Utilizing Network Science and Honeynets for Software Induced Cyber Incident Analysis

Napoleon C. Paxton
U.S. Naval Research Laboratory
napoleon.paxton@nrl.navy.mil

Dae-il Jang
Arizona State University
daelsmail@gmail.com

Stephen Russell
U.S. Naval Research Laboratory
stephen.russell@nrl.navy.mil

Gail-Joon Ahn
Arizona State University
gahn@asu.edu

Ira S. Moskowitz
U.S. Naval Research Laboratory
ira.moskowitz@nrl.navy.mil

Paul Hyden
U.S. Naval Research Laboratory
paul.hyden@nrl.navy.mil

Abstract

Increasing situational awareness and investigating the cause of a software-induced cyber attack continues to be one of the most difficult yet important endeavors faced by network security professionals. Traditionally, these forensic pursuits are carried out by manually analyzing the malicious software agents at the heart of the incident, and then observing their interactions in a controlled environment. Both these steps are time consuming and difficult to maintain due to the ever changing nature of malicious software. In this paper we introduce a network science based framework which conducts incident analysis on a dataset by constructing and analyzing relational communities. Construction of these communities is based on the connections of topological features formed when actors communicate with each other. We evaluate our framework using a network trace of the BlackEnergy malware network, captured by our honeynet. We have found that our approach is accurate, efficient, and could prove as a viable alternative to the current status quo.

1. Introduction

Today the importance of developing effective methods to analyze and defend against cyber attacks is no longer in doubt. Incidents such as recent data breaches at Target retail stores and the PF Chang restaurant chain as well as discovered and alleged attacks against Nation states, bring worldwide attention to the cyber attack problem [14, 9, 4]. This problem has been growing exponentially despite constant research geared towards reducing its impact. One of the major issues causing this upward trend in attacks is the dependence on outdated analysis systems [3]. Modern day attacks are often sophisticated and carried out at high speeds. In order to effectively understand and defend against these attacks, it is necessary to identify key situational attributes and actions quickly, before the originators of the attack can cover their tracks or attack other targets.

When dealing with attacks where malicious software (malware) directly interacts with a victim computer, current methods of cyber incident analysis involve two major steps. Step one is to discover and analyze the malware that was sent to the infected system. This is also called the static step and is typically a manual analysis. Step two is to run the discovered malware in a closed simulated network (sandbox) and evaluate its actions based on the intelligence that was learned in step one. This is called the dynamic step. In theory these two steps represent the most effective method to conduct a detailed analysis of the incident, if the analyst conducting the static analysis step is a highly competent expert in malware analysis. Unfortunately, modern malware continues to evolve and become increasingly complex. Because of this, a detailed static step analysis is likely to take a significant amount of time for even the most highly qualified analyst to complete [3]. Furthermore, in the case of the dynamic step most sandbox analysis systems have limited network simulation capabilities. For instance, in order to completely simulate an environment, the sandbox would need to be configured to include every item and option in the network that could be modified in any way [20]. This level of customization is difficult to develop or operate and thus unrealistic for any affordable simulation system. Moreover, malware attacks are intrinsically driven by situational performers (actors) and incorporating this notion into the dynamic and static steps introduces significant additional complexity into the assessment process.

We believe an approach that can improve the efficiency and effectiveness of analyzing cyber

1530-1605/15 $31.00 © 2015 IEEE
DOI 10.1109/HICSS.2015.619

incidents should be able to identify the important actors involved in the incident without requiring a detailed internal description of the actor. Awareness of actors' roles and locations would reduce the time necessary for manually analyzing complex malware code. We also believe an approach that enhances the effective static and dynamic steps needs to be able to identify and track the transactions that take place between actors. Beyond further characterizing attack malware, transaction attribution would also reduce the dependence on sandboxes for the discovery of behavioral characteristics. Community-based analysis from the multi-disciplinary field of network science provides techniques that can satisfy these requirements. Here actors and their interactions are modeled using graph theory, where each actor is represented by a graph vertex V and each interaction between actors is represented by a graph edge, E. Communities are then extracted based on the context of the discovered interactions [5]. One of the findings we present in this paper shows that discovered community structures and the relationships within those communities can contribute to the understanding of the network as a whole. This finding strengthens statements about the benefits of community detection made in other fields, such as biology and social networking [12, 16].

In this paper, we present and discuss our honeynet-based framework that captures network traces of cyber incidents, identifies actors within the traced network, and constructs communities based on the interactions of the discovered actors. Our initial evaluation of the framework is very encouraging. Distinct communities were extracted from the data and the discovered relationships within and outside of the communities strongly suggest the role of the actor and the purpose of the network.

The remainder of the paper is structured as follows. In Section 2 we discuss related work in the area of software-based cyber incident analysis, honeynet technology, and network community detection using network science methods. Section 3 discusses the components of the honeynet-based framework. In Section 4 we discuss the results of each module within the framework, and Section 5 concludes the paper.

2. Background

In this section we discuss previous research that focused on analyzing software induced incidents. We also provide a background on important elements that play a key role in our approach, such as honeynets, network science based community detection, and the dataset we collected and used for our evaluation of the framework's operation.

2.1. Related Work

In a recent survey on network-based botnet detection methods by Garcia et al. [7], the authors took an in-depth look at the most widely used and researched botnet detection tools available. In this review there was a discussion which highlighted the fact that bot detection mechanisms have different requirements than botnet detection mechanisms due to the difference in detecting one machine as opposed to a group of machines. Detecting a bot fits in the realm of this paper since a bot is software that induces the cyber incident. Of these methods, "BotMiner" [9], "BotSniffer" [10], "N-gram" [1], and "Tamed" [23], were similar to our method based on their detection approach. Our method is different from these and all the other approaches found in this review because once the malware is discovered using our system, it is grouped into communities, which can then be analyzed to determine relationships between actors located in the same community and throughout multiple communities. The methods presented in this survey employ clustering techniques in their detection algorithms. Clustering is conducted based on a metric of distance and does not take into account relationships [5].

In another recent review, which compared dynamic malware analysis techniques, Egele et al. conducted a study on methods that look to dynamically analyze malware in an attempt to reduce the time gap between discoveries of the malware to gaining intelligence from it [4]. In this survey the authors acknowledge that most forms of malware analysis still rely heavily on manual static based analysis. They also discuss the major forms of malware discovered on the Internet today. The methods discussed in this work seek to conduct an analysis of the malware without first performing the manual static analysis step. They also use clustering techniques and automated dynamic analysis reports to describe observed actions. These actions are then turned into behavioral profiles. Our approach also focuses on behaviors, but instead of using arbitrary or metric-based node clustering techniques, we use node relationship based community detection techniques which allows us to identify relationships for a more fine grained and relational analysis.

Two works which developed systems to detect and analyze software induced malware networks are "In Mining Botnet Behaviors on the Large-scale Web Application Community" [6] and "Botnet Detection Based on Traffic Behavior Analysis and Flow Intervals" [24]. In this research the authors used machine learning techniques to discover patterns within the network that attempted to explain botnet

behaviors. Our framework has the same goal of identifying patterns, but machine learning techniques suffer from their dependence on generating a training set of data. Also, this type of analysis does not make any connections between identified nodes of interest. Our approach uses characteristics of the connections between the nodes of interest to identify patterns.

2.2. Honeynet Technology Overview

Honeynets are networks composed of machines that are geared to attract and capture transactions of malicious users [11]. If one machine is configured to collect this data it is called a honeypot. This technology has proven very useful in studying and defending against malicious networks. In most cases a general honeypot with a simple vulnerability will receive many attack attempts within minutes. Virtually any analysis tool or method can be plugged into the honeynet in order to run analysis on the discovered attacks. Honeynets can be configured in a variety of ways, based on the desired level of interaction from the malicious software. Simple implementations seek only to capture traces of automated attacks which search for vulnerabilities within systems on a subnet. These types of honeynets are considered low interaction. Other forms of honeynets allow the malware to connect to locations outside of the network. These types of honeynets are considered high interaction. The connections are normally limited so as not to allow the malware to damage outside sources from the honeynet. These types of implementations can become very elaborate. It is possible to create a framework which is an exact replica of a production network. Creating such a honeynet would allow the system administrator or researcher to investigate the effect a piece of malware would have on the corresponding production network. The goal of a successful honeynet is to record data and discover patterns in malicious traffic, without alerting the attacker, in order to discover a way to render the attack useless. Researchers and security professionals have used these methods to identify and shutdown attacks from all over the world. For our research we implement a high interaction honeynet in order to discover enough information to track and record all the interactions of the botnet. Recording this information gives us details of the full botnet, which is needed in order to verify the validity of our approach.

2.3. Network Science Based Community Detection

Network Science is an inter-disciplinary field that studies the network representations of physical, biological, and social phenomena. It includes methods and theories from a wide range of fields. The basis of our approach, which is detecting the community structure of graphs, is a graph theoretic approach that fits into this area. A network is said to have community structure if the nodes of the network can be grouped into multiple sets of nodes, where the sets of nodes are more connected internally than externally [13].

Community detection is different from node clustering because community detection group nodes based on context and not just distance [5]. This fact becomes especially useful when considering node overlap. Node overlap is when a node is a member of two distinct communities at the same time [22]. By investigating the context and the semantics of both community memberships we can begin to understand the purpose of the memberships. We can also begin to understand the connections between the two communities.

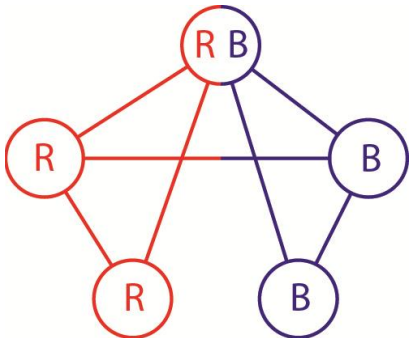

Figure 1: Three communities that overlap

There are several methods used to construct the communities. The most prevalent methods are hierarchy based [21], modularity (null model) based [18], information theory based [19], and clique based [17]. Each model has its drawbacks and advantages. In this paper we utilized the clique based method due to its natural ability to discover overlaps within communities. The clique based method forms communities by the percolation of fully connected adjacent sub-graphs [17]. Sub-graphs are said to be adjacent when they share *k-1* nodes. The major drawback to this approach is it is not considered the most efficient community detection approach, but recent tests show that it tested very well for networks up to several million [17]. This is sufficient for our initial tests, but we will explore optimizing our algorithm in future work.

2.4. BlackEnergy Botnet Dataset

We chose the BlackEnergy 1.8 Botnet Tool Kit for our malicious dataset analysis. The BlackEnergy Tool Kit functions using HTTP as its means to communicate with the command and control server. After a machine is infected, it initiates a POST message to a PHP script. This script is hosted on the command and control server, logs all information from each infected machine, and stores that content in a MySQL database. Bots that are controlled using HTTP each have their own separate connection channel to the command and control. This is different from bots that are part of IRC-administered botnets, which can view all other bot commands and transactions if they are part the same botnet channel. The dedicated communication channel makes analysis of the entire botnet more difficult and requires internal access of the command and control server. To gain this level of access, we first installed and executed the BlackEnergy toolkit on our honeynet, using a Windows based machine on VMware as the honeypot. Once an initial POST command was sent to the command and control server, we were able to use this information to masquerade as the command and control server and capture communications sent to it from other infected machines. Figure 2 shows the initial POST after executing BlackEnergy and figure 3 shows a second infected machine sending its initial POST to the command and control. In both figures the bots send the initial message in red, which is the POST request to stat.php. Stat.php is the php script that collects messages from bots on the command and control server. We also see the Host (anonymized here), which is the command and control IP address. The content length is also shown, which is the size of the message and the build_ID is also displayed. The build_ID is created when the bot is built and it consists of the SMB hostname of the infected system as well as the System Volume ID from the C:\ drive.

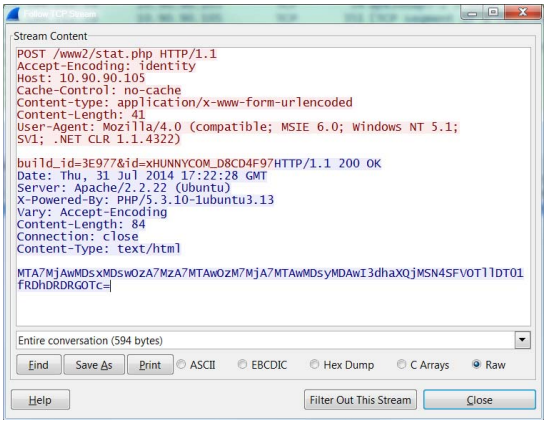

Figure 2: Initial Connect to Command and Control

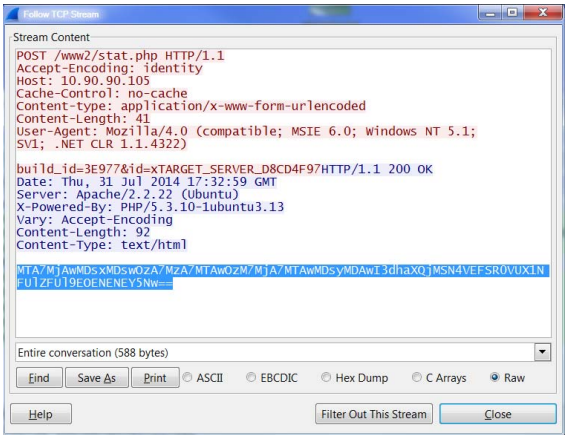

Figure 3: New Bot Connecting to Command and Control

3. Cyber Incident Analysis Framework

Our incident analysis framework is composed of five modules which can each act as a standalone method for malware analysis. In this particular application we are conducting an analysis of a botnet which is a network of computers that has been infected by agents. This network is controlled by one or more commanders, which are called botmasters or bothearders. We chose a botnet as our first test dataset because the important actors (Bots, Botnets, and Command and Control Centers) are well defined and network traffic concerning each actor can be detected by our honeynet. These factors made it a good choice for community detection. Figure 4 shows the framework.

3.1. Traffic and Log Collection Module

The traffic and log collection module is the entry point to the framework. It aims to capture and monitor network traffic at the edge of networks and at each sensor of installed honeypots within the honeynet. This module has two modes. It can capture live traffic from a network in the form of packets by using the PCAP library and it can load captured files stored on a file system. Currently the types of stored traffic that can be read with this module are PCAP, Netflow, Argus, and Sebek. Other forms of log data can be formatted to be included in the framework. Collected data is forwarded to the Traffic processing module.

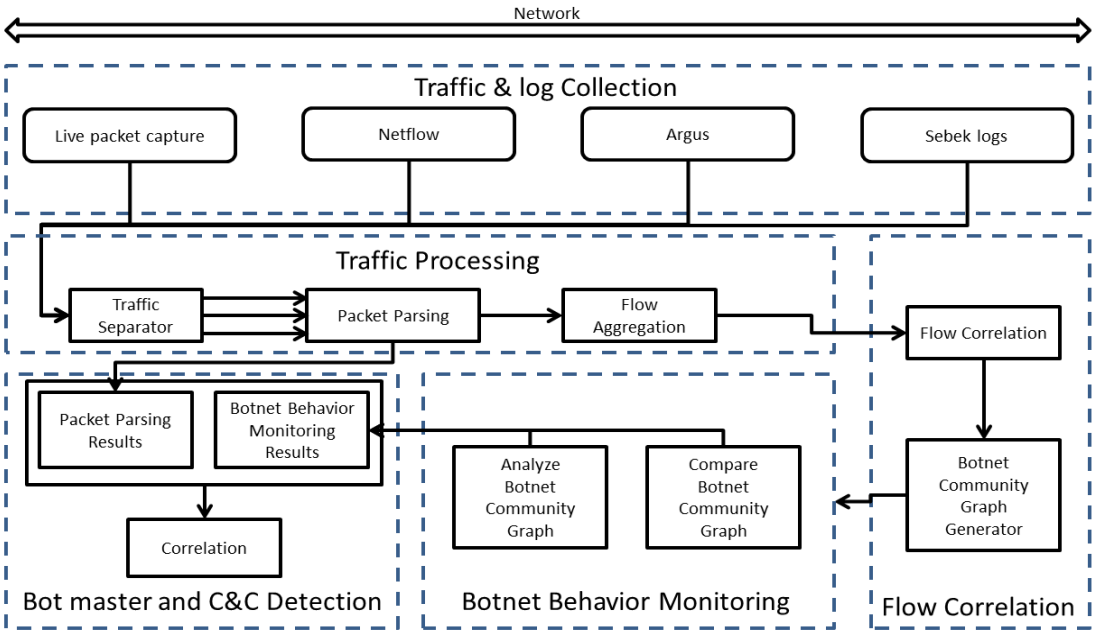

Figure 4: Community Based Cyber Incident Analysis Framework

3.2. Traffic Processing Module

The Traffic processing module aims to pre-process the traffic for flow correlation. The Traffic separator module within the component separates the traffic into 10 minute time windows in order for us to observe changes in behavior over time. The time window was set at 10 minutes because it is the default time programmed into the malware for connection intervals. In the future we will modify the time window and evaluate whether or not 10 minutes is the optimal value for community detection. The packet parsing module extracts header level information from the traffic such as source IP (SrcIP), destination IP (DstIP), source port, destination port, TCP/UDP payload size, source to destination packet count and data size, destination to source packet count and data size, and session interval. And this module generates plain text and comma separated value (CSV) files based on above information. The Flow aggregation module generates an input file for flow correlation.

3.3. Flow Correlation Module

The flow correlation module creates communities from the normalized data inputted from the flow aggregation module located in the traffic processing component. We use a clique based method to generate communities. In order to discover communities using cliques, we first needed to identify the nodes and edges. We modified the original *k*-clique percolation algorithm [17] to perform our analysis. Instead of identifying one entity to be a node, we added a secondary node. Particularly, for our analysis each IP address was equal to a "Primary" Node, and the message size was equal to the "Secondary" Node. The link between nodes is a tuple of {Time,SrcIP,DstIP,MsgLen}. For the purpose of constructing the communities, we do not consider direction. Once the communities were formed we use the tuple {SrcIP, DstIP} to add direction in order to identify the role of each Primary Node. The purpose of the Secondary Node is to identify all relevant communications. For instance, many botmaster communications only include the botmaster and the command and control node. By adding the message size as a secondary node, we are able to separate each meaningful communication into a community even if it only contains two nodes. Figure 5 shows three communities constructed using our method. The first two communities are composed of a botmaster connecting to the command and control with different message lengths and the third community shows 2 bots connected to a command and control using the same message length. Since the message length is the same, it becomes the node that connects the two bots into the same community.

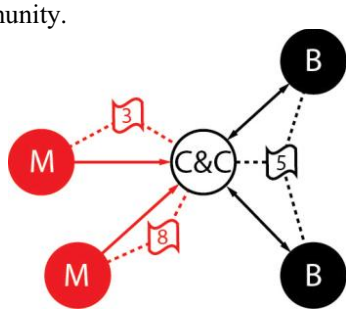

Figure 5: Three communities

The botnet community graph generator component inside the flow correlation module produces a graph of the communities within a 10 minute time period. This time interval was set based on the default time interval for bots to poll the command and control server. The graph is then passed to the botnet behavior monitoring module.

3.4. Botnet Behavior Monitoring Module

In the botnet behavior monitoring module we use a custom python script to analyze the current community graph and then compare it to previous graphs to determine comparisons that have occurred over time. Here we can discover the evolution of nodes and fluid relationships within the same community and across multiple communities.

3.5. Bot Master and C&C Detection Module

The Bot master and C&C detection module is where the results from the other modules are combined and correlated to produce intelligence.

The botnet behavior modeling results component gives a display of what has changed over time within the community graphs.

The packet parsing results module captures and displays parsing results from each data source that was used in the analysis. This module was included to provide the analyst with a lens into the lower level data in case there is a question about the validity of the monitoring results. For instance, the full packet from PCAP data is available here. By including the payload of the monitoring results, which is based on flows, we add context to the analysis by being able to drill down into the payload content. The analyst can also manipulate the data before correlation is facilitated.

The correlation module adjusts the output of the behavior monitoring results depending on what has changed in the packet parsing results. If no change has been made the correlation results will be identical to the input for the behavior monitoring results. If a change has been made, the results will change accordingly. An example change would be an inaccuracy that the analyst notices. The significance of this module is, if there are minor issues with the analysis they can be corrected here instead of rendering the entire analysis useless.

4. Framework Evaluation Results

Our honeynet captured live PCAP traffic to and from the command and control servers. In our initial experiment, connections between nodes were identical for each of the collection components. Because of this, we will only discuss the results in terms of PCAP traffic. After capturing and processing the data in the ***Traffic and Log Collection Module*** and the ***Log Processing Module*** respectively, our data was analyzed by the ***Flow Correlation Module*** and full PCAP data was sent to the ***Botmaster and C & C Detection Module***.

In the ***Flow Correlation Module***, the component, *Botnet Community Graph Generator* runs the community detection algorithm to discover graphs of communities within the data. Table 1 shows details of the results.

Table 1: Community Graph Metric Values

Metric	Value
Analysis Period (hrs)	168
Community Graphs	1008
Communities	3120

As shown in table 1, the analysis period was approximately 168 hours, which is also 7 days of analysis. Since the Flow Correlation Module generates community graphs in 10 minute increments, 1008 community graphs were generated. Within those graphs we observed 3120 communities.

The ***Botnet Community Graph Module*** then analyzes the details of the communities in each graph using the *Analyze Community Graph Component*. Table 2 shows the details of one of the community graphs. Communities and each node within every community are delineated using a period. Table 2 shows the details of community graph 128. There are 4 communities within the 10 minute community graph. Communities 128.1, 128.2, and 128.4 each have 3 nodes within it. Community 128.3 has 24 nodes within it.

Table 2: Community Graph 168

Community	Nodes	Overlap
128.1	3	{128.1.1:128.2.1;128.4.1}, {128.1.2:128.2.2;128.3.2;128.4.2}
128.2	3	{128.2.1:128.1.1;128.4.1}, {128.2.2:128.1.2;128.3.2;128.4.2}
128.3	24	{128.3.2:128.1.2;128.2.2;128.4.2}
128.4	3	{128.4.1:128.1.1;128.4.1}, {128.4.2:128.1.2;128.3.2;128.2.2}

The overlap of communities 128.1, 128.2, and 128.4 show that they are closely related. For instance, node 128.1.1 is the same node as 128.2.1 and node 128.4.1. The second component within the Botnet Community Graph Module is the *Compare Botnet Community Graph Component*. In this component the current incoming community graph is compared with all the previous community graphs. The overall node overlap is very significant within the botnet (84%). This was an expected finding since most attacks and other communications between bots and the command and control server are coordinated. Previous research found that most connections on the Internet have a very low overlap rate amongst nodes [22]. Since we have discovered that botnets have a high overlap rate, our finding could prove to be significant if we can use the presence of this high overlap as a detection tool. We will explore this in future research.

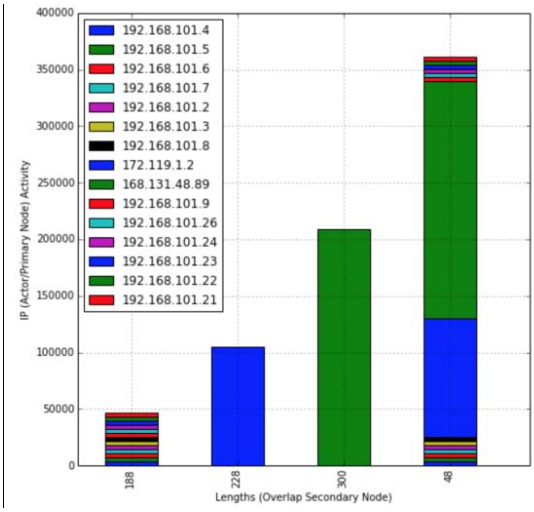

Figure 6: G-test of top 15 actors based on aggregate message sizes

Our ***Botmaster and C&C Detection Module*** attempts to discover the role of each node within the communities. In the *Botnet Behavior Monitoring Results Component*, we find relationships in the flow data that was collected. Each node is grouped based on overlap. We then use the packet payloads within the *Packet Parsing Results Component* to discover the relationships between overlapping nodes.

To evaluate the significance of community overlap we conduct a g-test on the 15 most active actors in the botnet that sent messages to a command and control server. A g-test is a measure of fitness to a distribution of data. We use it to discover correlations between actors and the overlap nodes. In our test active actors refer to aggregated IP length sizes. Figure 6 displays the g-test. Each IP length is an overlap node since the IP length is used as the secondary node that connects the primary nodes (IPs) as shown in figure 5. The results showed that the 15 most active actors only sent 4 message types of sizes (188, 228, 300, 48). The varying activity of the actors highlight their roles. IP length size 228 was only sent by actor (192.168.101.4) and size 300 was only sent by actor (192.168.101.5). This suggests a lack of coordination, which leads us to believe these two actors are botmasters. IP length size 188 was used by the other 13 actors equally. This suggests the activity was coordinated which leads us to believe these actors are bots. Length size 48 was correlated with each of the actors, but actors (192.168.101.5 and 192.168.101.6) had uneven distributions while the other 13 actors had an equal amount of data sent. This is further evidence that the 2 uncoordinated actors are botmasters and the 13 coordinated actors are bots.

To evaluate the ability to discover the overall purpose of the botnet using our framework we investigate the content of the overlap node messages. Message 48 was the basic "GET // HTTP" message sent when a node refreshes its command and control administration information. Message 228 and message 300 are Distributed Denial of Service commands being sent to the command and control server. When the bots check back with the server they will get the command to attack a target. These four overlap nodes represent ~85% of the data the dataset. By correlating these nodes we can reasonably assume this botnet is being used specifically for DDoS attacks.

Highly overlapping nodes, that were not command and control nodes, were considered to be bots. This is consistent with previous research which discovered that flash crowds of a high percentage represented a coordinated event caused by bots [9, 10]. Figure 7 shows a screen shot of our web-based community detection analysis tool with the roles assigned to each node.

6. Conclusion

In this paper we introduced a cyber incident analysis framework which is based on the detection of communities within discovered attack data. This framework does not need to perform a time consuming manual analysis step or a closed system dynamic analysis step to identify intelligence from the data. Instead, all that is needed is high level identification data and knowledge of communications between the identified actors. Correlations between the overlap nodes provided intelligence about the actors within the malicious dataset as well as the overall intent of the botnet. Coordinated activity suggested the presence of bots within the network, and a high volume of

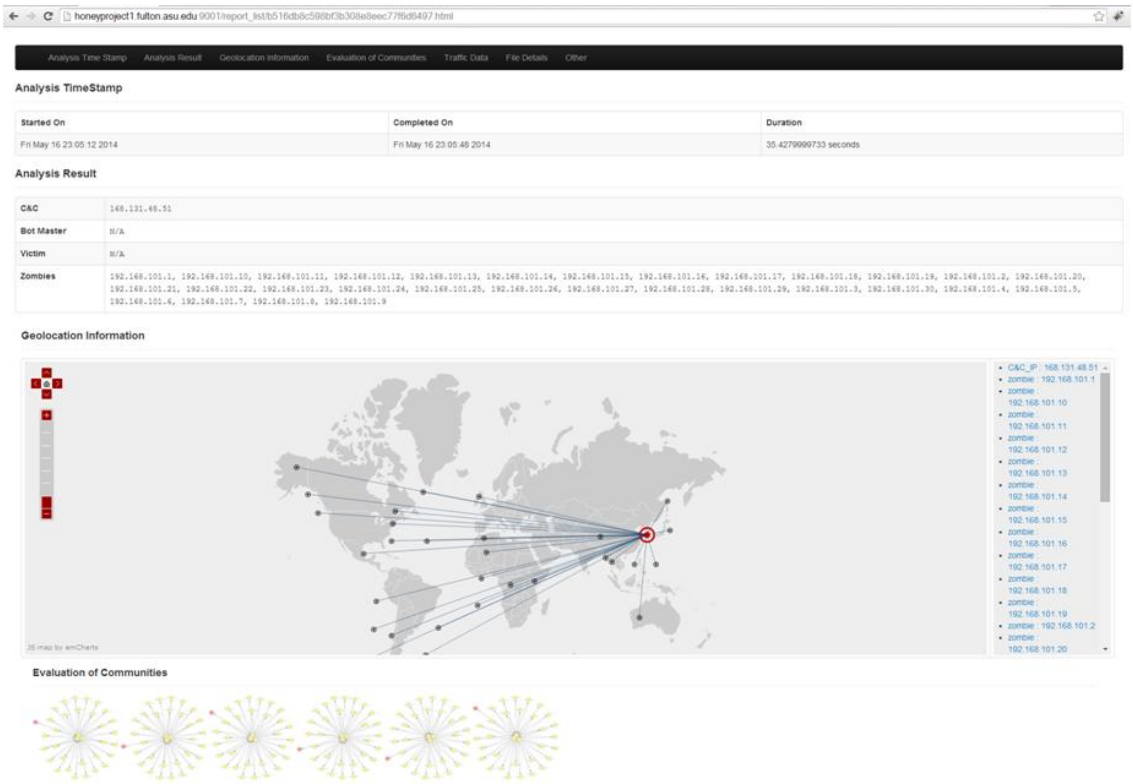

Figure 7: Screen shot of web-based analysis tool

uncoordinated activity, but membership within the community suggested the presence of botmasters. These correlations also revealed a high overlap percentage within the botnet, which is a significant contrast with normal Internet traffic. Since all botnets are defined by their coordinated activity, we hope to use this trait as an identifiable metric for a general community detection algorithm for botnets. Future revisions of our methods aim to detect botnets without initiating contact.

Cyber incident analysis in general includes the gamut of cyber attacks conducted on a computer network. One thing that all cyber incidents have in common is unwanted activity on computer networks. Community detection provides a way to identify patterns in network traffic based on changes, so we believe we can modify our techniques to analyze other datasets besides botnets. For example, when malware is installed on a system, processes are created, registry entries are added or changed, and files are also added or changed. Currently we are investigating using these changes to files, processes, and registry entries as nodes and the similarities between them as links. If successful, we believe our approach will greatly improve the current process of incident analysis.

7. References

[1] Abou-Assaleh, T., Cerone, N, Keselj, V, and Sweidan, R., "N-gram-based Detection of New Malicious Code.", In Proceedings of the 24th Annual International Computer Software and Applications Conference (COMPSAC 2004), Hong Kong, 2004

[2] Codenomicon, "The Heartbleed Bug", http://www.heartbleed.com, April, 2014

[3] Dittrich, D., "So you want to take over a botnet", In Proceedings of 5th USENIX conference on Large-Scale Exploits and Emergent Threats, USENIX, 2012

[4] Egele, Manuel, Scholte, Theodoor, Kirda, Engin, and Kruegel, Christopher, "A Survey on Automated Dynamic Malware Analysis Techniques and Tools, ACM Computing Surveys, Vol. 44, No. 2, 2012

[5] Fortunato, S., "Community detection in graphs", Physics Reports, 2010

[6] Garant, Daniel and Lu, Wei, "Mining Botnet Behaviors on the Large-Scale Web Application Community", In Proceedings of 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA), IEEE, 2013

[7] Garcia, S., Zunino, A., and Campo, M., "Survey on network-based botnet detection methods", Security and Communication Networks, May 2014

[8] Geers, K., Kindlund, D., Moran, N., and Rachwald, R., "World War C: Understanding Nation-State Motives Behind Today's Advanced Cyber Attacks", http://www.fireeye.com/resources/pdfs/fireeye-wwc-report.pdf, 2013

[9] Guofei, G., Perdisci, R., Zhang, Junjie, and Lee, Wenke, "BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection, USENIX, 2008

[10] Guofei, Gu, Zhang, Junjie, and Lee, Wenke, "Botsniffer: Detecting botnet command and control channles in network traffic.", In Proceedings of the 15th Annual Network and Distributed System Security Symposium, NDSS, 2008

[11] Honeynet Project, "Know Your Enemy: GenII Honeynets", http://old.honeynet.org/papers/gen2/, 2005

[12] Jia, Y., Garland, M., and Hart, J.C. "Social Network Clustering and Visualization using Hierarchical Edge Bundles", Computer Graphics Forum, December 2011

[13] Lancichinetti, Andrea and Fortunato, Santo, "Community detection algorithms: A comparative analysis", Phys. Rev. E 80, 056117, Novermber 2009

[14] Krebs, B., "The Target Breach, By the Numbers", http://krebsonsecurity.com/2014/05/the-target-breach-by-the-numbers/, May 14, 2014

[15] Krebs, B., "P.F. Chang's Breach Likely Began in Sept. 2013", http://krebsonsecurity.com/2014/06/p-f-changs-breach-likely-began-in-sept-2013/, June 14, 2014

[16] Leydesdorff, L. and Ahrweiler, P., "In Search of a Network Theory of Innovations: Relations, Positions, and Perspectives", Journal of the American Society for Information Science and Technology (JASIST), 2013

[17] Palla, Gergly, Derenyi, Imre, Farkas, Illes, and Vicsek, Tamas, "Uncovering the overlapping community structure of complex networks in nature and society", Nature, June 2005

[18] Perry, Patrick and Wolfe, Patrick, "Null Models for Network Data", Available at http://arxiv.org/abs/1201.5871v1, 2012

[19] Rosvall, Martin and Bergstrom, Carl, T., "An information-theoretic framework for resolving community structure in complex networks", Proceedings of the National Academy of Sciences of the United States of America, 2007

[20] Rossow, C., Dietrich, C., J., Bos, H., Cavallaro, L., Steen, M., Freiling, Felix, C., and Pohlmann, N., "Sandnet: Network Traffic Analysis of Malicious Software", ACM, 2011

[21] Wang, Jianxin, Li, Min, Chen, Jianer, and Pan, Yi, "A Fast Hierarchical Clustering Algorithm for Functional Modules Discovery in Protein Interaction Networks, Computational Biology and Bioinformatics, IEEE/ACM Transactions, 2011

[22] Xie, J., Kelley, S., and Szymanski, B., "Overlapping community detection in networks: the state of the art and comparative study. In Social and Information Networks, ACM, 2012

[23] Yen, Ting-Fang and Reiter, Michael, K., "Traffic Aggregation for Malware Detection", Detection of Intrusions and Malware, and Vulnerability Assessment Lecture Notes in Computer Science, Volume 5137, 2008

[24] Zhao, David, Traore, Issa, Sayed, Bassam, Lu, Wei, Saad, Sherif, Ghorbani, Ali, and Garant, Dan, "Botnet detection based on traffic behavior analysis and flow intervals", 27th IFIP International Information Security Conference, Computers and Security, Volume 39, Part A, November 2013