual queries in approximately 1-100 seconds and utilizes two key characteristics to provide this response time:

- SPARQL [22] / RDF and
- Their robust, efficient implementation on Cray’s XMT2 hardware [24]

1) RDF / SPARQL

SPARQL is a query language for semantic-graph databases containing data represented in the Resource Description Framework (RDF), with its name being a recursive acronym for SPARQL Protocol and RDF Query Language. It comes from the semantic web community and is a recommendation of the World Wide Web Consortium [4]. The primary goal for RDF was to make web pages machine- readable, and the goal for SPARQL was to enable higher-level querying of the semantic web. The resulting capabilities proved to be valuable for graphs that did not necessarily originate as web pages; i.e., queries on highly heterogeneous and richly interconnected data, data that reflected the Open World Assumption [25] that one’s set of data is never complete and so tools must be built expecting to easily incorporate new data and new types of data. Readers who use SQL will find many SPARQL constructs familiar.

RDF defines data in terms of triples consisting of a subject, a predicate or relationship, and an object. For example, the triple ”Cray is-headquartered-in Seattle” has ”Cray” as the subject, ”is-headquartered-in” as the predicate, and ”Seattle” as the object. Well-defined RDF data will use Universal Resource Identifiers (URIs, [26]) for subjects, predicates, and most objects. An RDF graph is a collection of these triples.

An example of SPARQL graph matching comes from Cray’s SCinet [27] analysis work. It is presented in lines 1-9.

```
1.  PREFIX      ocop:    <http://opencog.net/p/>
2.
3.  SELECT ?src_addr ?dest_addr
4.  WHERE {
5.  ?uid ocop:sourceAddress ?src_addr .
6.  ?uid ocop:destinationAddress ?dest_addr .
7.  ?uid ocop:hasProtocol oco:proto#tcp.
8.  ?uid ocop:destinationPort oco:port#22.
9.  }
```

SPARQL variables are denoted by an initial ? or $ character, e.g, ?src_addr in the example above. In the example, the SPARQL statements within the WHERE clause lines(4-9), known as a basic graph pattern, can be interpreted as ”find all ?uids that have a protocol of tcp and port of 22 (lines 7-8) and then find all ?src_addr ?dest_addr (lines 5-6) pairs (line 3) that are connected to those ?uids.” In English, the query can be restated as ”find source and destination address that are communicating using SSH [39] (Secure Shell) (tcp port 22).”

Cray’s implementation of RDF/SPARQL was designed from the outset for very large memory, extreme parallelism, and production environments. Its performance on standard SPARQL benchmarks such as LUBM [28] is unmatched because competing systems do not even publish their results on big data sizes. While conforming with the SPARQL 1.1 standard, Cray has extended SPARQL with mechanisms such as built-in graph functions that enable deep analysis of on-the-fly generated graph in human-responsive time. These built-in graph functions include community detection, betweenness centrality, shortest-path, and BadRank algorithms.

2) Urika-GD’s hardware characteristics

Urika-GD’s custom hardware has three distinct elements:

- its large shared memory,
- scalable I/O system and
- purpose-built Threadstorm™ graph processor.

Urika-GD’s huge, globally shared memory architecture of up to 512TB can hold the entire graph of relationships (5 billion – 2.5 trillion relationships) in memory. The scalable I/O subsystem, which can scale up to 350TB per hour, enables continuous updates to the graph as new data streams in. If the entire graph can’t fit in memory, a number of graph partitioning can be applied to segment the graph and use the scalable I/O subsystem to load and unload the segments into and out of memory.

The massively multi-threaded architecture of the Threadstorm™ processor (128 independent threads) is specially designed for analyzing graphs and allows threads to continue executing even if some are waiting for data to be returned from memory, preventing any processor stalls. This architecture delivers orders of magnitude better performance on complex graph problems than commodity hardware [40].

III. Representative use case: SCinet at SC14

SC [29] is an annual supercomputing technical conference in its 28th year supported by SCinet, the network providing connectivity to the conference and exhibit hall. SC14 refers to the conference held during 2014. Cray participated in SCinet by performing computer network security analysis as part of the network security committee.

A. *What is SCinet?*

SCinet is the high-bandwidth network that supports the SC technical conference and exhibit hall. SC14 had the largest network to date, with 1.2 terabits per second reaching the show floor and 11,000 devices using SCinet. SCinet has a /17 (32K IP addresses) of publicly routable IPv4 space. The SCinet network diagram is in Figure 2 and demonstrates the complexity of SCinet and the elements that are similar to enterprise networks. SCinet is a volunteer organization of 300+ individuals from National Labs, universities, and vendors working together to design, construct, operate, and dismantle the network every year. The network is operational for approximately 10 days every year during the conference.

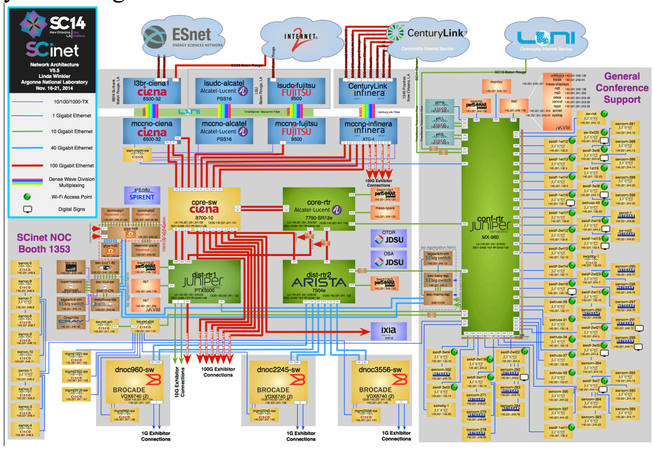


Figure 2. SC14's SCinet network layout

At SC14, the SCinet network security team had two primary missions:

1. Detect and mitigate any outbound scanning or attacking behavior.
2. Identify, quantify and inform likely compromised hosts on the network.

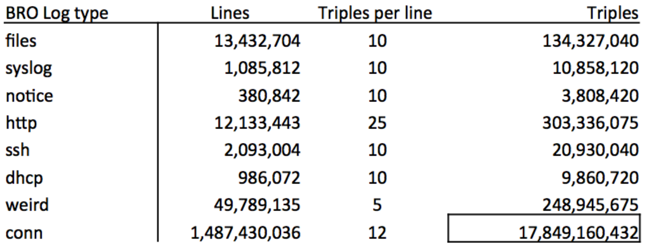
Cray supported these missions with "Discover," a 2 terabyte Cray Urika-GD graph discovery appliance.

B. *Cray's SCinet participation*

Cray participates in SCinet on the network security committee, because SCinet is a reasonable analogue for an enterprise-scale network, with 11,000 devices on the network and billions of transactional records (1.6 billion transactions generated in 5 days). SCinet's computer network data analysis encounters the same challenges as enterprises when conducting computer network data analysis. Cray's analytic approach at SCinet demonstrates overcoming the data scale time-to-first solution, and time-to-solution challenges.

C. *SCinet computer network data*

Data is key when performing analysis, and at SCinet, the network security team used BRO [30], an open-source deep packet inspection system to monitor nearly 290 GB/s of network bandwidth. These sensors generated nearly 1.6 billion transactional records, and Cray parsed the records into 18.6 billion RDF triples for analysis. These transactional records included multiple data types, including network flow, HTTP, file download (files), odd network behaviors (weird), IDS alerts (notice) [31], SYSLOG notices [32], and SSL [33] certificate information. Figure 3 is a table of BRO-generated transactional records and triple counts for each type of transactional data. The SCinet network security team combined these transactional datasets with enrichment data of IP assignment / role information for each of the subnets present in SCinet's network block.



BRO Log type	Lines	Triples per line	Triples
files	13,432,704	10	134,327,040
syslog	1,085,812	10	10,858,120
notice	380,842	10	3,808,420
http	12,133,443	25	303,336,075
ssh	2,093,004	10	20,930,040
dhcp	986,072	10	9,860,720
weird	49,789,135	5	248,945,675
conn	1,487,430,036	12	17,849,160,432

Figure 3. Transactional records generated and used during SCinet network security analysis

D. *Graph algorithms used during SC14*

Cray analyzed the graph formed from the RDF triples generated from transactional records with three graph algorithms: subgraph isomorphism / pattern matching, Jaccard similarity coefficient, and betweenness centrality. This section outlines the SPARQL code used to execute these algorithms with Discover during SC14.

1) *Jaccard similarity coefficient*

This SPARQL code performs Jaccard similarity coefficient calculations against graphs of network flow data. Line 29 contains the analyst-provided seed, V1, port 9162. Line 4 contains ?port and ?proto which are the candidate V2 ports, and ?client_count is the numerator, and ?big_client_count is the denominator from (1).

Written in approximately 60 minutes and executing in less than 1 minute in its finished form, this example, development time, and execution time demonstrate the applicability of SPARQL and RDF to computer network analysis problems.

```
1.  PREFIX    ocop:    <http://opencog.net/p/>
2.  PREFIX    oco:     <http://opencog.net/>
3.
4.  SELECT ?proto ?port ?client_count ?big_client_count
```

```
WHERE {
{ SELECT ?proto ?port (count(distinct ?ap_addr) as ?big_client_count)
WHERE {
?uid3 ocop:sourceAddress> ?ap_addr.
?uid3 ocop:destinationAddress> ?dest_addr2 .
?uid3 ocop:destinationPort> ?port .
?uid3 ocop:hasProtocol> ?proto .
?uid3 ocop:hasRespBytes> ?rbytes2.
}
GROUP BY ?proto ?port
}
{
SELECT ?proto ?port (count(distinct ?ap_addr) as ?client_count)
WHERE {
?uid3 ocop:sourceAddress> ?ap_addr.
?uid3 ocop:destinationAddress> ?dest_addr2 .
?uid3 ocop:destinationPort> ?port .
?uid3 ocop:hasProtocol> ?proto .
?uid3 <http://cs.org/p/hasRespBytes> ?rbytes2.
FILTER(?rbytes2 > 0)
?uid4 ocop:sourceAddress> ?ap_addr.
?uid4 ocop:destinationAddress> ?dest_addr .
?uid4 ocop:destinationPort> oco:port#9162.
?uid4 <http://cs.org/p/hasRespBytes> ?rbytes1.
FILTER(?rbytes1 > 0)
}
GROUP BY ?proto ?port
HAVING (?client_count > 1)
}
}
ORDER BY DESC(?client_count)
```

2) *Betweenness centrality*

This SPARQL code calculates betweenness centrality over a subgraph built from network flow data revealing chains of SSH sessions and some enrichment data. Lines 10-21 search for the first SSH sessions, and lines 24-35 search for any second SSH sessions. Lines 12-13 show the SCinet network security team included IP assignment in the graph with transactional data. Lines 25-27 show how intrusion detection alerts are merged with network flow data in lines 28-33. Lines 39-40 show how to call Cray's SPARQL extensions over the subgraph extracted in lines 6-38.

Written in approximately 40 minutes and executing in less than 1 minute in its finished form, this example, development time, and execution time also demonstrates the applicability of SPARQL and RDF to computer network analysis problems.

```
1.  PREFIX    ocop:   <http://opencog.net/p/>
2.  PREFIX    oco:    <http://opencog.net/>
3.
4.  SELECT ?vertices ?scores
5.  WHERE {
6.  CONSTRUCT{
7.  ?src_addr <urn:p/hasSSH> ?dest_addr.
8.  ?dest_addr <urn:p/hasSSH> ?dest_addr2 }
9.  WHERE {
10. SELECT distinct ?src_addr ?dest_addr ?dest_addr2
11. WHERE
12. {?booth2 a <http://sc14.org/class#SCinet_subnet> .
13. ?booth2 oco:hasMember ?dest_addr .
14. ?uid3 ocop:sourceAddress ?dest_addr .
15. ?uid3 ocop:destinationAddress ?dest_addr2 .
16. ?uid3 ocop:hasProtocol oco:proto#tcp.
17. ?uid3 ocop:destinationPort oco:port#22.
18. ?uid3 ocop:start ?start_time2.
19. ?uid3 <http://cs.org/p/hasRespBytes> ?rbytes2.
20. FILTER (?rbytes2 > 12000)
21. FILTER (?start_time < ?start_time2)
22. OPTIONAL
23. {
24. SELECT ?src_addr ?dest_addr ?start_time
25. {?uid <http://cs.org/p/hasNoticeNote> <http://cs.org/notice_node#SSH::Password_Guessing>.
26. ?uid <http://cs.org/p/hasNoticeMsg> ?msg.
27. ?uid <http://cs.org/p/hasOrigAddr> ?src_addr.
28. ?uid3 ocop:sourceAddress ?src_addr .
29. ?uid3 ocop:destinationAddress ?dest_addr .
30. ?uid3 ocop:hasProtocol oco:proto#tcp.
31. ?uid3 ocop:destinationPort oco:port#22.
32. ?uid3 ocop:start ?start_time.
33. ?uid3 <http://cs.org/p/hasRespBytes> ?rbytes2.
34. FILTER(?rbytes2 > 12000)
35. }
36. LIMIT 500
37. }
38. }
39. }INVOKE yd:graphAlgorithm.betweenness_centrality (.5,1)
40. PRODUCING ?vertices ?scores
41. }
42. ORDER BY DESC(?scores)
```

E. *Analytic findings from graph analysis during SC14*

Cray and the SCinet network security team used subgraph isomorphism, Jaccard similarity coefficient, and betweenness centrality to generate a number of analytic findings during SC14. This discusses the network security events that occurred during SC14 and how graph algorithms were used during the analysis of these events.

1) *Subgraph isomorphism*

The SCinet network security team used subgraph isomorphism/pattern search to detect outbound scanning and outbound SYN flooding behaviors.

Detecting outbound scanning can be performed using a graph concept called dispersion: the out-degree of a given node. In computer network analysis, dispersion is used with port and protocol combinations to show nodes exhibiting unusual behaviors in a specific area. Cray has used dispersion at SC13 and SC14 to successfully identify outbound scanning. At SC14, Cray identified infected clients on the wireless network scanning for vulnerable servers or sending malware out of the network

SYN flooding is a computer network attack where a client opens partial connections to a host and leaves them open. This attack is designed to remove a host from the network by exhausting its ability to accept inbound connections. If the host isn't on the network, it isn't accessible in the current "available-everywhere" networked-computing environment. At SC14, the network security team observed two clients on the network participating in SYN flood attacks. These two network events accounted for 86% of all network flow observed on the network. The flow count histogram is shown in figure 4 with the two SYN floods circled in red.

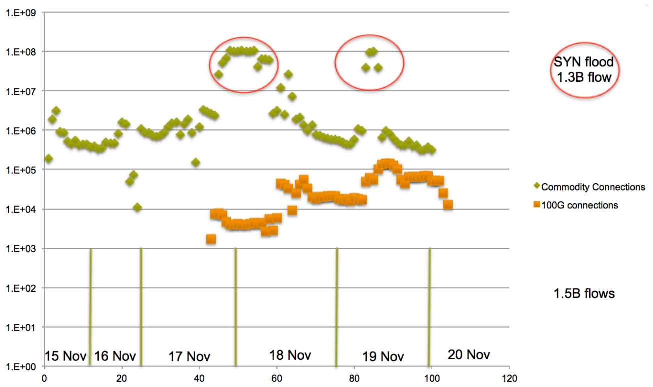


Figure 4. SCinet's network flow counts. Green diamonds are from 10GB/s commodity links, and orange diamonds are from 100GB/s sensors. The y-axis is logarithmic

2) *Jaccard similarity coefficient*

All of the network security team's tools identified these SYN floods. The team then looked for the root cause of the SYN floods. Cray combined alerts from one IDS system (showing a malware download on an odd port) with network flow and Jaccard similarity coefficients to identify likely infected client behaviors. This analysis revealed that port 7668 was most similar to the seed port of 9162. A visualization of clients exhibiting network activity on both of these ports is figure 5.

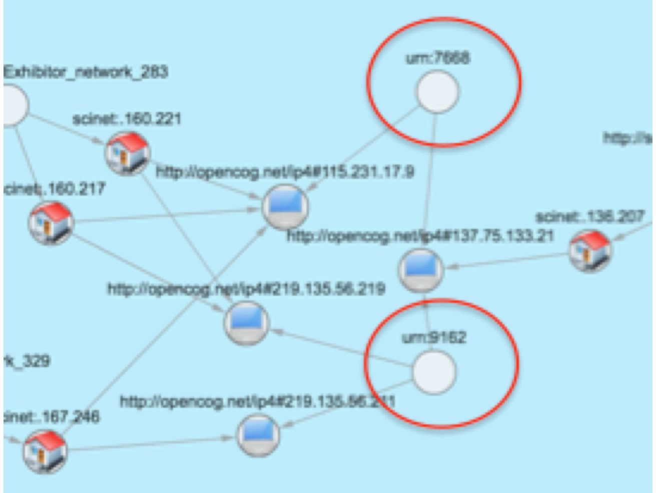

Exhibitor_network_283
urn:7668
scinet:.160.221
http://s
http://opencog.net/ip4#115.231.17.9
cinet:.160.217
scinet:.136.207
http://opencog.net/ip4#137.75.133.21
http://opencog.net/ip4#219.135.56.219
urn:9162
k_329
inet:.167.246
http://opencog.net/ip4#219.135.56.211

Figure 5. Jaccard similarity coefficient visualization for V1=9162 and V2=7668

The root cause of the infection was believed to be malware targeting Linux systems running SSH servers that infected the systems by brute-forcing passwords to accounts with SSH login privileges on the systems. Password brute-forcing is where an adversary connects to an SSH server and attempts to guess the password for an account on the system. The adversary repeatedly connects, guessing different passwords each time until they find one that works.

With the Linux malware in the environment, the network security team spent the four days of the conference identifying infected hosts using IDS tools and then notifying infected users. The team identified multiple hosts per day and visited multiple booths each day to inform them of their infection and recommend mitigations.

3) *Betweenness centrality*

The last day of SC14, the network security team used a subgraph isomorphism to build a subgraph of SSH sessions that seemed to originate from hosts that successfully brute-forced SSH passwords. This subgraph is displayed in figure 6. The network security team then applied betweenness centrality to the generated subgraph. This analysis revealed the host where system administrators and security staff should begin to mitigate the infection network. This host is circled in red in figure 6.

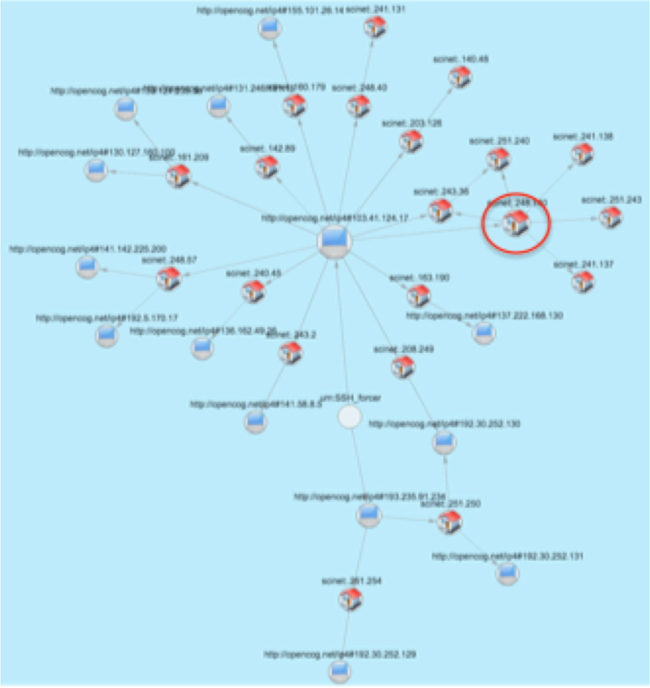

http://opencog.net/ip4#155.101.26.14
scinet: 241.131
scinet: 140.48
scinet: 248.40
scinet: 203.126
scinet: 251.240
scinet: 241.138
http://opencog.net/ip4#130.127.160.100
scinet: 161.209
scinet: 142.89
scinet: 243.36
scinet: 248.160
scinet: 251.243
http://opencog.net/ip4#103.41.124.17
http://opencog.net/ip4#141.142.225.200
scinet: 248.57
scinet: 240.45
scinet: 163.190
scinet: 241.137
http://opencog.net/ip4#137.222.168.130
http://opencog.net/ip4#192.5.170.17
http://opencog.net/ip4#136.162.49.26
scinet: 243.2
scinet: 208.249
http://opencog.net/ip4#141.58.8.5
urn:SSH_forcer
http://opencog.net/ip4#192.30.252.130
http://opencog.net/ip4#193.235.91.234
scinet: 251.250
http://opencog.net/ip4#192.30.252.131
scinet: 251.254
http://opencog.net/ip4#192.30.252.129

Figure 6. SSH connection subgraph and

IV. Applying Graph Analytics to Cyberthreat Analytics

Cray has supported computer network security analytic needs using graph algorithms at SC14 and through other engagements. These algorithms and approaches can also be applied to cyberthreat analytics when combined with graphs built from appropriate data sources. These data sources include published vulnerabilities and exploits, blacklists, and professional networking publications. Combining these data sets and applying graph algorithms allows enterprises to perform cyberthreat analytics earlier in the "Cyber Kill Chain."

V. Conclusion

Graph algorithms and analytic approaches are a new tool available to enterprises attempting to overcome the challenges presented by large, heterogeneous data sets that require flexible, responsive analytic execution. Cray, through its work as part of SCinet during SC14, applied graph algorithms to help answer computer network security analytic questions against large (18 billion triple), heterogeneous (multiple types of data) graphs using flexible analytics based on open standards (SPARQL / RDF). These approaches generated new analytic results and enabled SCinet's network security team to be more flexible and respond to myriad new threats in minutes to hours. These approaches can be easily adapted to use additional datasets and be applied by enterprises seeking to become more resilient against increasingly sophisticated adversaries and malicious activities.

Acknowledgment

E.D. thanks his family for giving him the time to participate in SCinet and SC14 in the two weeks before Thanksgiving.

References

[1] http://www.cray.com/products/analytics/urika-gd

[2] http://www.cisco.com/c/en/us/products/ios-nx-os-software/netflow-version-9/index.html

[3] Network Working Group. Hypertext transfer protocol (RFC 2616). Technical report, Internet Engineering Task Force, 1999.

[4] Network Working Group. Domain Name Service (RFC 1034). Technical report, Internet Engineering Task Force, 1987.

[5] Network Working Group. File Transfer Protocol (RFC 959). Technical report, Internet Engineering Task Force, 1985

[6] Network Working Group. Telnet Protocol specification (RFC 854). Technical report, Internet Engineering Task Force, 1987

[7] Network Working Group. WHOIS (RFC 954). Technical report, Internet Engineering Task Force, 1987

[8] Cisco 2014 annual security report, http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf

[9] Verizon 2014 Data Breach Investigation Report, http://www.verizonenterprise.com/DBIR/2014/

[10] J. Janies, Protographs: Graph-Based Approach to NetFlow Analysis; In Proceedings from FloCon 2011

[11] E. Dull, Discovering Unknown Network Activity Using Graphs and Computer Network Data; In Proceedings from FloCon 2014

[12] E. Dull, and S.P. Reinhardt, Network flow analysis at 880 Gb/s: Lessons learned from SCinet '11-'14; In Proceedings from FloCon 2015

[13] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. 2012. An in-depth comparison of subgraph isomorphism algorithms in graph databases. Proc. VLDB Endow. 6, 2 (December 2012), 133-144. DOI=10.14778/2535568.2448946

[14] E. Dull, R. Kartch, and R.W. Technintin, Semantic Representations of Network Flow: A Proposed Standard with the What, the Why, and the How; In Proceedings from FloCon 2015

[15] E. Jason Riedy, Henning Meyerhenke, David Ediger, and David A. Bader. Parallel community detection for massive graphs. In David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors, Graph Partitioning and Graph Clustering, volume 588 of Contemporary Mathematics, pages 207–222. American Mathematical Society, 2013.

[16] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". Numerische Mathematik 1: 269–271. doi:10.1007/BF01386390

[17] Collins, M. (2014). Network security through data analysis: Building situational awareness.

[18] Kolda, Tamara G; Procopio, Michael J; "Generalized badrank with graduated trust", Sandia National Laboratories, California, 2009

[19] Information technology -- Database languages -- SQL -- Part 1: Framework. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=63555

[20] M.E.J. Newman, "Modularity and community structure in networks", Data Analysis, Statistics and Probability (physics.data-an); Statistical Mechanics (cond-mat.stat-mech); Physics and Society (physics.soc-ph), Journal reference: Proc. Natl. Acad. Sci. USA 103, 8577-8582 (2006), DOI: 10.1073/pnas.0601602103, arXiv:physics/0602124

[21] Lockheed Martin, "Cyber Kill Chain", http://www.lockheedmartin.com/us/what-we-do/information-technology/cyber-security/cyber-kill-chain.html

[22] S. Harris and A. Seaborne. SPARQL 1.1 query language (W3C recommendation 21 march 2013). Technical report, World Wide Web Consortium, 2013. http://www.w3.org/TR/sparql11-query/

[23] RDF Working Group. Resource Description Framework (RDF). Technical report, World Wide Web Consortium, 2004. http://www.w3.org/RDF/.

[24] D. Mizell, "Introduction to the Cray XMT", http://wwwjp.cray.com/downloads/XMT-Presentation.pdf

[25] D. Gruhl, R. Guha, D. Liben-nowell, and A. Tomkins. Information diffusion through blogspace. In In WWW '04, pages 491–501. ACM Press, 2004.

[26] Network Working Group. Uniform resource identifier (URI): Generic syntax (RFC 3986). Technical report, Internet Engineering Task Force, 2005. http://tools.ietf.org/html/rfc3986.

[27] "SCinet overview", http://sc14.supercomputing.org/scinet

[28] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics, 3(2-3), 2005.

[29] "About the SC Conference Series", http://supercomputing.org/about.php

[30] "The BRO Network Security Monitor", https://www.bro.org/

[31] Network Working Group. The Syslog protocol (RFC 5424). Technical report, Internet Engineering Task Force, 2009.

[32] Network Working Group. The Secure Sockets Layer Version 3.0 (RFC 6101). Technical report, Internet Engineering Task Force, 2011

[33] https://sourceforge.net/projects/opencyberontology/

[34] Network Working Group. Transmission Control Protocol (RFC 793). Technical report, Information Systems Institute, University of Southern California, 1981

[35] Network Working Group. User Datagram Protocol (RFC 768). Technical report, Information Systems Institute, University of Southern California, 1980

[36] Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, http://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf

[37] Network Working Group. An Ethernet Address Resolution Protocol (RFC 826). Technical report, Internet Engineering Task Force, 1982

[38] Network Working Group. Internet Protocol (RFC 791). Technical report, Information Systems Institute, University of Southern California, 1981

[39] Network Working Group. The Secure Shell Protocol Architecture (RFC 4251). Technical report, Internet Engineering Task Force, 2006.

[40] John Feo. Graph Analytics in Big Data. Proceedings of HPC Graph Analytics workshop 2012 @ SC12.