An Automated Signature Generation Method for Zero-day Polymorphic Worms Based on C4.5 Algorithm

Mohssen M. Z. E. Mohammed[1], Eisa Aleisa[2], Neco Ventura[3]
[1,2]College of Computer and Information Sciences, Al-Imam Muhammad Ibn Saud Islamic University, Riyadh, Saudi Arabia
[2]Department of Electrical Engineering, University of Cape Town, Rondebosch, South Africa
m_zin44@hotmail.com, aleisa@ccis.imamu.edu.sa, neco@crg.ee.uct.ac.za

***Abstract*— Polymorphic worms are considered as the most critical threats to the Internet security, and the difficulty lies in changing their payloads in every infection attempt to avoid the security systems. In this paper, we propose an accurate signature generation system for zero-day polymorphic worms. We have designed a novel double-honeynet system, which is able to detect zero-day polymorphic worms that have not been seen before. To generate signatures for polymorphic worms, we have two steps. The first step is the polymorphic worms sample collection, which is done by the double-honeynet system. The second step is the signature generation for the collected samples, which is done by a decision tree algorithm (C4.5 algorithm). The main goal for this system is to get accurate signatures for Zero-day polymorphic worm.**

Keywords- Honeynet; Polymorphic; Worms; Machine Learning; Algorithm.

I. INTRODUCTION

Due to the enormous threat from the worms, many efforts have been taken previously to tackle worms by detecting and preventing them. Later in this paper, the relevant works are discussed. However, in this section, internet worm defense methods and their limitations are mentioned in brief.

One avenue to deal with worms is prevention. We usually know that prevention is better than cure. Since worms need to exploit software defects, by eliminating all software defects we could eradicate worms. While theoretically this seems to be easy, the reality finds this as an almost impossible goal. Although significant progress has been made on software development, testing, and verification, empirical evidence [1][12] suggests that we are still far from producing defect-free software.

Another avenue to solve the worm problem is containment. Containment systems accept that software has defects that can be exploited by worms, and they strive to contain a worm epidemic to a small fraction of the vulnerable machines. The main challenge in designing containment systems is that they need to be completely automatic, because worms can spread far faster than humans can respond [1]. Recent works on automatic containment [14][15] have explored network-level approaches. These rely on heuristics to analyze network traffic and derive a packet classifier that blocks or rate-limits forwarding of worm packets.

It is hard to provide guarantees on the rate of false positives and false negatives with these approaches because there is no information about the software vulnerabilities exploited by worms at the network level. False negatives allow worms to escape containment, while false positives may cause network outages by blocking normal traffic. We believe that an automatic containment systems will not be widely deployed unless they have a negligible false positive rate.

It should be noted here that dealing with the prevention mechanisms is out of the scope of this paper because our work mainly focuses on containment mechanism of the worms.

We use a supervised Machine Learning (ML) algorithm [16] to generate signatures for polymorphic worms. Supervised machine learning is the search for algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances. In other words, the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features. There are several applications for ML, the most significant of which is data mining. People are often prone to making mistakes during analyses or, possibly, when trying to establish relationships between multiple features. This makes it difficult for them to find solutions to certain problems. Machine learning can often be successfully applied to these problems, improving the efficiency of systems and the designs of machines. Every instance in any dataset used by machine learning algorithms is represented using the same set of features. The features may be continuous, categorical or binary. If instances are given with known labels (the corresponding correct outputs) then the learning is called supervised [16], in contrast to unsupervised learning [16], where instances are unlabeled. By applying these unsupervised (clustering) algorithms, researchers hope to discover unknown, but useful, classes of items. Another kind of machine learning is reinforcement learning [16]. The training information provided to the learning system by the environment (external trainer) is in the form of a scalar reinforcement signal that constitutes a measure of how well the system operates. The learner is not told which actions to take, but rather must discover which actions yield the best reward, by trying each action in turn [16].

This paper is organized as follows: After Section I, Section II gives an introduction to decision tree algorithms.

 ISBN: 978-1-61208-367-4

Section III discusses the related works regarding automated signature generation systems. Section IV talks about the preliminaries of worms and their attacks. Section V discusses our Double-Honeynet system. Section VI introduces the proposed C4.5 algorithm. Section VII concludes the paper.

II. OVERVIEW FOR DECISION TREES

Decision trees classify instances by sorting them down the tree from the root to some leaf node, where:

- Each internal node specifies a test of some attribute.
- Each branch corresponds to a value for the tested attribute.
- Each leaf node provides a classification for the instance.

Figure 1 is an example of a decision tree for the training set of Table I.

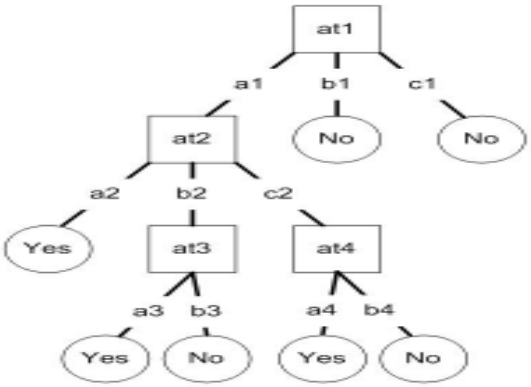

Figure 1. A decision tree.

TABLE I. TRAINING SET

at1	at2	at3	at4	Class
a1	a2	a3	a4	Yes
a1	a2	a3	b4	Yes
a1	b2	a3	a4	Yes
a1	b2	b3	b4	No
a1	c2	a3	a4	Yes
a1	c2	a3	b4	No
b1	b2	b3	b4	No
c1	b2	b3	b4	No

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance [3].

Here, we give some explanation of Figure 1. The instance ⟨at1 = a1, at2 = b2, at3 = a3, at4 = b4⟩ would sort to the nodes: at1, at2, and finally at3, which would classify the instance as being positive (represented by the values "Yes"). The problem of constructing optimal binary decision trees is an NP-complete problem and thus theoreticians have searched for efficient heuristics for constructing near-optimal decision trees.

The feature that best divides the training data would be the root node of the tree. There are numerous methods for finding the feature that best divides the training data, such as information gain and gini index. While myopic measures estimate each attribute independently, ReliefF algorithm estimates them in the context of other attributes. However, a majority of studies have concluded that there is no single best method. Comparison of individual methods may still be important when deciding which metric should be used in a particular dataset. The same procedure is then repeated on each partition of the divided data, creating sub-trees until the training data is divided into subsets of the same class.

Below, we present a general pseudo-code for building decision trees.

```
Check for base cases
        For each attribute a
         Find the feature that best
         divides the training data such as
         information gain from splitting on a
 Let a_best be the attribute with the
 highest normalized information gain
         Create a decision node node that
         Splits on a_best
 Recurse on the sub-lists obtained by
 splitting on a_best and add those
 nodes as children of node
```

A decision tree, or any learned hypothesis h is said to overfit training data if another hypothesis h′ exists that has a larger error than h when tested on the training data, but a smaller error than h, when tested on the entire dataset. There are two common approaches that decision tree induction algorithms can use to avoid over-fitting training data which are [3]:

- Stop the training algorithm before it reaches a point at which it perfectly fits the training data,
- Prune the induced decision tree. If the two trees employ the same kind of tests and have the same prediction accuracy, the one with fewer leaves is usually preferred.

The most straightforward way of tackling over-fitting is to pre-prune the decision tree by not allowing it to grow to its full size. Establishing a non-trivial termination criterion such as a threshold test for the feature quality metric can do

 ISBN: 978-1-61208-367-4

that. Decision tree classifiers usually employ post-pruning techniques that evaluate the performance of decision trees, as they are pruned by using a validation set. Any node can be removed and assigned the most common class of the training instances that are sorted to it. Elomaa [16] concluded that there is no single best pruning method.

Even though the divide-and-conquer algorithm is quick, efficiency can become important in tasks with hundreds of thousands of instances. The most time consuming aspect is sorting the instances on a numeric feature to find the best threshold t. This can be expedited if possible thresholds for a numeric feature are determined just once, effectively converting the feature to discrete intervals, or if the threshold is determined from a subset of the instances. Elomaa and Rousu [16] stated that the use of binary discretization with C4.5 [3] needs about the half training time of using C4.5 multi splitting. In addition, according to their experiments, multi-splitting of numerical features does not carry any advantage in prediction accuracy over binary splitting.

Decision trees use splits based on a single feature at each internal node, so that they are usually univariate. In fact, most decision tree algorithms cannot perform well with problems that require diagonal partitioning. The division of the instance space is orthogonal to the axis of one variable and parallel to all other axes. Therefore, the resulting regions after partitioning are all hyperrectangles. However, there are a few methods that construct multivariate trees. S. B. Kotsiantis [16] presented Zheng's work, who improved the classification accuracy of the decision trees by constructing new binary features with logical operators such as conjunction, negation, and disjunction. In addition, Zheng created at-least M-of-N features. For a given instance, the value of an at o trees. In this model, new features are computed as linear combinations of the previous ones.

In the fact, decision trees can be significantly more complex representation for some concepts due to the replication problem. A solution to this problem is using an algorithm to implement complex features at nodes in order to avoid replication. In [16], S. B. Kotsiantis discussed Markovitch and Rosenstein work, and they presented the FICUS construction algorithm, which receives the standard input of supervised learning as well as a feature representation specification, and uses them to produce a set of generated features. While FICUS is similar in some aspects to other feature construction algorithms, its main strength is its generality and flexibility. FICUS was designed to perform feature generation given any feature representation specification complying with its general purpose grammar.

III. RELATED WORKS

Honeypots are an excellent source of data for intrusion and attack analysis. Levin et al. [4] described how Honeynet can be used to assist the system administrator in identifying malicious traffic on an enterprise network and how Honeypot-extracts with details of worm can be analyzed to generate detection signatures. The signatures are generated manually.

One of the first systems proposed was Honeycomb developed by Kreibich and Crowcroft [5]. Honeycomb generates signatures from traffic observed at a Honeypot via its implementation as a Honeyd plugin. The Longest Common Substring (LCS) algorithm, which looks for the longest shared byte sequences across pairs of connections, is at the heart of Honeycomb. Honeycomb generates signatures consisting of a single, contiguous substring of a worm's payload to match all worm instances. These signatures, however, fail to match all polymorphic worm instances with low false positives and low false negatives.

Kim and Karp [6] described the Autograph system for automated generation of signatures to detect worms. Unlike Honeycomb, Autograph's inputs are packet traces from a DMZ (demilitarized zone) that includes benign traffic. Content blocks that match "enough" suspicious flows are used as input to COPP [6], an algorithm based on Rabin fingerprints that searches for repeated byte sequences by partitioning the payload into content blocks. Similar to Honeycomb, Auto-graph generates signatures consisting of a single, contiguous substring of a worm's payload to match all worm instances. These signatures, unfortunately, fail to match all polymorphic worm instances with low false positives and low false negatives.

Singh et al. [7] described the Earlybird system for generating signatures to detect worms. This system measures packet-content prevalence at a single monitoring point, such as a network DMZ. By counting the number of distinct sources and destinations associated with strings that repeat often in the payload, Earlybird distinguishes benign repetitions from epidemic content. Earlybird, also like Honeycomb and Autograph, generates signatures consisting of a single, contiguous substring of a worm's payload to match all worm instances. These signatures, however, fail to match all polymorphic worm instances with low false positives and low false negatives.

New content-based systems, like Polygraph [8], Hamsa [10] and LISABETH [11], have been deployed. All these systems, similar to our system, generate automated signatures for polymorphic worms based on the following fact: there are multiple invariant substrings that must often be present in all variants of polymorphic worm payloads even if the payload changes in every infection. All these systems capture the packet payloads from a router, so in the worst case, these systems may find multiple polymorphic worms but each of them exploits a different vulnerability from each other. So, in this case, it may be difficult for the above systems to find invariant contents shared between these polymorphic worms because they exploit different vulnerabilities. The attacker sends one instance of a polymorphic worm to a network, and this worm in every infection automatically attempts to change its payload to generate other instances. So, if we need to capture all polymorphic worm instances, we need to give a polymorphic worm, chance to interact with hosts without affecting their performance. So, we propose a new detection method "*Double-honeynet*" to interact with polymorphic

 ISBN: 978-1-61208-367-4

worms and collect all their instances. The proposed method makes it possible to capture all worm instances and then forward these instances to the Signature Generator which generates signatures, using a particular algorithm.

An Automated Signature-Based Approach against Polymorphic Internet Worms by Tang and Chen [9] described a system to detect new worms and generate signatures automatically. This system implemented a Double-honeypot (inbound Honeypot and outbound Honeypot) to capture worms payloads. The inbound Honeypot is implemented as a high-interaction Honeypot, whereas the outbound Honeypot is implemented as a low-interaction Honeypot. This system has limitations. The outbound Honeypot is not able to make outbound connections because it is implemented as low-interaction honeypot which is not able to capture all polymorphic worm instances. Our system overcomes this disadvantage by using Double-honeynet (high-interaction Honeypot), which enables us to make unlimited outbound connections between them, so that we can capture all polymorphic worm instances.

All of the above works have used different algorithms to generate signatures for polymorphic worms, but there is no one in the above works using data mining algorithms to detect polymorphic worms. Data mining is a new technology and has successfully applied on a lot of fields; the overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use. Data mining is mainly used for model classification and prediction. classification is a form of data analysis that extracts models describing important data classes. C4.5 [16] is one of the most classic classification algorithms on data mining. So, in this paper, we used C4.5 algorithm for polymorphic worm classification. The C4.5 algorithm can classified each type of polymorphic worm into group.

The objective of using C4.5 is to generate signatures for polymorphic worms.

The advantages of using C4.5 algorithm over the others algorithms is the C4.5 can generate an accurate signatures for polymorphic worms.

IV. PRELIMINARIES OF WORM AND WORM ATTACKS

In this section, we talk about worms, so that the readers can learn how worm can attack victim computers connected to the Internet.

Worms are basically computer programs that self-replicate without requiring any human intervention; especially, by sending copies of their code in network packets and ensuring the code is executed by the computers that receive it. When computers become infected, they spread further copies of the worm and possibly perform other malicious activities.

A. Worm Infection

Remotely infecting a computer requires coercing the computer into running the worm code. To achieve this, worms exploit low-level software defects, also known as vulnerabilities. Vulnerabilities are common in current software, because today's software is usually large, complex, and mostly written in unsafe programming languages. Several different classes of vulnerabilities have been discovered over the years. Currently, buffer overflows, arithmetic overflows, memory management errors, and incorrect handling of format strings, are among the most common types of vulnerabilities exploitable by worms.

While we should expect new types of vulnerabilities to be discovered in the future, the mechanisms used by worms to gain control of a program's execution should change less frequently. Currently, worms gain control of the execution of a remote program using one of three mechanisms: injecting new code into the program, injecting new control-flow edges into the program (e.g., forcing the program to call functions that should not be called), and corrupting data used by the program.

B. Spread of Internet Worms

After infecting a computer, worms typically use it to infect other computers, giving rise to a propagation process which has many similarities with the spread of human diseases.

The spread of the worm in its most basic sense depends mostly on how it chooses its victims. This not only affects the spread and pace of the worm network but also its survivability and persistence as cleanup efforts begin. Classically, worms have used random walks of the Internet to find hosts and attack. However, new attack models have emerged that demonstrate increased aggressiveness.

C. Components of Worm

There are five basic components of worm:

Reconnaissance. The worm network has to hunt out other network nodes to infect. This component of the worm is responsible for discovering hosts on the network that are capable of being compromised by the worm's known methods.

Attack Components. These are used to launch an attack against an identified target system. Attacks can include the traditional buffer or heap overflow, string formatting attacks, Unicode misinterpretations (in the case of IIS (Internet Information Server) attacks), and misconfigurations.

Communication Components. Nodes in the worm network can talk with each other. The communication components give the worms the interface to send messages between nodes or some other central location.

Command Components. Once compromised, the nodes in the worm network can be issued operation commands using this component. The command element provides the interface to the worm node to issue and act on commands.

Intelligence Components. To communicate effectively, the worm network needs to know the location of the nodes as well as characteristics about them. The intelligence portion of the worm network provides the information needed to be able to contact with other worm nodes, which can be accomplished in a variety of ways [21].

 ISBN: 978-1-61208-367-4

V. OUR HONEYNET SYSTEM

We propose a Double-honeynet system to detect new worms automatically. A key contribution of this system is the ability to distinguish worm activities from normal activities without the involvement of experts.

Figure 2 shows the main components of the Double-honeynet system. Firstly, the incoming traffic goes through the Gate Translator which samples the unwanted inbound connections and redirects the sample connections to Honeynet 1. The gate translator is configured with publicly-accessible addresses, which represent wanted services. Connections made to other addresses are considered unwanted and redirected to Honeynet 1 by the Gate Translator.

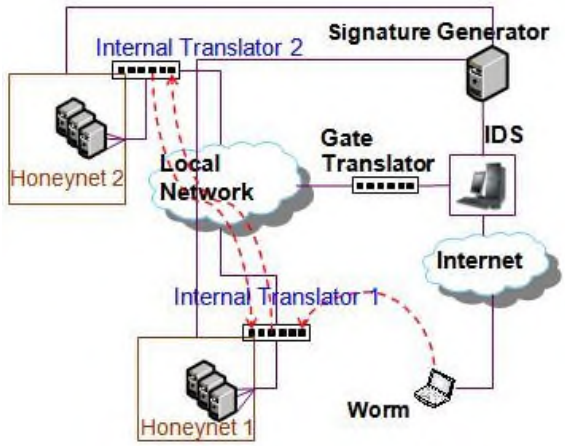

Figure 2. Double-honeynet system.

Secondly, once Honeynet 1 is compromised, the worm will attempt to make outbound connections. Each Honeynet is associated with an Internal Translator implemented in router that separates the Honeynet from the rest of the network. The Internal Translator 1 intercepts all outbound connections from Honeynet 1 and redirects them to Honeynet 2, which does the same, forming a loop.

Only packets that make outbound connections are considered malicious, and hence the Double-honeynet forwards only packets that make outbound connections. This policy is due to the fact that benign users do not try to make outbound connections if they are faced with non-existing addresses.

Lastly, when enough instances of worm payloads are collected by Honeynet 1 and Honeynet 2, they are forwarded to the Signature Generator component which generates signatures automatically using specific algorithms that will be discussed in the next section. Afterwards, the Signature Generator component updates the IDS database automatically by using a module that converts the signatures into Bro or pseudo-Snort format.

The above mentioned system was implemented by using Vmware Server 2 [13]. The details of the core implementation matters are out of the scope of this paper and were reported earlier; the readers are encouraged to read on the Double-honeynet architecture in our previously published work [13][17][18].

VI. C4.5 ALGORITHM

Motivation for Using C4.5 for Polymorphic Worms Detection

As it is known that a polymorphic worm can change its payload in every infection attempt, it is so difficult to know all instances of a polymorphic worm. In this paper, we use a well-known algorithm in classification problems, which is the C4.5. The advantage of using the C4.5 is polymorphic worm classifications.

We propose C4.5 algorithm to detects Zero-day polymorphic worms. The most well-know algorithm for building decision trees is the C4.5 [3]. C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan. This is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier.

C4.5 builds decision trees from a set of training data in the same way that ID3 does, using the concept of information entropy. The training data are a set $S = s_1, s_2, ...$ of already classified samples. Each sample, $s_i = x_1, x_2, ...$ is a vector where $x_1, x_2, ...$ represent attributes or features of the sample. The training data is augmented with a vector $C = c_1, c_2, ...$ where $c_1, c_2, ...$ represent the class to which each sample belongs.

At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. Its criterion is the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurses on the smaller subsists [3].

This algorithm has a few base cases:

- All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
- None of the features provides any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
- Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

We should mention that machine learning algorithm are very slow in working, so in the future work we would like to use some of mathematical methods to enhances the our machine learning algorithm efficiency.

VII. CONCLUSION

In this paper, we have proposed an automated signature generation mechanism for zero-day polymorphic worms using a decision tree algorithm (C4.5 algorithm). In fact,

 ISBN: 978-1-61208-367-4

there are many other algorithms that have been proposed to generate signatures for zero-day polymorphic worms, but most of them have limitation to detect unknown pattern and also they have high computational complexity. Therefore, we have used C4.5 algorithm which overcomes these problems (detecting unknown pattern and computational complexity). One of the main advantages of machine learning algorithms is their great capacity to extract unknown and general information from a given data set (polymorphic worms samples) and its application on new data. The main goal of this paper was to use a machine learning technique (C4.5 algorithm) which can get better results than other algorithms such as string matching algorithms or similar others.

REFERENCES

[1] L. Spitzner, "Honeypots: Tracking Hackers," Addison Wesley Pearson Education: Boston, 2002.

[2] H. Bidgoli, "Handbook of Information Security," John Wiley & Sons, Inc., Hoboken, New Jersey.

[3] D. Gusfield, "Algorithms on Strings, Trees and Sequences,", Cambridge University Press: Cambridge, 1997.

[4] J. Levine, R. La Bella, H. Owen, D. Contis,and B. Culver, "The use of honeynets to detect exploited systems across large enterprise networks," Proc. of 2003 IEEE Workshops on Information Assurance, New York, Jun. 2003, pp. 92- 99.

[5] C. Kreibich and J. Crowcroft, "Honeycomb–creating intrusion detection signatures using honeypots," Workshop on Hot Topics in Networks (Hotnets-II), Cambridge, Massachusetts, Nov. 2003.

[6] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," Proc. of 13 USENIX Security Symposium, San Diego, CA, Aug., 2004.

[7] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," Proc. Of the 6th conference on Symposium on Operating Systems Design and Implementation (OSDI), Dec. 2004.

[8] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," Proc. of the 2005 IEEE Symposium on Security and Privacy, pp. 226 – 241, May 2005.

[9] Y. Tang, S. Chen, "An Automated Signature-Based Approach against Polymorphic Internet Worms," IEEE Transaction on Parallel and Distributed Systems, pp. 879-892, July 2007.

[10] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao and B. Chavez, "Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience," Proc. of the IEEE Symposium on Security and Privacy, Oakland, CA, May 2006.

[11] L. Cavallaro, A. Lanzi, L. Mayer, and M. Monga, "LISABETH: Automated Content-Based Signature Generator for Zero-day Polymorphic Worms," Proc. of the fourth international workshop on Software engineering for secure systems, Leipzig, Germany, May 2008.

[12] J. Nazario, "Defense and Detection Strategies against Internet Worms " Artech House Publishers (October 2003).

[13] M.M.Z.E. Mohammed, H.A. Chan, N. Ventura. "Honeycyber: Automated signature generation for zero-day polymorphic worms"; Proc. of the IEEE Military Communications Conference, MILCOM, 2008.

[14] Snort – The de facto Standard for Intrusion Detection/Prevention. Available: http://www.snort.org, 1 April 2012.

[15] Bro Intrusion Detection System. Available: http://www.bro-ids.org/, last accessed: 4 November 2013.

[16] S.B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," Informatica 31 (2007) 249-268.

[17] M.M.Z.E. Mohammed and A.-S. K. Pathan. Automatic Defense against Zero-day Polymorphic Worms in Communication Networks, ISBN 9781466557277, CRC Press, Taylor & Francis Group, USA, 2013.

[18] M.M.Z.E. Mohammed and A.-S. K. Pathan, "Using Routers and Honeypots in Combination for Collecting Internet Worm Attacks," The State of the Art in Intrusion Prevention and Detection (Edited by Al-Sakib Khan Pathan), ISBN: 9781482203516, CRC Press, Taylor & Francis Group, USA, 2014 (To Appear).

 ISBN: 978-1-61208-367-4