Scalable Attack Propagation Model and Algorithms for Honeypot Systems

Ariel Bar, Bracha Shapira, Lior Rokach and Moshe Unger
Department of Information Systems Engineering and Telekom Innovation Laboratories
Ben-Gurion University of the Negev
Beer-Sheva, Israel
{arielba, bshapira, liorrk, mosheun}@bgu.ac.il

***Abstract*—Attack propagation models within honeypot systems aim at providing insights about attack strategies that target multiple honeypots, rather than analyzing attacks on each honeypot separately. Traditional attack propagation models focus on building a single probabilistic model. This modeling approach may be misleading, since it does not take into consideration contextual information such as the country from which the attack is initiated. In addition, with the massive increase in the magnitude of attacks on honeypots, a scalable modeling approach is required.**

In this work we present a novel attack propagation model that can utilize contextual information about the attacks by training multiple Markov Chain models. Moreover, we add additional layers of analysis: first, we present a likelihood estimation procedure that can identify new and evolving attack patterns; and second, we introduce a method for generating simulated attack sequences that can be used for training or sensitivity analysis. Lastly, we present, in details, a MapReduce design for all suggested algorithms in order to address scalability issues. We evaluate our methods on a massive dataset which includes approximately 170 million attacks on an operational honeypot system. Results indicate that contextual modeling is important for explaining attack propagation that may vary by country. In addition, we show the effectiveness of the suggested method for generating simulated sequences by comparing the attack propagation patterns we learned in the generated dataset and the original one. Finally, we demonstrate the scalability of all of the proposed algorithms on real and synthetic datasets that include over a billion records.

Keywords- Cyber Security; Honeypots; Attack Propagation; Markov Chains; Random Walk; MapReduce

I. INTRODUCTION

Honeypots are computer resources which are designed to attract attackers by presenting false or misleading information that an attacker will want to obtain, attack, or control. Attackers that compromise the honeypot system leave valuable tracks about their attack methods that can be further analyzed by security experts in order to discover common and new attack patterns. In recent years, we have observed a significant increase in the volume and variety of data collected by honeypots. When first introduced, honeypot initiatives such as the Honeynet Project [1] observed a limited amount of traffic and generated small datasets of several megabytes per day. Nowadays, frameworks such as T-Pot[1] may observe millions of attacks per day and generate gigabytes of summarized metadata on those attacks. The large scale of data collected raise new challenges in designing effective and efficient solutions for analysis of the massive amount of data collected by honeypot systems.

In this work we present a scalable framework for modeling attack propagation patterns within honeypot systems. Propagation patterns between honeypots are assumed to occur in situations in which the same attacker attacks or scans for vulnerabilities several honeypots under the same circumstances, e.g., within a one hour time frame. The propagation of attacks within a honeypot system is most likely to occur in the context of scan activities, worm activity, or systematic planned attacks. In particular, we utilize, formalize, and extend the probabilistic propagation model presented in [2]. The proposed modeling approach is based on Markov chains modeling [3], and we present several algorithms for handling three types of tasks: 1) Training contextual Markov chain models that capture propagation patterns within a honeypot system. 2) Estimating the likelihood of observed attacks according to the trained models. 3) Generating synthetic attack sequences that follow the characteristics of the propagation patterns. To the best of our knowledge, this is the first work that addresses the scalability issues of attack propagation models within honeypots, and the first to include contextual modeling, maximum likelihood estimation, and synthetic data generation components in the model.

The proposed framework is implemented in the MapReduce[4, 5] programing model. The first two tasks in the suggested framework, which concern training Markov models and likelihood estimation are ideal for the MapReduce model, since they require processing a large amount of available data. However, the task of generating simulated sequences is unusual for MapReduce frameworks. In this scenario, we need to *create* an enormous dataset from the beginning, as very little data is available from start.

We evaluate the effectiveness and efficiency of the proposed algorithms using several massive datasets. We first analyze a real-world dataset of cyber-attacks on a honeypot system. The dataset contains 114G of data, with approximately 170 million atomic attacks collected for over a year. Second, we analyze synthetically generated datasets containing up to 100 million sequences and over a billion atomic records in various configurations. Results in all

[1] http://dtag-dev-sec.github.io/

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

experiments are consistent and demonstrate the high scalability of the proposed solution.

The rest of this paper is organized as follows: in Section II we present related work on analytical attack propagation models within honeypot systems; Section III includes a detailed review of the proposed methods and algorithms. Empirical evaluation is described in Section IV; and finally, in Section V we conclude our work and outline several directions for further research.

II. Related Work

One of the main research areas in honeypot technology concerns with utilization of the output data collected by the honeypots [6]. The propagation model suggested in [2] is designed to detect attack propagations which occur when the IP address of an attacking machine is observed on multiple honeypots. This scenario is most likely to occur in the context of scan activities or worm propagation. Propagation from honeypot HP_i to honeypot HP_j is assumed to happen if the same attacker attacks HP_j immediately after HP_i. The suggested model is a propagation graph, where each node represents a honeypot; a link between two nodes represents propagation between two honeypots, and each link is assigned with a transition probability. This model serves as the basis for our work, and we extend by formalizing it as contextual Markov chain model and adding additional layers of analysis.

Previous works [7, 8] suggested classification models which aim to distinguish between two families of malicious web sessions: "attacks" and "vulnerability scans". The data was collected by honeypots, while the main modeling process included feature extraction and training supervised machine learning models that classify whether a session is an attack or a scan activity. We can consider these types of works parallel and complementary to ours, since scan activities are an example of attack propagation.

Analysis and detection of polymorphic worms may also be considered parallel and complementary to our work in the same vein. Two examples of such works can be found in [9, 10]. The first work identified signatures of polymorphic worms by examining special tokens (substrings) in the worms' payload data; while the latter identified worm signatures by examining sequences of operations on the SMB protocol that were triggered by the worm.

On a broader perspective, the suggested algorithms concerns with scaling up Markov chain models. The mechanisms that are presented in the sugeested algorithms can be applied in domains that utilize Markov chain models such as in chemistry [11], biology [12], computer science [13], and others. Many of these works also indicate that handling large amounts of data is a challenging task so a scalable approach such as presented in this work may be beneficial to these works as well.

III. Method

A. Learning Contextual Propagation Models

The first task in the proposed modeling approach is to learn a set of Markov chain models that capture the propagation patterns. A key concept in the suggested modeling approach is to train a set of Markov chain models rather than a single global one. The main assumption behind the modeling approach is that propagation patterns may vary according to the context in which the attacks occur. Examples of such contextual features may include the time of the attack and the country that originated those attacks.

In essence, we model sequences of attacks on the honeypot system where an attacker attacks one honeypot after another. Given a dataset of atomic attack records on the honeypot framework, where each record contains the source IP of the attacker, a timestamp, and the attacked Honeypot identifier, we model these attacks according as a Markov process. We start with defining an attack session as all atomic attack records from the same attacker in a limited time interval. The attacks in a session are then ordered according to their timestamps. Attackers are identified by their source IP within a time window of 1 hour in order to handle scenarios of dynamic IP changes. Next, for each attack session we assign the relevant contextual features such the contrary that originated the attack sessions using Geo-Location APIs. Once the attack sequences are formed, the honeypot IDs, denoted by HP_{id} are considered as states in the Markov chain model.

The most demanding computational process of learning Markov chain models is calculating the estimated probabilities of the transition matrix and starting states probability vector. These probabilities can be estimated by scanning the observed data: Given a state space with cardinality S, the probability a sequence will start at state i (denoted as st_i) is presented in equation 1. The probability is equal to the number of sequences which started at state i divided by the total number of sequences. Assuming that each sequence starts from a certain state $j \in S$, the total number of sequences is equal to the sum of the starting state counters over all $j \in S$.

In the same way, equation 2 describes the required calculations for estimating the transition matrix's probabilities. The probability of moving from the current state i to state j (denoted as p_{ij}) is equal to the number of observed transitions from state i to state j, divided by the number of all observed transitions from i to any state $k \in S$.

$$st_i = \frac{\#sequences\ starting\ at\ state\ i}{\sum_{j \in S} \#sequences\ starting\ at\ state\ j} \quad (1)$$

$$p_{ij} = \frac{\#transitions\ from\ state\ i\ to\ j}{\sum_{k \in S} \#transitions\ from\ state\ i\ to\ k} \quad (2)$$

In order to learn contextual attack propagation models, we present two MapReduce algorithms. The first algorithm serves as a preprocessing step which is designed to build sequence records from atomic ones. The second algorithm is responsible for calculating all the required statistics for the Markov models.

The input for Algorithm 1 is a dataset of atomic records, when each record contains an $objectId$ (identifies the monitored entity), a $stateId$ (identifies the status of the monitored entity) and a $timestamp$ specifying when this

record was collected. The atomic attack records on the honeypots are easily mapped to this schema: the IP address of the attacker and the time interval of the observed attacks are mapped to $objectId$, the attack time is mapped to $timestamp$ and the attacked honeypot is mapped to the $stateId$.

The output of this algorithm is a dataset of sequences modeled in a key-value format, where the key is the $objectId$ and the value is the sequence associated to it. Sequences are modeled as a list of $timestamp$, $stateId$, and the original data record triplets, ordered chronologically by the timestamp. The map function (lines 1-5) works on each atomic record and emits intermediate key-value pairs, while the reduce function (lines 6-8) collects pairs with the same $objectId$ and builds the sequence.

The process of calculating the Markov chain models statistics is described in Algorithm 2. The input for the algorithm is a dataset of sequences, in a key-value format, where the key is an $objectId$ and the value is the sequence. The output of the algorithm is a sparse representation of all count-statistics for each of the contextual Markov chain models as required for equations 1 and 2. In particular, the results is a set of key-value pairs, where the keys are composed of $context$, $source$ and $destination$ triplets that indicate that transitions between these two states have been observed under the relevant context. The associated value for each key is the number of observed/relevant transitions. The map function (lines 1-8) emits key-value pairs of all the observed transitions in the current sequence: First, we extract all the relevant context values of the current sequence (lines 2-3). Second for each of extracted contexts, we emit all the observed transitions with a value of 1.0 (lines 4-7). Notice that the first observed stated id uses a wild card token denoted by "*" that specifics this is the beginning of the sequence in order to be consistent with the rest triplet structure. The reduce function simply sums all the values that are associated with same key (lines 8-9).

Algorithm 1: Pre-Processing

Input: S – Dataset of atomic data records
Output: Sequence dataset.

1. **function**: **MAP** Preprocess ($s \in S$)
2. $objectId \leftarrow$ extract object id from s
3. $timestamp \leftarrow$ extract timestamp from s
4. $stateId \leftarrow$ extract state id form s
5. emit $(objectId, (timestamp, stateId, s))$
6. **function**: **REDUCE** Preprocess ($objectId$, $[(timestamp, stateId, s)]$)
7. $sortedList \leftarrow$ sort by timestamp all triplets in $[(timestamp, stataId, s)]$
8. $emit\ (objectId, sortedList))$

Algorithm 2: Building Markov Chain

Input: $Sequences$ – Dataset of sequences
Output: Key, value pairs of all the required counting statistics for the Markov chain Model.

1. **function**: **MAP** BuildMarkov ($seq \in Sequences$)
2. $CF \leftarrow \emptyset$
3. $CF \leftarrow$ extract all contextual features from seq
4. For each $c \in CF$
5. $emit((c,*, seq_{stateId[1]}), 1.0)$
6. For $id = 2\ to\ n$ // n is the length of seq
7. $emit((c, seq_{stateId[id-1]}, seq_{stateId[id]}), 1.0)$
8. **function**: **REDUCE** BuildMarkov $((c, s_{src}, s_{dst}), cnt_1, cnt_2, \dots)$
9. $emit\ ((c, s_{src}, s_{dst}), \sum cnt_i)$

After computing all the transition count statistics that resulted from Algorithm 2, we can compute for each context c the associated Markov chain model probabilities according to equations 1 and 2. Given context c, the number of sequences starting at state st_i is the value associated to the key $(c,*, st_i)$, and the number of transitions between states st_i and st_j is the value associated to the key(c, st_i, st_j).

B. Likelihood Estimation

Given an attack propagation model, we apply a likelihood estimation procedure in order to estimate whether a new observed attack sequence in length n is common or not. The likelihood is calculated by series probabilities' multiplications:

$$Likelihood(s) = st_i \prod_{i=2}^{n} p_{i-1,i} \quad (5)$$

In order to avoid a series of long multiplications, a more convenient way is to calculate the log likelihood of a sequence:

$$LogLikelihood(s) = log(st_i) + \sum_{i=2}^{n} log(p_{i-1,i}) \quad (6)$$

A sequence with a small log likelihood measure indicates that it does not fit to the learned probabilities in the model, and may be considered abnormal. We extend this procedure to a set of contextual Markov chain models by repeating it for each of the models. Moreover, we treat each of contextual models as a separate expert and combine their likelihood estimations using a predefined aggregation function. For example, if we observed an attack sequence that originated from "China" on a "weekend", we can take the likelihood estimation of the two relevant contextual models and return the maximum of the two.

In Algorithm 3 we present a MapReduce algorithm for calculating the log likelihood of sequences given a series of

contextual Markov chain models. The input for algorithm is a set of sequences (denoted as $Sequences$). In addition several global parameters are shared cross the algorithm: MC is the set of contextual Markov chain models; $f()$ is the aggregation function for estimating the combined likelihood; and ϵ is a very small probability value which replaces starting states or transition probabilities which are equal to 0. The output of the algorithm is a series of log likelihood estimations ($MLEAggSet$) – one for each of the models in MC, and additional one for the combined model using $f()$ as an aggregation function. In addition, the algorithm reruns the number of sequences in each context for calculating the average log likelihood per sequence.

In particular, the map (lines 1-13) function takes as an input single sequence, extracts the relevant contexts and calculates the matching log-likelihood estimations according to equation 6. In addition, we calculate the log estimation using the combined model using $f()$ and a constant token "$C\#$". The reduce (lines 14-15) function sums separately the likelihood estimation associated for each key/model.

Algorithm 3: Likelihood Estimation

Input: $Sequences$ – Dataset of sequences.
Global:

- $MC_{[1..C]}$ – Set of Markov Chain models for each contextual value (|C| Markov models).
- ϵ – Minimum transition/starting state probability.
- $f()$ – Aggregation function.

Output: MLE aggregative values on the observed data.

1. **function**: **MAP** Likelihood ($seq \in Sequences$)
2. $CF \leftarrow \emptyset$
3. $CF \leftarrow$ extract all contextual features from seq
4. Initialize $MLEAggSet$
5. For each $c \in CF$
6. $lh = Max\,(MC_c.st_{seq_{stateId[1]}}, \epsilon)$
7. $MLEAggSet_C \mathrel{+}= log(lh)$
8. For $id = 2\ to\ n$ // n is the length of seq
9. $lh = Max\,(MC_c.p_{seq_{stateId[i-1]}, seq_{stateId[i]}}, \epsilon)$
10. $MLEAggSet_C \mathrel{+}= log(lh)$
11. For each $c \in CF$
12. $emit\,\left(c, \left(MLEAggSet_{c,}, 1.0\right)\right)$
13. $emit("C\#", (f(MLEAggSet), 1.0))$
14. **function**: **REDUCE** Likelihood $\left(c,\ (mle_1, cnt_1), (mle_2, cnt_2) \ldots\right)$
15. $emit\ (c, (\sum mle_i, \sum cnt_i))$

C. Random Walk Simulation

A random walk is a formalization of a path that consists of a succession of random steps on a certain distribution. The process of performing a random walk on a Markov chain model is straightforward: first, one needs to sample a random starting state aligned to the starting states probability vector; second, in an iterative process, sample the next state according to the current one and with respect to the probabilities on the transition matrix. This process can be repeated many times, and as a result, generate a large set of simulated sequences. Designing a MapReduce algorithm for this task is challenging since unlike data processing algorithms, the goal of this algorithm is to *create* a massive dataset of N sequences, given a relatively compact Markov chain model.

The main idea of proposed approach is to create a temporary distributed numeric data structure with values between 1 to N. Initializing the temporary distributed has a low computation cost and its contribution is two-fold: 1) This dataset serves as an artificial starting point for the MapReduce algorithm. 2) Each id serves as a seed number for initializing a temporal random number generator (RNG) for sampling the states of the current sequence. Relying on controlled and local RNGs assures persistent simulation results and removes computational bottlenecks in maintaining global read/write RNG resource. Once we generated this data structure, the map function is responsible to sample the states of the current sequence using a roulette wheel selection method [14].

IV. Evaluation

A. The T-Pot Use Case

In this section we present experimental results from applying the suggested methods on a massive data set of attacks on the T-Pot Multi-Honeypot Platform, an operational and globally deployed honeypot system. Each record in the analyzed dataset contains meta-data about an atomic attack on the system and includes the attacked honeypot, the source IP of the attacker and the timestamp. Overall, we analyzed a 114 GB dataset which includes approximately 170 million attack records between July 2014 and August 2015. The attacks targeted 253 different honeypots. Attacks were originated from almost 1 million distinct IP addresses from 229 countries.

The first experiment with the T-Pot datasets was training contextual propagation models. We applied algorithm 1 in order to create attack sequences using the attacker IP address and intervals of 1 hour as the object IDs. Next, we applied algorithm 2 and trained a series of contextual Markov chain models taking into consideration the country that originated the attacks. Overall we created multiple Markov chain models: a global one for all of the attacks, and country-wise specific models. The structure of the transition graph of global model is presented in figure 1. Each node/state on the graph represents a honeypot; while, each directed link between two pairs of nodes represents the relevant non-zero entry in the model's transition matrix. The nodes' color is a function of the number of its adjacent states. Deeper colors are indicators for more connected states.

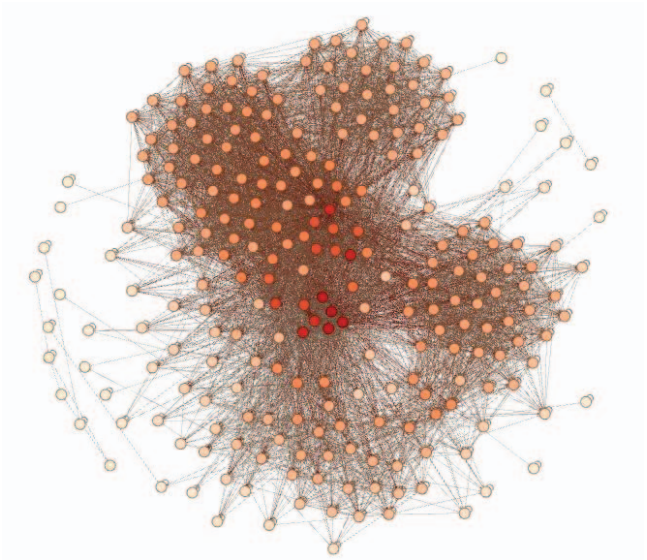

Figure 1. T-Pot Propagation Model (All Attacks)

Next, we analyzed whether the contextual modeling according to the attacking country revels different patterns. In particular, we focused on the top 3 attacking countries: China (CN), Russia (RU) and USA (US). These countries are responsible in total for 63.4% of the observed attacks in the dataset. To evaluate the effectiveness of the country based modeling we compered the results of the likelihood estimation measures from Algorithm 3.

We evaluated the likelihood of each of the four subsets of attacks: "All attacks", "CN attacks", "RU attacks" and "US attacks" on all of the four matching models "ALL", "CN", "RU "and "US". In table 1 we present normalized likelihood estimation results of all 16 <attacks, model> combinations. The normalized measure for each pair is the ratio between the likelihood of the attacks according to the "native" model and the likelihood according to the current evaluated model. The native "model" of an attack set is defined as the model that derived directly from those attacks. Overall we observed that contextual modeling is effective as it is preferable to model attacks from different countries separately, rather than relying on a global model. The most noticeable difference relates to attacks that originated from Russia. The likelihood ratio is less than 0.7, meaning relying solely on the global model when analyzing RU attacks will cause approximately 30% decrease the likelihood accuracy and may flag many common attacks sequences as new or abnormal while they are not.

TABLE I. CONTEXTUAL MODELING COMPARASION

Model / Attacks	ALL	CN	RU	US
ALL	1	0.878	0.541	0.851
CN	0.939	1	0.491	0.722
RU	0.693	0.314	1	0.284
US	0.889	0.705	0.553	1

Our last experiment with the T-Pot dataset concerned with creating simulated attack sequences. We applied the suggested approach on the four Markov chain models from the previous experiment i.e. the "ALL", "CN", "RU" and "US" models. For each model we produced 10 million simulated attack sequences. In order to evaluate the quality of the simulated sequences, we trained a new set of Markov chain models based on the simulated sequences and compered them to the original ones using likelihood estimation as preformed in the previous experiment. The pairwise comparison of similarity ranged between 0.998 and 0.99994. These high similarity values indicate that the generation process captures the dynamics of the sequences' transitions very well, as the Models trained on the simulated data are almost identical the ones from the original data.

B. Scalability and Synthitic Data Experiments.

In order to evaluate the scalability of the proposed algorithms, we implemented them on the Apache Spark framework with the Java API. All the experiments described in this section were performed in Spark's standalone mode with a quad-core 3.6 GHz CPU, 8G Ram and 500G hard drive. In addition to the T-Pot dataset, we generated synthetic ones from predefined Markov chain models. In our analysis we focused on two important characteristics of Markov chain models – the number of states and the density of the transition matrix. We evaluated Markov chains with 10, 100 and 1000 states along with density levels of 20%, 50% and 80%. For each of the 9 Markov chain configurations we generated a massive dataset containing 100 million sequences. On average each dataset contained 22 GB of data and more than a billion atomic records in the structure of <objecid, timestamp, stated> triplets as required for applying the suggested algorithms.

We applied the suggested MapReduce algorithms on the synthetic data records in a similar protocol to the T-Pot dataset and analyzed the three modeling tasks:

- Train Markov Chain Models – applying algorithms 1 and 2 on the available dataset.
- Likelihood Estimation – applying algorithms 1 and 3 on the trained models from the previous step and the available datasets.
- Random Walk Simulation - generating 100 million sequences using the trained models.

Table 2 summarizes the runtime results for both the T-pot dataset sets and the synthetic ones. The synthetic datasets are denoted as "synth" with the number of states and density level in parentheses. For each dataset and modeling task, we compered the run time in seconds when modifying the number of workers (ranged from 1 to 4) in the MapReduce environment. In the last row on table 2 we present the average scaling factor for each task and worker. The scaling factor is defined as the ratio between the runtime results of a single worker and the results with multiple workers. On average, cross all modeling tasks, we observed that adding more computational resources improves the run-time performances linearly, demonstrating the high scalability of the proposed approach.

TABLE II. SCALABLLITY ANALYSIS (RUN TIME RESLUTS IN SECONDS, 100 MLLION SEQUENCES)

Task/ # of Workers / Dataset	Train Markov Chain Models				Likelihood Estimation				Random Walk Simulation			
	1	2	3	4	1	2	3	4	1	2	3	4
T-Pot	13140	6821	4423	3404	10281	5080	3316	2396	284	153	108	87
Synth (10, 20%)	2634	1416	1022	762	2040	1046	771	545	58	33	25	18
Synth (10, 50%)	2683	1467	1084	799	2019	1134	788	562	64	36	26	20
Synth (10, 80%)	2691	1473	1100	818	2116	1056	874	610	68	38	28	20
Synth (100, 20%)	2952	1597	1234	871	2131	1188	952	627	160	84	59	42
Synth (100, 50%)	2986	1619	1215	834	2371	1228	900	597	278	146	102	76
Synth (100, 80%)	3054	1650	1211	923	2361	1286	867	674	270	144	99	74
Synth (1000, 20%)	3987	2201	1627	1102	2966	1604	1290	815	342	183	130	92
Synth (1000, 50%)	4980	2765	1971	1340	3856	2037	1549	961	665	346	262	203
Synth (1000, 80%)	5885	3211	2384	1600	4181	2562	1770	1206	972	446	376	281
Average Scaling Factor		**1.84**	**2.53**	**3.53**		**1.87**	**2.55**	**3.71**		**1.88**	**2.58**	**3.47**

V. CONCLUSIONS

In this work we presented a novel method for modeling contextual attack propagations patterns within honeypot systems. The suggested method extends previous work by adding new layers of analysis which include contextual modeling, likelihood estimation, and synthetic data generation. Empirical results on a massive dataset with approximately 170 million attacks on a honeypot system indicate that contextual features, such as the country from which the attack was initiated, have a substantial effect on the models and should be taken into consideration during the modeling phase. In addition, our findings show that the algorithm for generating synthetic attack sequences is capable of preserving the propagation patterns observed in the original data. Moreover, we demonstrate the high scalability of the proposed algorithms by experimenting with real and synthetic datasets consisting of over a billion records. We showed that the processing time decreases linearly when adding more computational resources in various configurations. These findings indicate that the proposed solution is applicable for analyzing massive datasets of operational honeypot systems.

Future research directions include the design of additional scalable methods for detecting contextual propagation patterns within honeypots, such as clustering methods or association rules. In addition, we want to explore methods which add additional layers of analysis to our analytical models, such as profiling and ranking the attackers according to the discovered propagation patterns.

REFERENCES

[1] Spitzner, L. " The honeynet project: Trapping the hackers". IEEE Security & Privacy, 1(2), 15-23, 2003.

[2] M. Kaaniche, Y. Deswarte, E. Alata, M. Dacier, and V. Nicomette, “Empirical analysis and statistical modeling of attack processes based on honeypots,” in Workshop on Empirical Evaluation of Dependability and Security (WEEDS), Philadelphia, USA, June 2006, pp. 119–124

[3] Grinstead, C. M., & Snell, J. L. (2012). Introduction to probability. American Mathematical Soc.

[4] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

[5] Chu, C., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G., Ng, A. Y., & Olukotun, K. (2007). Map-reduce for machine learning on multicore. Advances in neural information processing systems, 19, 281.

[6] Bringer, Matthew L., Christopher A. Chelmecki, and Hiroshi Fujinoki. "A survey: Recent advances and future trends in honeypot research." International Journal of Computer Network and Information Security 4.10 (2012): 63.

[7] Goseva-Popstojanova, Katerina, et al. "Characterization and classification of malicious Web traffic." Computers & Security 42 (2014): 92-115.

[8] Goseva-Popstojanova, Katerina, et al. "Quantification of attackers activities on servers running Web 2.0 applications." Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on. IEEE, 2010.

[9] Newsome, James, Brad Karp, and Dawn Song. "Polygraph: Automatically generating signatures for polymorphic worms." Security and Privacy, 2005 IEEE Symposium on. IEEE, 2005.

[10] Su, Ming-Yang. "Internet worms identification through serial episodes mining." Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON), 2010 International Conference on. IEEE, 2010.

[11] Kutchukian, P. S., Lou, D., & Shakhnovich, E. I. (2009). FOG: Fragment Optimized Growth algorithm for the de novo generation of molecules occupying druglike chemical space. Journal of chemical information and modeling, 49(7), 1630-1642.

[12] Tong, L., & Thompson, E. (2008). Multilocus lod scores in large pedigrees: combination of exact and approximate calculations. Human heredity, 65(3), 142-153.

[13] Sarukkai, R. R. (2000). Link prediction and path analysis using Markov chains. Computer Networks, 33(1), 377-386.

[14] Golberg, D. E. Genetic algorithms in search, optimization, and machine learning. Addion wesley, 1989.