2014

Application of Cellular Automata to Detection of Malicious Network Packets

Robert L. Brown
Nova Southeastern University, robebrow@nova.edu

Application of Cellular Automata to Detection of Malicious Network Packets

by

Robert L. Brown

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Information Systems

Graduate School of Computer and Information Sciences
Nova Southeastern University

2014

We hereby certify that this dissertation, submitted by Robert L. Brown, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

_____ _____

James D. Cannady, Ph.D. Date
Chairperson of Dissertation Committee

_____ _____

Bob Harbort, Ph.D. Date
Dissertation Committee Member

_____ _____

Wei Li, Ph.D. Date
Dissertation Committee Member

Approved:

_____ _____

Eric S. Ackerman, Ph.D. Date
Dean, Graduate School of Computer and Information Sciences

Graduate School of Computer and Information Sciences
Nova Southeastern University

2014

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Application of Cellular Automata to Detection of Malicious Network Packets

by
Robert L. Brown
January, 2014

A problem in computer security is identification of attack signatures in network packets. An attack signature is a pattern of bits that characterizes a particular attack. Because there are many kinds of attacks, there are potentially many attack signatures. Furthermore, attackers may seek to avoid detection by altering the attack mechanism so that the bit pattern presented differs from the known signature. Thus, recognizing attack signatures is a problem in approximate string matching. The time to perform an approximate string match depends upon the length of the string and the number of patterns. For constant string length, the time to match n patterns is approximately $O(n)$; the time increases approximately linearly as the number of patterns increases.

A binary cellular automaton is a discrete, deterministic system of cells in which each cell can have one of two values. Cellular automata have the property that the next state of each cell can be evaluated independently of the others. If there is a processing element for each cell, the next states of all cells in a cellular automaton can be computed simultaneously.

Because there is no programming paradigm for cellular automata, cellular automata to perform specific functions are created *ad hoc* by hand or discovered using search methods such as genetic algorithms.

This research has identified, through evolution by genetic algorithm, cellular automata that can perform approximate string matching for more than one pattern while operating in constant time with respect to the number of patterns, and in the presence of noise. Patterns were recognized by using the bits of a network packet payload as the initial state of a cellular automaton. After a predetermined number of cycles, the ones density of the cellular automaton was computed. Packets for which the ones density was below an experimentally determined threshold were identified as target packets. Six different cellular automaton rules were tested against a corpus of 7.2 million TCP packets in the IDEval data set. No rule produced false negative results, and false positive results were acceptably low.

Acknowledgements

My thanks go to my committee members, Dr. Bob Harbort and Dr. Wei Li, who provided support and guidance throughout the process, and most especially to Dr. James Cannady, who never gave up on me, even when I had given up on myself.

Thanks go to my colleagues at Southern Polytechnic State University who were patient with me while I divided my time between my teaching duties and my studies, and to the hundreds of my students who were similarly patient.

I am grateful to Cindy Neck, who volunteered for proofreading *again.* Thank you again, Cindy. Thank you, Betty Abbott, for the second round of proofreading.

Thank you, Campers, for making me a member of the family, for support over many years, and for keeping me fed! Thank you for understanding all those times when I said I had to work.

Finally, to Gina, who may not remember saying, "I will when you do." You've got a lot of catching up to do, Child!

Table of Contents

List of Tables

Tables

List of Figures

Figures

Chapter 1

Introduction

Background

A problem in computer security is the detection of malicious data in transit through networks or on entry to a host computer. One approach to this problem is signature-based detection, in which data are compared against known patterns (Stallings & Brown, 2008). This is very similar to pattern detection in large data sets. In both cases, efficient algorithms for comparing data to multiple known patterns are required.

Cellular automata have been used for language recognition (Sommerhalder & Westrhenen, 1983), pattern recognition (Ganguly et al., 2004), and as associative memories (Chowdhury et al., 2002). This research demonstrates that a cellular automaton can be an effective tool for both signature-based detection of malicious data and for pattern searching in large data sets. A cellular automaton is a Moore model[1] finite state machine in which the next state of a node (cell) depends only upon the current state of the node and the current states of neighboring nodes (Sarkar, 2000). Thus, the next state of each cell can be computed independently of the next states of any others; with sufficiently many computing elements, the next state of every cell can be computed in parallel with all the others (Sommerhalder & Westrhenen, 1983). Because of the

---

[1] In a Moore model finite state machine, the outputs are driven by the state register, and so can change only when the state changes. This is in contrast to the Mealy model finite state machine, in which the outputs are driven by combinational logic and can change as a result of changing inputs, independent of a state change (Mano & Kime, 2007).

inherent parallelism in cellular automata, this approach is potentially faster than other methods of pattern recognition provided that the implementation is able to take advantage of the parallelism.

While cellular automata have the advantages just described, there is a major disadvantage: there is no programming paradigm that will produce a cellular automaton for a specified task (Crutchfield et al., 1998). Instead, cellular automata for specific applications must be discovered. Because the search space gets very large for even simple cellular automata (Mitchell et al., 1996) a heuristic search rather than an exhaustive search is needed. One mechanism for such a heuristic search is the genetic algorithm (Reeves & Rowe, 2003).

This report describes cellular automata that can recognize specific patterns relevant to computer security in constant time and in the presence of noise. Such cellular automata are potentially applicable to searching of large databases. Heuristic search with a genetic algorithm was used to discover the cellular automata described here.

Problem Statement

There is currently no way of identifying single packet network attack signatures in less than approximately linear time. A network attack is the transmission of data to a system with the intent of violating the system's security policy (Bishop, 2003). A single packet attack signature is a characteristic pattern that can be identified without reference to the other packets that may accompany it. The time required by current methods to check for characteristic patterns increases approximately linearly as the number of signatures to be tested increases. However, if two or more patterns could be checked simultaneously, the process of testing for those patterns would operate in constant time,

O(1). Cellular automata have the property that a given number of cycles requires constant time (Sommerhalder & Westrhenen, 1983). This report identifies a specific pair of patterns that can be detected in constant time through the application of a cellular automaton. However, it may not be the case that any arbitrary set of patterns can be detected in constant time.

For known attacks against computer or network resources, malicious packets can often be detected using signature analysis, matching incoming packets against patterns known to represent attacks (Bishop, 2003). However, signature matching to detect malicious packets is complicated by the fact that the contents of the packets may change over time. Sometimes the changes are result of modifications by the attacker. Often, malicious software is designed to be self-modifying, substituting equivalent instructions, so the function of the malicious packet remains the same but the actual bits transmitted change in an attempt to defeat signature-matching defenses (Vinod et al., 2009). These perturbations appear as noise with respect to the signature (pattern) and the data being checked (Erdogan & Cao, 2007).

Thus, the problem of detecting known malicious network packets through signature analysis is a problem of approximate string matching. Navarro (2001) provides an extensive survey of approximate string matching and shows that the best algorithms operate in near linear time, $O(n)$, where n is the size of the string to be searched. Checking each signature pattern requires time $O(n)$. As the number of signatures to be checked increases, the time to examine each packet increases. So, for m signatures, the time to check a packet is $O(nm)$, linear time if n is constant. When the number of patterns becomes large, the time to check each packet against every pattern also becomes large.

This report describes the use of cellular automata for malicious packet detection, including the degree of parallelism possible and the sensitivity and specificity of detection.

Dissertation Goal

This report describes a proof of concept mechanism for comparing payload portion of network packets to two patterns known to represent attacks through the use of a cellular automaton. The mechanism detects such patterns even in the presence of small changes in the actual packet and has constant detection time with respect to the number of patterns checked. Specifically, adding a second pattern does not increase detection time.

Implementation was on a standard personal computer without parallel computation capability. However, the mechanism itself is capable of highly-parallel operation if implemented in a parallel computing environment. Testing used the MIT/DARPA Intrusion Detection System Evaluation data set, a publicly available data set, as described in (Haines et al., 2001) and others.

Research Questions

Three principal questions were addressed by this research. First, whether it is possible to identify cellular automata that can recognize the patterns of more than one known network attack. Second, whether recognizing such patterns is possible in the presence of noise or perturbation. Third, whether genetic algorithms are an appropriate vehicle for identifying such cellular automata. Each of these questions has been addressed separately by others. They are considered together here, and in the context of intrusion detection and the recognition of patterns in the presence of noise.

Cellular automata have been applied to the task of pattern recognition since Smith's work in 1971 (Smith, 1971) and to language recognizers, including the work of Sommerhalder and Westrhenen (1983). The work of Wolfram (2002) has shown that there exist cellular automata that are sensitive to initial conditions. Mitchell et al. (1994) and Crutchfield et al. (1998) have successfully evolved cellular automata using genetic algorithms and have shown that there exist cellular automata that reach known configurations under certain conditions of initial input but not others. Wolfram (1994), in describing three classes of cellular automata, observed that Class 2 cellular automata function as filters. Wolfram (1994) suggested their application to enhancing specific patterns in digital image processing. Ganguly et al. (2004) have used cellular automata in pattern recognition.

A major advantage of using a cellular automaton for pattern recognition is the ability to detect patterns in the presence of noise or perturbation (Maji et al., 2003). Another advantage is the ability to operate in constant time (Ganguly et al., 2004; Sommerhalder & Westrhenen, 1983) . A third advantage is that the nature of the cellular automaton can be changed by changing the rule that is evaluated; changes to the underlying implementation of the cellular automaton are not needed (Wolfram, 1994).

It follows from the characteristics of cellular automata as described by Wolfram (1994) that, if there is a processing unit for each cell, the next state for every cell can be computed in parallel. Thus, in the case of fully parallel computation, the time to detect a pattern depends only on the number of cycles required to reach a recognizable state, and not on the number of cells in the cellular automaton. For the patterns studied, the time to recognize a pattern is also independent of the number of patterns recognized (Ganguly et

al., 2004). Such a cellular automaton operates in constant time, *O(1)* because the number of cycles necessary to identify a pattern is independent of the size of the string being checked and of the number of patterns being compared. If fully associative comparison is used to detect the state of the cellular automaton that signals recognition, then the entire mechanism operates in constant time. However, operation in constant time is possible only for the special case that there exists a cellular automaton that can detect a specific set of patterns.

This report describes the use of a genetic algorithm to evolve a cellular automaton that can detect two specific patterns, each of which characterizes an attack against a network, even in the presence of noise or purposeful perturbation, and which operates in constant time with respect to the number of patterns.

Relevance and Significance

The results reported here directly address the problem of identifying malicious network packets by their signatures in the presence of perturbations and in constant time for a given packet size regardless of whether one or two patterns were checked. The resulting mechanism could be applied to network packets at the point that they enter a protected computing system, namely at the network interface. It could also provide an efficient, highly parallel mechanism for searching large data sets for known patterns. In either case, the inherent parallelism of cellular automata allows all bits of the packet to be examined simultaneously provided sufficient parallelism is available in the hardware running the cellular automaton. Fully parallel implementation would require specialized hardware, such as multiple graphics processor cards or an application-specific integrated

circuit (ASIC). A detector for a 1,500 byte Ethernet packet requires 12,000 processing elements.

This report describes a mechanism capable of identifying malicious network packets at hardware speeds provided suitable parallel computation resources are provided. Although it is in no respect a complete solution to the problem of protecting networks and computing systems, it fits well with other current research in the area. Specifically, it represents a contribution to distributed intrusion detection as described by Rhodes et al. (2000) and Forrest et al. (1997), among others. Further, it contributes to the area of NIC-based intrusion detection as described by Singaraju et al. (2005), Clark et al. (2004) and Otey et al. (2003).

The result described is a proof of concept implementation in software of the cellular automaton. Such a cellular automaton is capable of improved protection of computer systems from identifiable attack signatures, even in the presence of noise. An important result of this research is a detection algorithm that can be implemented in a highly parallel fashion.

Barriers and Issues

The test data set used was the MIT/DARPA Intrusion Detection System Evaluation (IDEval) test data set. There has been considerable criticism of the qualities of this data set. Most of the criticism is of anomalies introduced through production of synthetic data. The experimental team at MIT synthesized the test data rather than using data captured from an operating network for considerations of privacy and the probability that sensitive data would be captured and subsequently exposed to the intrusion detection systems under evaluation (Mahoney & Chan, 2003).

Mahoney and Chan (2003) identified eight simulation artifacts in the TCP or IP headers and only three in the payload portion. Because the pattern recognition described here focuses on the payload data, the header artifacts do not constitute a barrier to the use of the data set.

The three payload-related simulation artifacts were all related to higher-level protocol information carried as payload in the TCP/IP packets, and not the actual data part of the payload. They included highly regular HTTP request headers, similar regularity in SMTP requests, and the fact that the same version number was used in all SSH requests (Mahoney & Chan, 2003) . Thus, the payload related artifacts did not constitute a barrier to the use of the data set in the work reported here.

The critique by McHugh (2000) was more qualitative than quantitative but neither his critique nor that of Mahoney and Chan (2003) identified simulation artifacts in the data portion of malicious packets.

The data set used for validation is the largest publicly-available one which has the payload portion intact. The other large data sets have had the payload portion stripped to protect privacy (Mahoney & Chan, 2003). Because the research reported here focuses entirely on the payload data, and as no anomalies have been identified in the payload data, the MIT/DARPA IDEval data set is a suitable test vehicle for the reported research. In spite of criticism, the MIT/DARPA IDEval data set continues to be used by others, including Tavallaee et al. (2010) and Löf and Nelson (2010). The use of a publicly-available data set will allow interested researchers to replicate and verify the results that are reported here.

Limitations and Delimitations

Because this is a proof of concept demonstration, search for cellular automata that can detect malicious patterns ceased when a rule that detects two such patterns was discovered.

The demonstration system was implemented on a single processor personal computer running standard software. For that reason, although complete parallelism is possible in computing the next state of cellular automata when there is one computing element per cell, that parallelism is not present in the demonstration. However, the demonstration shows the ability to detect two distinct patterns in a single execution of the cellular automaton and with a fixed number of cycles of the cellular automaton. That is the principal level of parallelism in the system. Parallel execution of next states is the second level; the next states of all cells can be computed simultaneously. Evaluating the state of the cellular automaton involves examining each individual bit. Accomplishing that in constant time requires a fully associative comparison. The time for determination of state does not increase when more than one pattern is recognized.

It is important to note in the context of the reported research that there exist attacks which cannot be recognized at all through inspection of network packets. One example is the ping flood, in which the attacked node is overwhelmed by a large volume of ICMP echo requests. The purpose is to saturate either the incoming network connection or the processing power of the node under attack. Another is the SYN flood, in which the attacker creates a large number of half-open TCP connections. The attacker's purpose is to exhaust the node resources used to account for connections in the process of being established (Stallings & Brown, 2008). In the first case, any one of the

flood of pings is indistinguishable from an innocent diagnostic ping intended to determine whether the node is operational and reachable. As with pings, a single SYN packet cannot, by itself, be determined to be part of an attack. In neither case does inspection of the network packets in isolation reveal the malicious intent of the attacker. These attacks must be detected or prevented using some mechanism other than the one described here.

This research was not intended to result in an entire intrusion detection system. Instead, it provides the proof of concept of one component of a distributed intrusion detection system that can respond quickly to specific events, and an approximate pattern matching system that can be implemented in a highly parallel fashion.

Summary

The ubiquitous network connectivity that arose at the beginning of the twenty-first century means that information systems are potentially exposed to attack from anywhere in the world. One method of detecting attacks is by their signatures, specific bit patterns that characterize known attacks. An important consideration in signature checking is the speed with which it can be accomplished. In order not to cause a bottleneck, such checking must operate at the speed of the network connection. This research has identified a mechanism that can check for two distinct signatures simultaneously and, with a suitable number of processing elements, can operate in a highly parallel fashion.

Chapter 2

Review of the Literature

Introduction

Information security is characterized by three properties: confidentiality, integrity, and availability. Each property describes a state or condition of an information asset (Bishop, 2003). Confidentiality is achieved when information assets are not disclosed other than to those who are, by policy, authorized to have access. Integrity addresses the trustworthiness of information, and exists when information in an automated system agrees with the source from which it was derived and has not been incorrectly altered or destroyed. Availability means that information assets are accessible to authorized persons when and where needed, with suitable response time (Stallings & Brown, 2008).

The confidentiality, integrity, or availability of information assets can be compromised through accident or by malicious intent. One refers to an "attacker" in cases of intentional attempts to compromise the confidentiality, integrity, or availability of information assets. According to Stallings and Brown (2008), the goals of the attacker are duals of the three properties of information security, namely disclosure, alteration, and denial of availability.

McCumber (1991) identifies three states of information in addition to the three properties. They are processing, transmission, and storage. The work described here addresses information during processing, transmission, and a subset of storage. Only

information in online storage is addressed; that is, information that is accessible through the execution of instructions on a computer processor and without manual intervention.

According to Pfleeger and Pfleeger (2006), a computer system is secure when it does what it is intended to do and nothing else. Technical security breaches are the result of a system being forced to operate outside its design parameters. Thus, technical security breaches are the result of errors in specification, design, or implementation. However, it is extraordinarily difficult to build non-trivial systems of hardware and software that are free of error (Stallings & Brown, 2008). It is necessary to compensate for potential errors in specification, design, or implementation through other means.

Some attacks against information assets involve intrusions into computer systems. An intrusion sometimes involves the introduction of executable instructions into a computer system with the intent of causing disclosure, alteration, or denial of availability (Solomon & Chapple, 2005) by executing actions contrary to the security policy of the system. Such executable instructions are called malicious software or malware (Bishop, 2003). Widespread Internet connectivity has made introduction of malicious software through network connections a frequent vector of such attacks. However, network connectivity is not the only such vector. Intrusions through malicious software can also be accomplished through portable storage devices if physical access is available (Stallings & Brown, 2012).

Although there are still people who attempt to intrude into computer systems for reasons of curiosity or notoriety, an important motivator is crime for financial gain (Hald & Pedersen, 2012). Although bank robber Willie Sutton denied in his autobiography that he robbed banks "because that's where the money is," (Sutton & Linn, 1976), that is

clearly a motivator today. Losses are difficult to quantify, but estimates in the billions of dollars have been given (Florêncio & Herley, 2011). Financial gain can be direct, such as use of credit card numbers or bank account credentials, or indirect, such as through the sale of stolen credentials or rental of botnet services (Egele et al., 2012). Use of intrusion for indirect gain has resulted in a very large underground economy (Zhuge et al., 2009).

Another motivator mentioned by Hald and Pedersen (2012) is cyber warfare, intrusion carried on using the resources of a national government to further that government's ends. Chen (2010) points to the Stuxnet worm as a probable example of cyber warfare based on the narrow choice of targets and the sophistication of the software. Governments may also employ malicious software for surveillance.

Malicious software can be categorized by its behavior or by its intended effect. For example, the terms "worm" and "virus" describe behavior. A worm is malicious software that is independent and self-propagating. A virus is self-propagating, but attaches itself to other software. "Spyware" and "bot" describe the intended effect. Spyware covertly retrieves information from the system under attack and sends it to the attacker. A bot is software that allows for covert remote control of the system under attack. The phrase can also refer to a system that is covertly remote controlled. A botnet is a collection of such systems (Egele et al., 2012)

There are several mechanisms for attempting to detect malicious software which are described in detail below. One common mechanism is to compare the incoming data stream to bit patterns known to characterize an attack (Egele et al., 2012). This is called signature detection. Signature detection is the prevalent approach in commercial anti-virus scanners (Stallings & Brown, 2012). Authors of malicious software attempt to

avoid signature detection by obscuring their software or changing it over time (Vinod et al., 2009).

The research reported here encompasses intrusion detection, detection of malicious software through recognition of signature patterns, cellular automata and their use in pattern recognition in the presence of noise, and the evolution of cellular automata using genetic algorithms. The literature of each of these subjects is reviewed here.

Intrusion Detection

Intrusion detection can be host-based, in which case events that cause protection state changes in host computers are analyzed, or network-based, in which case data transiting a network are analyzed (Pfleeger & Pfleeger, 2006). This research was focused on network-based intrusion detection, in which network traffic is examined for indications of malicious activity, although it is equally applicable to searching databases or files.

Bishop (2003) defines intrusion detection as monitoring to detect attempts at violating the security policies of a system, regardless of whether the attempt is successful. According to Bishop, intrusion detection systems have four goals: detecting a variety of intrusions, including novel forms of attack, detecting intrusions within an appropriate time frame, presenting an easy-to-understand analysis, and discriminating accurately between attacks and normal traffic.

Bishop's remarks on timely detection are particularly relevant to this research. He observes that not all intrusion attempts need to be detected in real time, but that they must be detected in time to take appropriate action. This research was predicated on the proposition that some kinds of events, namely those that can result in the compromise of

a computing system within seconds or minutes, should be detected and prevented in near real time if possible. Doing so can save the expense and difficulty of recovering a system that is potentially compromised (Bishop, 2003).

One goal of research into intrusion detection has been that of improving performance. Clark et al. (2004) describe a hardware platform for detecting and preventing intrusions. In this context, prevention implies detection in real time. Their research involved von Neumann architecture network processors coupled with micro-engine processors to run certain threads in parallel and a field programmable gate array (FPGA) for pattern comparison. Otey et al. (2003) describe research into intrusion detectors that are an integral part of network interface cards. Their algorithms run on von Neumann-style processors. Sekar et al. (1999) attack the problem of improving performance by improving pattern matching algorithms.

Another approach to improving performance is the use of content-addressable, or associative, memory. Yu, Katz, and Lakshman (2004) describe gigabit rate pattern matching hardware bases on ternary content-addressable memory, or TCAM. Each bit in such a memory has three states, zero, one, and "do not care." The latter state will match either zero or one in the data packet. That allows for approximate string matching. However, the position of the bits to be skipped must be known, and a successful match depends on the position of the matching bits in the data packet.

Salmela et al. (2007) recognize the need for both good performance and the ability to match multiple patterns simultaneously. In the absence of simultaneous matching techniques, the time to analyze a packet becomes the product of the time to check one pattern and the number of patterns. They address this problem using

overlapping *q*-grams and report good performance even with large numbers of patterns. *Q*-grams break a string into substrings, each of which is considered to be a single semantic token. Overlapping q-grams are formed by taking a string of length *q* from each consecutive position of the original text. In an example given by Salmela et al., an overlapping *q*-gram of length two on the string "pony" produces the strings "po-on-ny."

Zu et al. (2012) describe the use of regular expressions for intrusion detection. Regular expressions can be evaluated using finite automata. They point out that the state space of deterministic finite automats (DFA) expands rapidly to hundreds of thousands of states as the number of patterns increases, making DFA impractical for intrusion detection. Nondeterministic finite automata (NFA) can have more than one state transition on an input character, which has the effect of reducing the state space. More than one transition on an input character implies the possibility of multiple states being simultaneously active. Zu et al. achieve the desired speed through the use of a GPU to provide for parallel processing of multiple active states.

Q. Zhang et al. examine the problem of detecting encrypted network code. They observe that the decryption routine itself must be directly executable, although it may be obscured in a number of ways. The executable decryption code is often, but not always, preceded by the NO-OP sled that is characteristic of buffer overflow attacks (Q. Zhang et al., 2007).

J. Zhang et al. (2008) attempt to avoid false positives when checking signatures for polymorphic worms through characterizing normal traffic. Their premise is that the signature patterns in polymorphic worms are relatively small because a large part of the code is deliberately masked. It is, therefore, possible that the same patterns could appear

in normal traffic, resulting in false positives. They describe the characteristics of normal, non-malicious traffic by identifying strings that occur frequently in normal traffic. They call this a "white list." They then develop signatures for malicious traffic by identifying strings that occur frequently in the malicious traffic and are not on the white list. They also analyze protocol characteristics, such as the traffic flow between source and destination IP addresses. Having thus characterized malicious data, they then treat normal network traffic as noise and attempt to distinguish the malicious data from the noise.

Bishop (2003) describes three approaches to intrusion detection. Misuse detection involves comparing signatures of known attacks against current activity. In the case of network-based intrusion detection, this means network packets or sequences of packets. Misuse detection can discriminate very precisely between patterns known to represent attacks and other patterns. It will produce few false positives, but cannot detect unknown types of attacks. Tavallaee (2009) observes that signature based detection is the favored approach in current commercial intrusion detection products.

Misuse detection need not be based on explicit comparison of events to patterns. Cannady (1998) describes training a neural network to recognize events characteristic of misuse. Rhodes et al. (2000) apply Kohonen self-organizing maps to differentiate normal and malicious traffic.

Anomaly modeling (Bishop, 2003) involves looking for deviations from a statistical characterization of a normal operating environment. Unlike misuse detection, anomaly modeling can detect previously unknown types of attacks, but at the cost of a number of false positive indications that depend on the precision of the model.

Specification-based detection (Bishop, 2003) compares current activity to a formal specification of states "known not to be good" and reports exceptions. Properly-modeled specification-based detection should be relatively immune to false positives, but cannot detect attacks outside its specification model.

Another trend in intrusion detection is distribution of the detection functions. In some cases this is an outgrowth of a detection method, as with detection engines integrated into network interface cards. However, distribution is also driven by the improvements available through specialization. Rhodes et al. (2000) describe the idea of a "monitor stack" organized analogously to a network protocol stack. The principle is that misuse can best be detected at the appropriate protocol level. That drives specialization and the gains from specialization drive distribution of detection efforts and concepts like the monitor stack.

Brooks et al. (2002) describe a model of information flow based on heterogeneous and rule-heterogeneous cellular automata and conjecture that this model can be used for anomaly detection through flow modeling. The research reported here also seeks to apply cellular automata to detect anomalies. However the automata used in this work are much simpler, with the intent that they could be implemented in hardware or in software suitable for execution on computers with a high degree of parallelism, such as graphics processing units. The idea that a "normal" pattern might be recognized by a cellular automaton rule represents a different approach, although similar to the white list of J. Zhang et al. (2008).

Crowcroft et al. (2003) characterize the state of the Internet in about 2002 as being composed of a 10Gb/s core infrastructure with an access infrastructure of at most

100Mb/s Ethernet. They recognize a cycle in which access speeds increase, necessitating an increase in core speeds. In 2003 they observed the beginning of a trend towards gigabit access speeds. We are currently on the access speed part of the cycle, with access speeds trending from the 100 Mbps connections of a few years ago to gigabit connections.

If a goal is detection quickly enough to discard malicious packets and so prevent intrusion, the bar is raised by the increase in access speeds. A reasonable assumption is that most (but not all) intrusion attempts will come from outside. In that case, "real time" means at the signaling speed of the access connection. Today, that means gigabit rates. Current intrusion detection systems are operating at the edge of their capabilities. It will be necessary to find improved mechanisms to safeguard the next generation Internet (Otey et al., 2003).

Cellular Automata

A cellular automaton is a discrete, deterministic, dynamical system (Wolfram, 1984). It consists of a lattice, or "game board," of n cells, an alphabet k of possible states for each cell in the lattice, a generator function, and an initial condition. The lattice and alphabet reflect the discrete nature of cellular automata. The generator function, also called a *rule*, specifies the next state of each cell in terms of the current state of the cell and the current states of its near neighbors (Wolfram, 2002). It is this property that allows fully parallel implementation of cellular automata. Sommerhalder and Westrhenen (1983) view one-dimensional cellular automata as a collection of Moore model finite state machines, one for each cell of the cellular automaton. Such a

collection of finite state machines can be directly implemented in hardware, although the cellular automata used in this research were simulated in software.

"Near" is defined by a radius *r*. The generator function examines each cell and its *r* neighbors on either side. The radius is generally small; cellular automata operate on local interactions only. The cellular automaton described below has $r = 1$, so that the generator function considers the cell itself and its left and right neighbors. The initial condition is the set of states at time t_0. Even extremely simple cellular automata such as these can exhibit complex behavior (Wolfram, 1994).

Binary cellular automata are those with alphabet $k = 2$, only two possible states per cell. Other numbers of states, or alphabets, are equally possible. Cellular automata for which the lattice is one cell high by *n* cells wide are one-dimensional cellular automata. A cellular automaton in which the number of states is different for different cells is called heterogeneous. If different cells have different generator functions, the cellular automaton is called rule-heterogeneous (Wolfram, 2002). Given these variations, the descriptions of cellular automata can become quite complex.

The research described in this report focused on implementations that can take advantage of parallel computing using simple computing elements, and so employed binary, one-dimensional, rule-uniform cellular automata with $r = 2$.

A striking feature of cellular automata is that some of them are self-organizing. Some cellular automata "evolve" to the same patterns even when given random and differing initial states (Wolfram, 2002). Cellular automata constructed from such rules exhibit a lack of sensitivity to initial conditions that may be interpreted as immunity to noise in initial conditions (Wolfram, 2002). For that reason, a cellular automaton used in

the way that is described in this report intrinsically does an approximate string search; perturbation of a number of bits in the target does not significantly impair the ability of the cellular automaton to recognize the pattern.

Cellular automata that reach a particular, recognizable state in this way are said to *relax*, even though the pattern need not be repeating and need not converge to any particular value (Wolfram, 2002). A state to which a cellular automaton relaxes is called a basin of attraction by Ganguly et al. (2002). They point out that a particular cellular automaton may have more than one basin of attraction, and in that way recognize more than one pattern. In the trivial state of relaxation, each cell repeatedly produces either zero or one. However, any identifiable pattern meets the definition of relaxation and of a basin of attraction.

Other cellular automata are very sensitive to initial conditions; a small perturbation in initial conditions can lead to large differences in results (Wolfram, 2002). It is this sensitivity to initial conditions that has made possible the differentiation of network packets reported here.

Wolfram (1994) describes a class of one-dimensional cellular automata in which the alphabet is binary (0/1) and the generator function F is:

$$a_i^{(t+1)} = F(a_{i-1}^t, a_i^t, a_{i+1}^t)$$

where a is the value of a cell in the cellular automaton, i is the position of the cell, and t is a discrete time interval or generation number. This generator describes a rule for producing the next generation of cells that looks only at the current cell and its left and right neighbors, so r =1. Since the alphabet of this class of cellular automata is binary, there are only eight possible inputs to computation of the next generation: 000, 001, …

110, 111. Each specific rule is characterized by an eight-bit number, one output bit for each of the possible input combinations. That means there are 2^8=256 distinct cellular automata that can be described with this general function and a binary alphabet. Wolfram's (1994) convention of referring to these rules by their characteristic output function has been widely adopted, so one speaks of rules zero to 255 for such a cellular automaton.

Chaudhuri et al. (1997) describe how a generator function like that given by Wolfram (1994) can be implemented with combinational logic and a state register, but also show how the generator function can be implemented in hardware or software as a table look-up. For example, the combinational logic for rule 90 is:

$$a_i^{(t+1)} = a_{i-1}^t \oplus a_{i+1}^t$$

The same rule can be represented as a table look-up as follows:

Neighborhood:	111	110	101	100	011	010	001	000
Next state:	0	1	0	1	1	0	1	0

The "neighborhood" represents the bit whose next state is to be determined, together with its left and right neighbors. The three bits of neighborhood can be used as an index into a table that holds the next state. For cellular automata of r = 1, the neighborhood is three bits and the table has eight entries, as shown. When r = 2, there are five bits and 32 table entries. Note that the zero value of the neighborhood refers to the low-order bit of the rule. The table can be implemented in software, as in this research, but it could also be implemented in hardware as a control store.

Some of the cellular automaton rules examined by Wolfram (2002) are symmetrical about one another, and some, like rule zero, degenerate immediately. There are 88 distinct and interesting cellular automata of this type. If there were only 88, or

even 256, possible binary, one-dimensional cellular automata, one could determine by exhaustive search whether any of them can partition network packets into normal and malicious categories. However, there are many other possible binary, one-dimensional cellular automata. Looking at the two nearest neighbors on each side, $r = 2$, (five bits in total) gives a generator function of five bits, $2^5 = 32$ input values and 2^{32} possible combinations. In general, the number of bits in the rule space is $b = 2^{2r+1}$ and the number of possible rules is 2^b (Mitchell et al., 1996). Thus, one goes from 256 rules when $r = 1$ to 2^{32} rules when $r = 2$ and 2^{128} rules when $r = 3$. Other generator function classes and different combinations can raise the number of possible combinations exponentially. Regardless of the choice of r, cellular automata consider only nearby cells. It is for precisely this reason that the characteristics of malicious packets can be encoded in a cellular automaton rule. It also makes imperative the use of an approach like genetic algorithms; an exhaustive search of such a large rule space is impossible in a reasonable amount of time (Mitchell et al., 1996).

Cellular Automata for Pattern Recognition

Although the use of cellular automata for pattern recognition dates back to Smith's work in 1971, there is also current research in the area, particularly with respect to error correction, approximate matching, and the ability of a single cellular automaton to detect two or more patterns. Each of these areas is relevant to the results reported here.

Chady and Poli (1997) worked with small feed-forward cellular automata. Feed-forward cellular automata are two-dimensional cellular automata for which the update rule allows propagation in only one direction. One can visualize such a cellular automaton as a rectangle in which the input bits are applied on the left and the result or

output bits appear on the right after a number of cycles. Chady and Poli applied small (8×8 and 16×16) feed-forward cellular automata to associative memory look-ups in the presence of noise. Their cellular automata can recognize up to four patterns with error correction capability of up to 20% noise applied to the input. Their cellular automaton rules were found using genetic algorithms, as is the case for this research.

Chady and Poli observe that the 8×8 cellular automaton performs better than the 16×16 version and conjecture that it is because the portion of the pattern analyzed by a single cell is greater than the noise applied to the input (Chady & Poli, 1997).

Brewer (2008) shows that the propagation property for cellular neural networks described by Chua and Yang (1988) also applies to cellular automata. The propagation property states that, as the number of cycles increases, each cell of a cellular automaton is influenced by an increasingly large area. This is a helpful counter to the conjecture of Chady and Poli because, as the number of cycles of the cellular automaton increases, each cell is influenced by an increasing portion of the input. The propagation property does seem to suggest that the number of cycles needed for pattern recognition may increase with increasing input sizes. For the research reported here, operation in four or eight cycles was found to be effective.

More recent work by Saha et al. (2002), Maji et al. (2003), and Ganguly et al. (2004) describe one-dimensional cellular automata that can recognize patterns in the presence of noise. The cellular automata described in these papers are rule-heterogeneous, that is, each cell may have a different generation rule. Most rules appear to be hand-crafted although the authors suggest finding appropriate rules using genetic

algorithms. The paper by Saha is specific to the discussion of evolving such rules through use of genetic algorithms.

Latif et al. (2010) have shown that a classification mechanism based on cellular automata for functional magnetic resonance imaging brain scans produces better results than singular value decomposition (SVD). The alphabet of the cellular automaton consists of the voxels of the fMRI image and the transition rule is based on the voxel distance of a cell's adjacent neighbors. The cellular automaton approach also provides better time performance than SVD. Because sorting the voxels is required, the cellular automaton approach requires time $O(n\log_2 n)$ while SVD requires $O(n^2)$.

Kundu and Roy (2010) suggested using cellular automata with multiple basins of attraction as classifiers for Web pages. Their paper describes a classifier for a relatively small lexicon and applies rule-heterogeneous cellular automata to the problem.

Brewer (2008) also points out that homogeneous cellular automata are translation invariant; that is, the position of a feature in the input space will not affect how it is processed. The implication is that rearrangement of the features of a malicious packet may not prevent it from being recognized as such. Ganguly et al. (2004) observe that "the time to recognize a pattern is independent of the number of patterns stored." Adding more signatures to be compared does not increase the time to perform the check provided the number of cycles of the cellular automaton can be held constant.

Use of Genetic Algorithms to Evolve Cellular Automaton Rules

The generator function described in Wolfram (1994) and given earlier in this paper defines 256 different cellular automata. Each rule can be specified as an eight-bit number and permuting the number in some way generates a new rule.

Matthew Cook (2004) showed that there exist cellular automata capable of universal computation by showing that the cellular automaton defined by rule 110 is equivalent to a universal Turing machine. According to Church's conjecture, every computable function can be computed by a Turing machine (Wood, 1987), so a proof that a cellular automaton is equivalent to a universal Turing machine provides the strongest possible evidence that the cellular automaton is capable of universal computation.

According to Crutchfield et al. (1998), even though there exist cellular automata capable of universal computation, there is no satisfactory programming paradigm for harnessing the inherent parallelism of cellular automata. Instead, programs for cellular automata are discovered through three major approaches. The first is hand-crafting on an *ad hoc* basis. The second is through the use of programs that simulate serial processes without taking advantage of the inherent parallelism of cellular automata. Finally, cellular automata for specific applications can be evolved using genetic algorithms. This research has taken explicit advantage of the parallelism of cellular automata by using genetic algorithms to evolve cellular automata that effectively classify malicious network packets.

Mitchell et al. (1994) observed that a cellular automaton that considered a neighborhood of seven cells could be represented as a 128-bit number. They considered this number to be the chromosome to be manipulated by the genetic algorithm. The size of the chromosome varies as 2^b where b is the number of bits needed to specify the rule. Even relatively large neighborhoods yield chromosomes of tractable size. A neighborhood of eleven cells ($r = 5$, eleven bits, and 2^{11} rules) can be specified using a 2,048 bit chromosome to explore the 2^{2048} possible values. Notice also that a rule-set

defining a neighborhood of *n* cells in a cellular automaton of given radius subsumes all rules of neighborhoods of smaller sizes; it is only necessary to encode zeroes for those combinations that are never used (Wolfram, 2002).

Mitchell et al. (1994) and Crutchfield et al. (1998) experimented with evolving cellular automata to do density classification by permuting the characteristic number that defines a cellular automaton's rule. They used the hand-crafted Gács, Kurdyumov and Levin (GKL) rule as a benchmark. The GKL rule has two basins of attraction; it relaxes to all zeros or all ones. Mitchell et al. (1994) describe epochs of innovation where the strategy of the genetic algorithm apparently changes. Although the cellular automaton produced by the genetic algorithm got successively better at density classification, it never achieved the effectiveness of the GKL rule. Mitchell et al. (1994) point out that the GKL rule was not invented for density classification; it was part of a study of phase transition and computation in one dimension.

The fitness function used by Mitchell et al. (1994) is that the cellular automaton relaxes to all ones when the input density is over one half and to all zeros otherwise. This is a very stringent condition. Cellular automata may relax to many other identifiable states. It is not even required that such a cellular automaton relax at all, provided it achieves a state that can be recognized easily.

The experiments reported here were designed to take advantage of the two sources of parallelism described above: the fact that a single cellular automaton is capable of recognizing more than one pattern in a single set of operations of the cellular automaton, and the fact that cellular automata are inherently parallel mechanisms that can be readily implemented using parallel processing hardware.

Realization of Cellular Automata in Software or Hardware

A cellular automaton can be realized in software by programming the generator function as described by Wolfram (1994) and iterating over the cells of the cellular automaton, and that is the approach used in the proof of concept reported here. However, doing so essentially serializes what should be a highly parallel operation. The parallelism can be preserved if a cellular automaton of *n* cells is evaluated on a computer with *n* processing elements because it is of the nature of a cellular automaton that the successor to each cell can be evaluated independently of the others (Sarkar, 2000). That suggests the possibility of a realization of the cellular automaton in a way that takes full advantage of the parallelism. Such a cellular automaton would require 12,000 computing elements to process a 1,500 byte Ethernet packet. Modern graphics cards include thousands of graphics processing units (GPUs). Programming paradigms are available to take advantage of the large number of GPUs and to increase the number of GPUs available by using more than one graphics card (Sanders & Kandrot, 2011). An implementation with 12,000 computing elements is very much within reach using commercial hardware. Such an implementation is also applicable to searching files or databases for target patterns.

Even rule-heterogeneous cellular automata with relatively large radii are simple and regular. Binary, one-dimensional cellular automata are the simplest and most regular of all. The next state of any cell can be computed with combinational logic consisting of only a few gates. Each computing element takes *n* bits of input, determined by the radius of the cellular automaton, and produces a single bit of output, the next state.

Only a single bit of state storage is needed per cell (Porter & Bergmann, 1999). This need not be a conventional memory; all that is needed is a latch, a bi-stable device

capable of storing a single bit (Katz, 1994). The next state of every cell can be computed simultaneously. The duration of the clock cycle is dictated only by the combinational logic settling time. Thus, it is reasonable to project that a cycle of such a cellular automaton could be completed in a few nanoseconds or less.

These characteristics make simple cellular automata ideal for implementation in very large scale integration. Chaudhuri et al. (1997) describe, among many other applications, the use of cellular automata to implement built-in self-test functions for other VLSI circuits. Because of the small demands for power and chip area made by cellular automata, it would be practical to include such an automaton in the network processing chips of network interface cards. On-chip attachment to the de-serialization shift register would make the input packet available to set an initial condition with very little circuitry and no time penalty. A stand-alone implementation could be produced using an application specific integrated circuit (ASIC).

For prototypes or lower volumes, an alternative is the field programmable gate array, or FPGA. Porter and Bergmann (1999) describe the use of cellular automata implemented in FPGAs for evaluating fitness functions in genetic algorithm processing. Clark et al. (2004) and Singaraju et al. (2005) incorporated FPGA hardware in their hardware-based intrusion detector. An FPGA implementation of cellular automata was described by Sommerhalder and Westrhenen (1983).

Regardless of whether ASIC, VLSI, or FPGA technology is chosen, an important feature of a hardware-based cellular automaton is that it can be made reconfigurable if a control store consisting of a few bits per cell is provided. The size of the control store is determined by the maximum radius of the cellular automaton. A cellular automaton with

r=2 has 2^5=32 possible input states, so a 32 bit control store is needed. If an input to the generator function is provided for each of the neighbors of a particular cell, AND gates connected to appropriate bits of the control store can select which neighbors participate in the generation function. In such a design, the control store is used only to drive combinational circuits, and not as a conventional memory, so there is no memory cycle and no time penalty other than one additional gate delay to making the cellular automaton reconfigurable (Clark et al., 2004).

The ability to reconfigure a hardware cellular automaton means that if a more effective pattern matching rule is developed, hardware already in the field could be upgraded to the newest configuration. A possible disadvantage is that the control store itself must now be protected from attack.

Chapter 3

Methodology

Introduction

A series of experiments was conducted during the development of the idea paper for this research to validate that at least some types of malicious network packets can, in fact, be discriminated from ordinary network traffic using cellular automata. Wolfram (1994) uses k to refer to the number of values possible for a cell in a cellular automaton, and r to refer to the number of adjacent cells that participate in computation of the next state. A binary cellular automaton has $k = 2$. When $r = 1$, the cell and its immediate left and right neighbors participate in the computation of the next state.

The preliminary experiments were confined to cellular automata of form $k = 2$ and $r = 1$. There are 256 such cellular automata (Wolfram, 1994), a sufficiently small number to allow for exhaustive testing. A cellular automaton can be considered to be linear, in which case the cells before the first and after the last are assumed to contain zeros, or circular, in which case the first and last cells are assumed to be adjacent (Wolfram, 1994).

A one-dimensional cellular automaton simulator of 80 cells and capable of displaying 80 cycles was programmed. The simulator took parameters of rule, initial state, and whether the cellular automaton was linear or circular. The rule is specified as a decimal number 0...255, each representing one of the 256 possible rules for $k = 2$ and $r = 1$. The initial state is entered as up to 10 pairs of hexadecimal digits. Hexadecimal input was chosen to allow for binary data in the packets to be examined.

The two attacks simulated were a buffer overflow attack and the backslash attack against an Apache HTTP server.

Detection was deemed to be successful if the cellular automaton reached a detectable state when presented with a packet containing the attack signature, and not for other packets.

Because it was easy to do so, all 256 possible one-dimensional cellular automata of type $k = 2$ and $r = 1$ were tested. Class 1 cellular automata are those that quickly reach a stable state regardless of input (Wolfram, 2002). As expected, the class 1 cellular automata and several others, such as the identity rule (rule 204) were not useful in differentiating potentially malicious packets from others. However, other rules gave more hopeful results.

The first test was for a simulated buffer overflow attack. For testing purposes, it was assumed that the attack is characterized by a "NO-OP sled" of four or more i86 no-operation instructions, hexadecimal 90 (Berghel, 2003).

Rule 90, which Wolfram (1994) classifies as an additive class 3 rule, produced a line of one-bits in the third cycle when presented with a string of hexadecimal 90. Not surprisingly, it reacted the same way when presented with a string of hexadecimal 09, but did not react in this fashion for other repeated values, including hexadecimal 80 and A0. Thus, rule 90 was shown to indicate the presence of hexadecimal 90, and so to indicate the presence of the NO-OP sled, but gave a false indication if confronted with hexadecimal 09. Rule 94 behaved similarly.

At first inspection, rule 102 also appeared to behave similarly. However, what was being detected was repetition, and not a specific sequence of characters. Rule 104

similarly detected repeating characters. Rule 126 appeared to detect any repeating pattern.

Rule 129 built an inverted triangle of all one-bits under the part of the packet containing hexadecimal 90 in the fifth cycle. That area was all zeros in the previous cycle. Further, the left bits of the pattern were discernibly different between 90 and 09 if the digit to the left is non-zero. Thus, rule 129 detected the NO-OP sled with fewer false positives than rule 90. Rule 161 behaved similarly to rule 129.

A second round of preliminary experiments tested data from the backslash attack. This attack is effective against some HTTP implementations on operating systems that use the backslash as a path delimiter, *e.g.* Microsoft systems, and is not the same as the "back attack" discussed later. It depends upon the use of backslash characters to traverse the file system tree to reach areas outside the server's document root. The backslash characters can be explicit or URL-encoded. The ten-byte string, "a.u\..\..\" was tested against the 256 cellular automata. Rule 183 detected this pattern by producing a very high ones-density in the area of the backslash-dot data in cycle four. This rule was effective for both linear and circular cellular automata.

When the backslashes were URL-encoded, that is, represented as %5C, rule 183 again produced a high density of one-bits in cycle four. In addition, rule 62 produced a high density of one-bits in cycle 1. Other encodings of the backslash attack are possible, for example, %255C where %25 encodes the percent sign (Mahoney & Chan, 2003). Only the two encodings mentioned above were tested.

The preliminary testing described above tested patterns associated with attacks using the simplest possible binary, one-dimensional cellular automata, namely those with

$r = 1$. The actions of the cellular automata were not codified in recognition rules. The tests did not recognize actual attack signatures nor was there any attempt to recognize more than one pattern with a single cellular automaton rule. Rather, those tests provided motivation for the research reported here.

The principal goal of the research reported here was the discovery of cellular automaton rules that can distinguish malicious traffic within the context of a narrow definition of "malicious packet" and for more than one kind of malicious packets. Discovery of two such rules, as reported below, establish the proof of concept. The resulting mechanism can be used for either offline searching or real-time identification of the specified patterns.

As explained in the research methodology below, the experimentation reported here was conducted with a rule-uniform, binary, one-dimensional, linear cellular automaton. Much more complex cellular automata are possible, but the research objective was to discover rules that could later be implemented readily in hardware or using parallel processor computers. That objective implied a simple cellular automaton and an evaluation function that could be computed in a fully-associative manner, for example, using only combinational logic.

Overview of Research Methodology

The research approach used was construction of software prototypes and demonstration that the prototypes addressed the research questions outlined in Chapter 1 above (Baskerville et al., 2009). Experimentation concluded successfully with a proof of concept implementation of a cellular automaton that could detect either of two actual attacks from the experimental data set.

Wolfram (1994) and others described the parallel evaluation of cellular automata. That parallelism is important in a fully-functioning detector, but the work of others made it unnecessary to demonstrate it experimentally. Instead, a software implementation was used to demonstrate that the concept was viable.

Because the use of genetic algorithms to find suitable cellular automata is part of developing a detector, not a part of its operation, it should remain a software component even if a hardware-based detector were developed. The implementation in hardware of the detector is left for future research.

This research was conducted in four phases. Phase one was the specification and establishment of test data. Both normal and malicious test data packets were taken from the DARPA Intrusion Detection System Evaluation data for 1998, described by Haines (2001). The suitability of this data was discussed in Chapter 1. This is the same type of test data set used by Rhodes et al. (2000) and others and was chosen for this project expressly because it has already been shown that it is possible to distinguish between normal traffic of this type and malicious traffic. What was tested here is the ability to achieve similar results using a cellular automaton as the detector.

Phase two was implementation of the cellular automaton and genetic algorithm test beds. The cellular automaton test bed was produced first. Experimentation began with the simplest type of binary, rule-uniform, linear cellular automaton. Initial testing was performed using binary, rule-uniform, linear cellular automata with $r = 1$ and 256 cells. This is the same configuration used by Wolfram (2002), and correctness of the implementation was confirmed by duplicating several of Wolfram's results.

The preliminary experiments described above had shown promise in detecting malicious network packets, but also showed that a cellular automaton with $r = 1$ was not sufficient to the task. The experiments described here began with a binary, rule-uniform, linear cellular automaton with $r = 2$, with the intention of trying successively larger values for r until a successful rule was found. It was not necessary to go beyond $r = 2$. This research has shown that a cellular automaton of $r = 2$ can reliably detect the two malicious packets selected from the test data. Had that not been the case, further experimentation would have been conducted using $r = 3$ or with other forms or cellular automata. Recommendations are in Chapter 5.

The software used for genetic algorithm development was JGAP, the Java Genetic Algorithm Program, developed by Klaus Meffert, Neil Rotstan, and others (Hall, 2013). It was chosen over other genetic algorithm platforms for a number of reasons. It is under active development, the version that was used for this research having been released in April, 2012. It is compatible with current releases of the Java platform. It is free software, licensed under the Free Software Foundation's lesser GPL and available from SourceForge. Most important, it is highly modular, released as source code as well as compiled classes, and amenable to modification by the researcher. The JGAP fitness function is a class that receives a chromosome and returns a fitness number. The fitness function class was developed in Java as part of this research and is described below.

In the default configuration of JGAP, the crossover operation randomly selects 35% of the population and produces two new individuals from each crossover operation, using a random crossover point. By default, mutation is applied to 1/12 of the population, so the probability of a gene being mutated is $1/12 \times p \times c$ where p represents

the population size and *c* represents the chromosome size (Hall, 2013). The selector is elitist and returns the top 90% of the population as ranked by fitness. After selection, the top 10% of elements by fitness are cloned to return the population size to 100% (Hall, 2013).

The default values supplied by JGAP were used in these experiments. The chromosome used consisted of a single gene, the cellular automaton rule to be evaluated. The datatype of the gene was defined to be a 32-bit integer with a range of -2^{31} to $+2^{31}-1$, allowing all 32 bits to be varied. The population size was set to 100 and the number of evolutions was set to 50. Initial populations were selected randomly by JGAP. As there was only one gene in the chromosome used, crossover is not meaningful in the experiments reported here. The fitness function used was developed for this research and is described below.

Although parallel implementations of genetic algorithms are possible (Shonkwiler, 1993), parallelism in the operation of the genetic algorithm engine was not required because the operation of the genetic algorithm is not a part of the actual detector, only a tool to find suitable cellular automata. Therefore, performance of the genetic algorithm is not relevant to the operational performance of the pattern matching engine. Each run of the genetic algorithm took 45 to 60 seconds on the equipment used.

The third phase of research was development of a fitness function. Previous work, such as the density classification work by Crutchfield et al. (1998), used relaxation to a state of all ones or a state of all zeroes after a specified number of iterations, in their case, 320. This is quite a restrictive measure. Sarkar (2000) suggested that a single bit is sufficient to represent an accepting or rejecting state, provided it is reliably on or off.

Although fitness functions could possibly have been determined by cluster analysis, the only suitable functions are those that can be computed by combinational logic or direct comparison. That follows from the research goal of finding an approach that would later be suitable for implementation in a highly parallel fashion.

The first attempt at a fitness function selected for a cellular automaton that generated high ones densities for target packets and much lower ones densities for packets not in the target population. Subsequent refinement produced a fitness function that produced a particular range of ones densities for the target packets and ones densities outside that range for packets not in the target population.

The fourth phase was testing the cellular automaton rules, evaluation of their effectiveness, and iterative refinement of the fitness function. Inputs to the test runs were a cellular automaton rule and associated evaluation rule, a cycle limit, and data packets from the MIT/DARPA IDEval data set. Initial testing was performed using only the Friday, week two test data of the IDEval data set. The test bed program read the entire data set and evaluated each packet according to the cellular automaton rule and also using string comparison rules hand-crafted to detect the specific attacks chosen for the demonstration. Use of such string comparisons was necessary to allow evaluation of the effectiveness of the cellular automaton as detector by providing an independent check against the data. Initial experiments were performed using the back attack packets from the Friday week two data of the training data set. When back attack packets could be detected reliably, the IMAP attack packet from the same data set was added.

Specific Research Approach

The MIT/DARPA IDEval data set from 1998 was used as the source of test data. Rationale for selection of that data set has been described in Chapter 1. The data set itself is described in Haines (2001) and in Lippmann (2000).

The JGAP program was installed and tested using the *makeChange* example fitness function that is provided with JGAP. A cellular automaton simulator program using the table look-up approach described in Chapter 3 was written and validated by duplicating several of Wolfram's (2002) results using cellular automata with $r = 1$. Once validated, the cellular automaton simulator was extended to 12,000 cells, representing the 1,500 bytes of an Ethernet packet, and radius $r = 2$. Although the cellular automaton accommodates a full 1,500 byte Ethernet packet, only the payload data is used for pattern matching.

Because the chosen test data are in TCPDUMP[2] format (Haines et al., 2001), Java programming to read that format was required. The most direct approach appeared to be the use of the SJPCAP program, published anonymously on Google Code. Testing showed that SJPCAP did not correctly account for whether the files to be parsed stored integers in big endian or little endian format, and in fact, mixed modes between 16 bit numbers (little endian) and 32 bit numbers (big endian). According to Mahoney (2003), some of the MIT/DARPA IDEval files are in big endian format and some are in little endian format. It was necessary to correct SJPCAP to handle endian-ness properly based on the "magic number," 0xA1B2C3D4, that is included for that purpose in the global header

---

[2] TCPDUMP and Ethereal, later renamed Wireshark, are utility programs for capturing network traffic, saving files of captured traffic, and examining such files. They are described and compared by Fuentes (2005).

found in TCPDUMP files. There was already a defect report about conversion of shortInt being little endian on the Google Code page for SJPCAP. As a result of this research, the Google Code page has been updated with a brief discussion of endian-ness of input files and correct use of the "magic number."

Characteristics of the Malicious Packets

Two types of malicious, or attack, packets were chosen for experimentation. They are called "back attack" and "IMAP attack" in Lippmann (2000) and in the MIT/DARPA documentation. Attack packets in the Friday, week 2 training data set were inspected with Wireshark to determine their characteristics.

Each IMAP attack pattern used in this research was a single packet consisting of the string "301 LOGIN" followed by 0x22 and the series of 0x90 bytes that characterize the NO-OP sled of a buffer overflow attack. Shell code following the NO-OP sled leads to a root compromise. The IMAP attack packet used in the genetic algorithm was packet number 175,454 from the Friday, week two training data. That is the only IMAP attack packet in that particular data file.

The back attack is a denial of service attack effective against versions of the Apache web server in use at the time. Despite the name of the attack, the 0x2f character is the forward slash. The back attack consists of two slightly different types of packets: initial packets and continuation packets. Initial packets consist of the string "GET /cgi-bin" with the remainder of the packet filled with 0x2f characters to a total of 1,460 bytes. Continuation packets consist of HTTP continuations with up to 1,460 bytes of 0x2f characters. When a network attack consists of multiple packets, as with the back attack,

it is important to detect the first packet so that it and subsequent packets of the attack can be blocked.

Non-Target Packets

Two packets were chosen arbitrarily from the Friday, week two training data as "non-target" or non-malicious packets used for comparison in the genetic algorithm. The first was packet number 825,453, a relatively short HTTP GET packet. The TCP packet length was 328 bytes. The second was packet number 1,819, a full-length packet of text HTML markup.

Development of Fitness Functions

Although the fitness function was refined iteratively during experimentation, all fitness functions used were based on the ones density of the cellular automaton after a predefined number of cycles. Only binary ($k = 2$) cellular automata were tested, so the ones density is the number of one-bits in the cellular automaton's cells after the final cycle divided by the number of cells. It is important to note that for the rules which were developed, the packets under test are of variable length, from a few bytes to 1,460 bytes. The cellular automaton under test was of fixed size, 12,000 cells, equivalent to the 1,500 byte size of an Ethernet frame. When a packet is loaded into the cellular automaton, remaining bits on the right are filled with zeros. The ones density calculation is over all 12,000 bits of the cellular automaton, and not over the area defined by the packet. The propagation property described by Brewer (2008) says that, as the number of cycles of a cellular automaton increases, the number of cells influenced by each initial bit also increases. After some number of cycles, the fact that a portion of the cellular automaton was filled with zeros is much less significant.

It is not enough for the cellular automaton to reach an identifiable state in the presence of a target packet. It must also reach some distinctly different state in the presence of non-target packets. So, cellular automata must be tested by the fitness function against both target and non-target samples from the data set. Development of fitness functions began with the hypothesis that the ones density of target packets would be different from the ones density of other packets. Experimentation confirmed that hypothesis for the two types of target packets tested.

Initial testing was conducted using only a back attack packet and a fitness function that attempted to find a rule that produced a ones density of less than 0.1 or greater than 0.9, *i.e.* nearly all ones or nearly all zeros for the target packets, and a density between 0.1 and 0.9 for the non-target packets. The same cellular automaton Java class used for pattern matching was used to determine densities in the fitness function. The number of cycles was adjusted iteratively, starting with 64 cycles and adjusting downward. A fitness function with ones density of target packets less than 0.1 or greater than 0.9 proved to be ineffective when tested against the full data set.

The first refinement was to look only for very dense configurations of the cellular automaton, with ones densities > 0.9 for the target packets and less for non-target packets. Although this appears to be more restrictive than the first fitness criterion, the genetic algorithm generated more effective cellular automaton rules.

A fitness function based on the average Hamming distance between target and non-target packets after operation of the cellular automaton was considered. This approach was discarded because of the computational effort needed to evaluate the result.

The fitness function that was adopted for the experimental phase was a density band approach. The assumption that the target packets would have a ones density different from that of the non-target packets was carried forward. The difference from previous attempts was the hypothesis that the target packets would occupy a density range of T_{min} to T_{max}, with few or no non-target packets also generating densities within that range. The fitness function also computed N_{min} and N_{max}, the minimum and maximum ones densities of the non-target packets. Using this approach, the genetic algorithm generated the cellular automaton rules discussed in the next chapter. The fitness function used was $T_{min} - N_{max}$ for $N_{max} < T_{min}$ and $N_{min} - T_{max}$ for $T_{max} < N_{min}$ with the added condition that no non-target's ones density be allowed to fall between T_{min} and T_{max}. If that occurred, fitness was set to zero. This revision of the fitness function differentiates ones density of target packets from that of non-target packets, but without the artificial high or low density requirement of the prior attempts. In addition to computing a fitness number, the fitness function logged the density values for later use.

Testing of Generated Cellular Automaton Rules

All IDEval test data containing either back or IMAP attacks similar to the packets selected for testing were used for testing. Each day's TCPDUMP file was first scanned by a program that counted and identified the two attacks under study using hand-crafted string comparison rules. That provided comparative data for evaluation of the effectiveness of the cellular automaton rules. The IMAP attacks were identified by looking for "301 LOGIN" in the first bytes of the packet, followed by 0x22 and five bytes of 0x90. Back attack initial packets were identified by checking for the string "GET /cgi-bin/" followed by only 0x2f characters to the end of the packet. Continuation packets contain only the

0x2f character; generally the entire packet is filled, but this was observed not to happen always. In particular, the final continuation packet was often not filled. Early in the experimental phase, the classification code was merged into the program used to test the cellular automata filters, rendering the separate classification program unnecessary.

Once the data files were characterized, the cellular automaton rules generated by the genetic algorithm were tested using a program that read the MIT/DARPA IDEval data and evaluated each packet in two ways. The first evaluation was based on the ones density classification into target or non-target packets. Experimentation showed that most rules performed better with a single comparison of ones density to the upper limit, rather than comparing for both upper and lower limits. Each packet was then evaluated according to the manual rules described above. Packets classified as target or malicious packets according to ones density but not identified by one of the manual classification rules were counted as false positives. Packets identified by the manual classification rules but not by the ones density rule were counted as false negatives and, for back attack packets, further divided according to the rules above. Because the two attacks are based on TCP packets, only TCP packets were evaluated. Similarly, because the two attacks were external attacks, only packets entering the test network from outside were evaluated. Incoming packets were identified as those not on the list of internal addresses provided with the IDEval data.

To check a packet for the presence of one of the patterns in the pattern set, the payload portion of the packet was used to establish the initial state of the cellular automaton. Bits of the cellular automaton not initialized from the payload were filled with zeros. The cellular automaton was allowed to operate for the number of cycles

identified during determination of the fitness value. During operation, the next state of each bit was determined by using the current state of that bit and its two left and two right neighbors, for a total of five bits. That five-bit number was used as an address for a table look up, as described on page 22. After the specified number of cycles, the ones density of the cellular automaton was determined by counting the one bits of the state vector. If the ones density was less than or equal to the target maximum density for the rule under test, the packet was determined to have matched a pattern in the pattern set. The rules tested, their fitness factors, and the density values for both strict and relaxed comparison are shown in Table 16. Detailed results for each rule tested are shown in Table 2 through Table 13.

Operation of the cellular automaton is visualized in Figures 1, 2, and 3. The rule used is 1205310289. In order to produce a visualization that fits the page, only 96 bits of the cellular automaton are shown. In Figure 1, the cellular automaton is initialized with the first 96 bits of the payload portion of packet 1,819 of the Friday, week two training data, representing a non-target packet. The initial ones density is 0.520. The behavior of the cellular automaton is apparently chaotic, producing no identifiable pattern. The cellular automaton has a ones density of 0.542 in cycles one, two, and three. In cycle four, the last cycle for rule 1205312089, the density is 0.406, which is greater than the threshold density for that rule (see Table 16) with the result that the data do not match either pattern.

Figure 2 shows the same 96 bits, initialized with 0x2f characters, representing a back attack packet. Initial ones density is 0.625. After two cycles, a clear pattern of four zero bits followed by four one bits emerges. The density after cycle two is 0.479. In

cycle three, the pattern becomes alternating ones and zeros and the density is 0.510. In cycle four, nearly every bit is zero, the density is 0.073, and the packet is identified as matching a target pattern in the pattern set.

Figure 3 demonstrates the initialization of the cellular automaton with 0x90 characters, representing the IMAP attack. The initial ones density is 0.250, which is below the threshold for selection. However, the initial density is not used in the algorithm. By cycle two, the same pattern of four zero bits followed by four one bits has emerged; the density is 0.489. In cycle three the pattern again becomes alternating ones and zeros with a density of 0.500. In cycle four, nearly every bit is zero, the density is 0.042, and the packet is identified as matching a target pattern in the pattern set.

The problem addressed by this research is one of approximate string matching; a certain number of incorrect matches are to be expected. The accuracy of such matching can be presented as sensitivity and specificity. The sensitivity of a test is the number of cases correctly identified divided by the total number of instances of the target value in the population tested. The specificity of a test is the proportion of true negatives divided by the total number of non-target instances in the population tested. False positives are 1 – specificity, and false negatives are 1 – sensitivity (Jaeschke et al., 2006). The classification scheme described above provided the measures of sensitivity and specificity given in Chapter 4.

Evaluating Tolerance for Perturbation

Each cellular automaton rule tested was also evaluated for sensitivity to perturbation, or noise. This was accomplished by taking a single sample packet for the back attack and the IMAP attack and randomly inverting bits using sampling with

replacement. From one percent to 25% of bits were modified, in steps of one percent. Cases where the ones density produced by operation of the cellular automaton was outside the range generated by the genetic algorithm were recorded. The number of bits changed while still remaining within the specified ones density was expressed as a fraction of total bits in the packet.

Experiment showed that the IMAP packet was the determining factor in establishing the target maximum ones density. Even a small perturbation of the IMAP packet often exceeded the density bound. To compensate for this, the density bound was relaxed by adding 25%, determined by experiment, of the distance between the target band and the non-target band. The perturbation tests performed prior to adding the tolerance band are not reported here. Tests of the sensitivity and specificity of the generated rules were re-run, and both sets of results are reported here.

The IMAP attack consists of a single packet, and the packet used for operation of the genetic algorithm was also used for testing sensitivity to perturbation. The back attack consists of multiple packets; only the initial packet was tested for perturbation. The basis of that decision was that detecting the initial packet is the requirement for identifying the attack.

Resources Used

All research described here was completed with a standard office computer and can be readily duplicated without special equipment. The operating system used was Windows XP-SP3 with current Microsoft patches as of the time of the experiments. The programming language was Java, using the Java 7 runtime environment and the jGRASP development environment. Genetic algorithm functions were performed by JGAP 3.6.2.

The test data were obtained from the MIT IDEval pages in TCPDUMP format. Wireshark version 1.4.15 was used to extract test cases and when it was necessary to examine the test data directly. Microsoft Excel 2007 was used to import the text files of false positives and false negatives produced by the experiments. Conversion from the binary TCPDUMP format to Java objects used SJPCAP, a program contributed anonymously to Google Code, and which received substantial revision by the author to handle the MIT IDEval data as described above. All other programming was done in Java by the author.

Version control was accomplished by copying all programming files to directories named using month, day, and year prior to making changes to the programming. A laboratory notebook of the results of the experimental phase experiments was recorded in a word processing document, most recent entry first.

Summary

The research approach used was prototype and demonstration. Genetic algorithm functionality was provided by JGAP, a standard genetic algorithm implementation with substantial opportunity for customization. Only the size of the initial population, the datatype of the single chromosome, and the fitness function were customized. The fitness functions and cellular automaton evaluation programs were coded in Java by the author.

After a suitable fitness function was devised, operation of the genetic algorithm produced cellular automaton rules that were able to identify the two types of malicious packets selected for testing from the rest of the packet population. There were no false positives in the test data. No initial packets from either attack were missed. False

negatives were all back attack continuation packets. Detailed results are given in the next chapter.

Chapter 4

Results

Data Analysis

Data were collected from four of the training data sets and all ten of the testing data sets of the MIT/DARPA IDEval 1998 data corpus. Two passes were made through the data. The first pass used a strict bound on ones density as determined by the genetic algorithm. The second pass used a relaxed density rule, in which the density boundary for selection was relaxed by 25% of the difference between the target density bound and the non-target density bound. The rationale for the relaxed density rule is to improve performance in the presence of noise or perturbation, as explained in Chapter 3.

Six cellular automaton rules were tested. Rule −369784237 [3] with a fitness of 0.8530 was the first rule generated by the genetic algorithm using the final version of the fitness function, and was used as a benchmark for most subsequent testing. The remaining rules were chosen in order of their fitness values from subsequent runs of the genetic algorithm using a cycle parameter of four. The rules and their fitness values are given in Table 16.

Most of the data collected during the experiments is presented in Tables 1 – 15; a complete list of data items collected is in Table 17. Some items are omitted from this

---

[3] This report follows the convention established by Wolfram (1994) in representing cellular automaton rules as decimal integers. The programming language used for these experiments was Java, and the rules were represented as signed integers. A rule expressed as a negative integer indicates that the leftmost bit is one.

report because they are not relevant to the analysis, and one data item is omitted because it exhibited no variation. Data on detection of IMAP attack packets is omitted from the tables because there were only two such packets in the corpus of data, and all six rules detected them without false negatives. Total number of packets read is omitted from the tables because, although relevant to validating the correct operation of the experimental programs, it is not relevant to the analysis of the effectiveness of cellular automata as recognizers of multiple patterns. Only TCP packets were screened by the cellular automaton.

Desired Outcomes

There are four characteristics that together characterize successful operation of a rule for detection of the target packets. They are minimal false negative results, minimal false positive results, few cellular automaton cycles to reach an accepting state, and tolerance for noise or perturbation. Of those, minimizing false negatives is arguably the most important. Whether the cellular automaton is used for initial screening or as the only detector, a false negative means that target data will be missed.

False positives mean that either subsequent screening by another method is required or that there will be false alarms. The best rule (−369784237, see Table 2) produced 1,215 false positives on data for fourteen days, or an average of 87 false alarms each day. Such a false alarm rate indicates that some post-screening mechanism would be required to limit false alarms.

Although cellular automata operate in constant time both with respect to the number of cells (Sarkar, 2000) and, as shown by this research, with respect to the number of patterns detected, the number of cycles required to reach an accepting state does

influence performance. Best performance comes from minimizing the number of cycles. Ability to function correctly in the presence of noise means that data different from the target pattern, but similar, can be detected reliably.

Performance of the Rules under Test

An important result of these experiments is that each rule tested recognized every instance of the IMAP attack and also every instance of the back attack initial packet. The absence of false negatives means no target data was missed by any of the cellular automaton rules. Although there were false negatives for back attack continuation packets, those are not relevant to the information security application. Detecting the initial packet would enable protective action prior to any of the continuation packets being processed. Back attack continuation packets were not part of the sample data processed by the genetic algorithm, and so were expected to be recognized only to the extent that they are similar to the back attack initial packets.

Best recognition was exhibited by rule −369784237 (Tables 2 and 8), which required eight cycles to reach an accepting state. Changing from strict to relaxed density boundaries reduced the number of false negative continuation packets by 15 but increased the number of false positives by 95 cases, as shown in Table 14. Rule 1205312089 produced performance almost as good, as shown in Tables 5 and 11. The number of false negative continuation packets was identical to rule −369784237. The number of false positives increased by 756, an increase of 62%.

Best performance for noise tolerance was given by rule −369784237, followed by rule −1144617577, as shown in Table 15.

Although rule −1023231863 produced the best results with respect to false negative continuation packets, it also produced many times more false positives, as shown in Tables 4 and 10.

Analysis of False Negatives

As shown by Tables 2 to 13, false negative back attack continuation packets occur only when a back attack is present, and almost always in number slightly fewer than the number of back attack initial packets. The program used for the experiment wrote all false negative packets to text data files in hexadecimal form. Length, packet number, and computed density were recorded with each packet. These were imported into Microsoft Excel 2007 and examined manually. In every case, the packet consisted only of a string of 0x2f, with lengths varying from 40 bytes to 1,088 bytes, with 520 bytes being the most frequently occurring length. In almost every case, the number of false negatives is the same as or slightly less than the number of initial back attack packets. The exceptions are data for training week two Friday and training week three Wednesday, when some rules found slightly more false negatives. The data show that packets significantly shorter than the 1,460 bytes common to most back attack packets are not reliably recognized by the cellular automaton. This could potentially be corrected in future research by including samples of such short packets in the input to the genetic algorithm and using a rule-heterogeneous cellular automaton, as suggested in Chapter 5.

The output recorded included the packet number, making manual inspection of the original TCPDUMP file possible. A small sample of the false negative packets was examined in that way. In every case in the sample, the false negative packet was in the middle of a sequence of back attack packets. That finding disproved an earlier conjecture

that these short packets represented the last packet of a back attack. All that can be inferred from the data are that these short packets are an artifact of the attack generation process used when the data were prepared.

Analysis of False Positives

False negatives fell neatly into a single category: short packets containing all 0x2f. Characterizing the false positives is not as neatly done. However, the false positives all share the characteristic of repetition. All false positives for rule −369784237 were examined. The four from training, week 7, Friday all had long sequences of 0x90, characteristic of a buffer overflow attack. The ones from testing, week 1, Tuesday all had long, repeating sequences of 0x21. Those from Wednesday had sequences of 0x90. The same is true of testing, week 2, Monday, Tuesday, and Friday. Thursday of that week had two sequences of 0x90, with the remaining 1,191 all containing long sequences of 0x84, the Unicode "control" character. Sampling the false positives from the other rules revealed similar patterns, and on the same days. For all six rules, the largest number of false positives was testing, week 2, Thursday. The conclusion is that the cellular automaton correctly detects long strings of 0x2f and 0x90 found in the target packets, but also detects certain other repeating patterns.

Summary

All six cellular automaton rules generated by the genetic algorithm detected the two target patterns with no false negatives. False negatives for back attack continuation packets were reported because they may give some insight into the operation of the cellular automaton as a pattern recognizer. All false negatives are explained as shorter

packets of 0xf2. Four of the six cellular automata were shown to be resistant to noise injected in up to 25% of their bits.

False positives were triggered by repeating patterns in the data. At least two of the false positives were actually other attacks, namely buffer overflow attacks against a mail transport agent program as shown by the receiving port number in the TCPDUMP file. Such attacks are characterized by the same sequence of 0x90 no-operation characters as is found in the IMAP attack, a pattern specifically intended to be recognized.

The best performing rule was −369784237, which required eight cycles to reach the accepting or rejecting state. Rule 1205312089 produced results almost as good, and only one of the six, rule −1023231863 produced extremely large numbers of false positives. This demonstrates that there exist multiple rule-uniform, linear cellular automaton rules in r=2 that can detect both of the two patterns selected for study, and that such rules can be discovered through the application of a genetic algorithm.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

The research reported here has demonstrated the concept that a single cellular automaton rule can recognize more than one target pattern, can do so in the presence of noise, and with few cycles of the cellular automaton. Such a result is potentially applicable to recognizing patterns in network communications, and also to searching data repositories so large that other approaches to approximate string matching are impractical.

The proposal for this research anticipated the need for post-screening using another matching method to remove false positives. The value of using the cellular automaton would have been to reduce the number of applications of the secondary matching method to a manageable number. For the particular sample of patterns chosen, very little post screening has been shown to be necessary for the best rules. It is important to recognize that the cellular automaton mechanism provides only a binary, match / no-match result; a particular record could match any pattern in the pattern set. If a particular application makes it necessary to further distinguish data that have matched the pattern set, post screening will be necessary in any case.

Six distinct cellular automaton rules have been shown to be capable of detecting the initial packets of both the IMAP attack and the back attack. The rules studied do not always identify every packet that is part of a back attack. For example, rule 1360891913

fails to identify 1,018 back attack continuation packets in the Friday, week two training data. This is not a fatal flaw because that rule identified every initial back attack packet, with no false positives. In the case of detecting malicious network packets, with the initial packet identified, the remaining packets from the same source would be discarded without the need to examine their contents.

Because a cellular automaton can be implemented using a table look-up (Chaudhuri et al., 1997), the rule used to drive the pattern match can be changed by changing the contents of the table. There is no need to modify any other part of the mechanism. Thus, a rule for a new set of patterns, or an improved rule for a particular pattern set, can be implemented without change to the underlying mechanism.

Considerations of Timing

Cellular automata operate in constant time, as previously shown by others, including Sommerhalder and Westrhenen (1983). Because the next state of each cell in a cellular automaton can be determined independent of the next states of the other cells, the number of cells in a cellular automaton, and so, the size of the pattern to be recognized, does not influence the timing. This research has shown that, for the two patterns examined, the number of patterns tested does not influence the time required.

However, there are two considerations that do influence the timing of the mechanism described here. The first is the number of cycles of the cellular automaton required to generate a recognizable result. A cellular automaton that requires fewer cycles to reach the desired result is clearly faster (given the same implementation) as one that requires more cycles. Because there is no general programming paradigm for cellular automata (Crutchfield et al., 1998), cellular automata for a particular task must be

discovered. This research used an iterative process to discover cellular automata that would identify the patterns under study while attempting to minimize the number of cycles required. A better approach might be to incorporate the number of cycles required into the fitness function of the genetic algorithm as described below.

The other factor influencing timing is the need to differentiate between a final state of the cellular automaton that indicates recognition of the pattern, an accepting state, from states that do not, the rejecting states. For this research, the two states are differentiated by the ones density of the cellular automaton. Unlike the operation of the cellular automaton itself, the calculation of ones density does vary with the size of the cellular automaton, and so with the size of the pattern to be recognized, but not with the number of patterns to be recognized. The software used for these experiments computed the ones density by sequentially counting the one bits. Such an approach is suitable in a proof of concept demonstration such as this one, but is far too slow for practical application. It requires n additions for a cellular automaton of n bits. The computation of ones density can be parallelized if sufficient computing elements are available. Blelloch (1993) describes the reduction operation, in which the sum of a vector of n elements can be computed in time $O(\log_2 n)$ with $n/2$ computing elements. The reduction operation begins by adding pairs of elements in parallel, then adding pairs of sums, and so on until the final sum is produced. For implementation in hardware, it may be possible to construct a combinational circuit of depth two to perform the addition (Alon & Bruck, 1994). Such a circuit would operate in constant time. It is important to note that the time required for computation of the ones density depends only on the size of the cellular automaton, and not on the number of patterns it recognizes.

This research used only bit density to indicate an accepting state in the cellular automaton. Other descriptions of the accepting state may be possible, depending upon how the fitness function is designed. As noted above, Sarkar (2000) has pointed out that only a single bit is necessary to indicate an accepting state, provided it is reliably on or off. A different definition of accepting state would require a different recognition algorithm.

Implications

The research presented has demonstrated a proof of concept that a single cellular automaton can detect more than one pattern in data presented to it, in constant time with respect to the number of patterns checked, and can do so in the presence of noise. However, the sequential programming used for this proof of concept would be far too slow for practical application. The cellular automaton approach has a speed advantage only if its inherent parallel nature is exploited. A mechanism such as is described here could take advantage of the parallelism of cellular automata through implementation on a general purpose computer with parallel computing capability, or through implementation directly in hardware.

Parallel Implementation on a General Purpose Computer

Parallel computation is readily available using off the shelf computers by taking advantage of the processing units that exist on graphics cards. Modern graphics cards incorporate thousands of graphics processing units and there is programming language support for application of multiple graphics cards in a single host computer (Sanders & Kandrot, 2011). Kauffmann (2008) has demonstrated the implementation of cellular automata using a general purpose computer with graphics cards. If the mechanism

described here were implemented using the parallel processing capabilities of graphics cards, the reduction algorithm described by Blelloch (1993) could be used to compute the ones density. Although per-processor local storage on graphics cards is limited, it is sufficient to hold the cellular automaton rule used for determination of the next state as well as the state bit itself (Sanders & Kandrot, 2011).

Parallel Implementation in Hardware

Each cell of a cellular automaton can be expressed as a one-bit latch (Katz, 1994). A writable control store holding the cellular automaton rule would be required for the table look-up function. A conceptual diagram showing computation of the next state of a cell using table look-up from a control store is presented in Figure 4. The reduction method described by Blelloch (1993) is amenable to implementation as a cascade of full adders, and would still operate in $O(\log_2 n)$ time where n is the size of the cellular automaton. Such a circuit could potentially be implemented directly on a network interface card. A conceptual diagram of a cellular automaton pattern recognizer implemented in hardware is given in Figure 5.

Recommendations

The research presented here has confirmed the hypothesis that there exist cellular automaton rules that can recognize more than one pattern in constant time with respect to the number of patterns, and in the presence of noise or perturbation. It remains to be shown whether other patterns than those chosen, or more than two patterns can be matched. There may exist better fitness functions than the one presented here, or more effective arrangements of the cellular automaton. Evaluation of the cellular automaton's final state for accepting or rejecting a pattern based on ones density takes $O(\log_2 n)$ time if

implemented in parallel using multiple processing units, where n is the number of cells in the cellular automaton (Blelloch, 1993). While the time to evaluate the final state is constant with respect to the number of patterns, it may be possible to improve upon it.

Only two patterns were tested in the experiments reported here. Empirical testing might establish an upper limit on the number of patterns that can be recognized by the cellular automaton described here, or by other types of cellular automata.

Regardless of the improvements that may be possible, if best evaluation speed is to be achieved, it is clear that most of the work must be in the genetic algorithm and not in the cellular automaton. This may mean a greater focus on the fitness function than on the cellular automaton itself. In any case, changes in the structure of the cellular automaton must be reflected in the fitness function.

Modification to the Cellular Automaton

The research presented here started with the simplest possible cellular automaton, a binary, rule-uniform, linear cellular automaton. Because preliminary research had shown that a radius of one was unlikely to be effective at matching more than one pattern, experiments began with cellular automata of $r = 2$. Radii larger than two increase the number of bits of the pattern that participate in the computation of the next state of the cellular automaton, which may offer an opportunity for improvement. Increased radii also increase the number of potential rules exponentially. A radius of three examines seven bits of the pattern when computing the next state of each cell and uses 128-bit rules, of which there are 2^{128}. An increase to a radius of four would mean a rule space of 2^{512}. It is unknown whether increasing the radius would improve the pattern recognition ability of the cellular automaton, nor whether such large rule spaces can be

searched effectively using genetic algorithms. The fact that there exist several rules for cellular automata of $r = 2$ that recognize the patterns studied suggests that the same condition may hold true for cellular automata of larger radius. If that is the case, searching by genetic algorithm may be practical even in the face of very large rule spaces.

The cellular automata presented here are rule-uniform; the same rule is used to evaluate the next state of every cell. Although the rules studied produced very high sensitivity, there were false negative results for some back attack continuation packets. That is expected because back attack continuation packets were not a part of the pattern set being tested. Packets that produced false negatives for continuation packets were written to a file and examined individually. In every case, they were shorter than the packets that were detected correctly. This research was based on a cellular automaton of fixed length. The shorter packets were padded on the right with zeros to make all packets the same length. A rule-heterogeneous cellular automaton might perform better by allowing those rightmost bits to be evaluated by a different rule than the leftmost bits. Each rule would be represented by a gene in the chromosome evaluated by the genetic algorithm. Alternatively, the length of the cellular automaton itself might be varied depending upon the size of the patterns in the set of target patterns. If all states are evaluated in parallel, there is no time penalty for doing so.

Improvements in the Fitness Function

The experiments reported here identified several rules that would recognize the patterns under study. Even rules with fitness > 0.75 varied in sensitivity and specificity, suggesting that improvements are possible in the fitness function. One such

improvement might be the inclusion of more samples of the pattern to be detected in the evaluation by the fitness function. An entirely different approach to the fitness function might also produce better results.

The number of cycles required for a cellular automaton to reach an accepting or rejecting state is an important parameter with respect to performance. The cycle count was determined iteratively by experiment for the results reported here. Incorporating the cycle count as a gene in the chromosome evaluated by the genetic algorithm could potentially identify cellular automata that require fewer cycles and still exhibit equivalent performance.

More Effective Accepting State Conditions

The accepting and rejecting states for the cellular automata studied here are derived from the ones density of the cellular automaton after a specified number of cycles. The fastest programmatic computation of ones density is the reduction operation described by Blelloch (1993), which requires $O(\log_2 n)$ operations. The two-level digital logic construction described by Alon and Bruck (1994) could potentially compute the ones density in constant time, but requires direct implementation in digital logic. Speed improvements might also be possible by defining the accepting state in terms other than ones density. A single bit is enough to define an accepting state if it reliably reflects the accepting or rejecting state (Sarkar, 2000). A simpler accepting state consisting of a pattern of bits might thus be recognizable in less time than that required for computation of the ones density.

Another alternative is an accepting state that generates a repeating value in the cellular automaton itself. Such a condition can be recognized by a bitwise comparison of

current cell values of the cellular automaton with previous cell values. If the implementation includes one processor per cell, the comparisons can be made in parallel. The amount of storage for previous states depends upon the period of repetition of the pattern.

Summary

This research has produced a prototype that demonstrates the hypothesis that a cellular automaton can recognize more than one pattern using a single rule. Sensitivity was greater than 0.97 for all six of the rules tested and specificity was 1.0 for four of the rules tested. For the patterns selected for testing, very little post screening would have been necessary.

Discovery of rules suitable for pattern recognition through the application of genetic algorithms has been shown to be possible. The genetic algorithm was able to identify multiple rules that can detect the patterns chosen for these experiments.

Pattern recognition has been shown to be possible even in the presence of noise. When the density tolerance was widened to include 25% of the distance between the target density and the non-target density, four of the rules recognized the two patterns tested even when up to 25% of the bits in the packet were modified. Testing patterns for recognition using the relaxed density band produced results nearly identical to the original tests.

Experimentation was done entirely using sequential programming on a von Neumann architecture computer. Parallel implementation would be necessary to achieve processing times competitive with other approximate string matching algorithms.

Appendix A

Tables

Table 1. True incidence of target packets as determined by hand-crafted string comparison rules. For the training data sets, only those marked as having attack packets of the types under study were tested. All "testing" data sets were tested. "Total Packets" is the number of TCP packets subjected to testing. The total number of packets in each data set is larger because there are packets from protocols other than TCP. Those were not tested.

Data Set	IMAP Attack	Back Attack Initial	Back Attack Continuation	Total Attacks	Total Packets
Training , Week 2, Fri	1	1,000	36,711	37,712	215,666
Training , Week 3, Wed	0	1,000	36,729	37,729	264,776
Training , Week 6, Wed	0	100	3,694	3,794	475,551
Training , Week 7, Fri	0	108	3,507	3,615	636,290
Testing, Week 1, Mon	0	0	0	0	712,669
Testing, Week 1, Tue	0	0	0	0	719,181
Testing, Week 1, Wed	0	0	0	0	416,752
Testing, Week 1, Thu	0	0	0	0	520,950
Testing, Week 1, Fri	0	1,013	36,438	37,451	637,774
Testing, Week 2, Mon	0	0	0	0	520,025
Testing, Week 2, Tue	0	0	0	0	559,699
Testing, Week 2, Wed	1	100	3,666	3,767	40,026
Testing, Week 2, Thu	0	0	0	0	898,028
Testing, Week 2, Fri	0	0	0	0	625,445
Totals	2	3,321	120,745	124,068	7,242,832

Table 2. Performance of rule −369784237 with strict density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.22758333 after eight cycles of the cellular automaton. Rule −369784237 had a fitness of 0.8530 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,693	0	0	1,018	177,954	0.973	1.000
Train Wk 3 Wed	1,000	35,728	0	0	1,001	227,047	0.973	1.000
Train Wk 6 Wed	100	3,594	0	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	4	0	94	632,675	0.974	1.000
Test Wk 1 Mon	0	0	0	0	0	712,669		
Test Wk 1 Tue	0	0	6	0	0	719,181		
Test Wk 1 Wed	0	0	1	0	0	416,752		
Test Wk 1 Thu	0	0	2	0	0	520,950		
Test Wk 1 Fri	1,013	35,475	0	0	963	600,323	0.974	1.000
Test Wk 2 Mon	0	0	4	0	0	520,025		
Test Wk 2 Tue	0	0	4	0	0	559,699		
Test Wk 2 Wed	100	3,566	0	0	100	36,259	0.973	1.000
Test Wk 2 Thu	0	0	1,193	0	0	898,028		
Test Wk 2 Fri	0	0	1	0	0	625,445		
Totals	3,321	117,469	1,215	0	3,276	7,118,764	0.974	1.000

Table 3. Performance of rule −1144617577 with strict density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.34941667 after four cycles of the cellular automaton. Rule −1144617577 had a fitness of 0.7659 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,693	0	0	1,018	177,954	0.973	1.000
Train Wk 3 Wed	1,000	35,728	0	0	1,001	227,047	0.973	1.000
Train Wk 6 Wed	100	3,594	0	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	4	0	94	632,675	0.974	1.000
Test Wk 1 Mon	0	0	0	0	0	712,669		
Test Wk 1 Tue	0	0	6	0	0	719,181		
Test Wk 1 Wed	0	0	1	0	0	416,752		
Test Wk 1 Thu	0	0	2	0	0	520,950		
Test Wk 1 Fri	1,013	35,475	0	0	963	600,323	0.974	1.000
Test Wk 2 Mon	0	0	4	0	0	520,025		
Test Wk 2 Tue	0	0	11	0	0	559,699		
Test Wk 2 Wed	100	3,566	0	0	100	36,259	0.973	1.000
Test Wk 2 Thu	0	0	1,341	0	0	898,028		
Test Wk 2 Fri	0	0	30	0	0	625,445		
Totals	3,321	117,469	1,399	0	3,276	7,118,764	0.974	1.000

Table 4. Performance of rule −1023231863 with strict density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.51041670 after four cycles of the cellular automaton. Rule −1023231863 had a fitness of 0.8730 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,745	214	0	966	177,954	0.975	0.999
Train Wk 3 Wed	1,000	35,759	264	0	970	227,047	0.974	0.999
Train Wk 6 Wed	100	3,594	477	0	100	471,757	0.977	0.999
Train Wk 7 Fri	108	3,415	799	0	92	632,675	0.979	0.999
Test Wk 1 Mon	0	0	684	0	0	712,669		
Test Wk 1 Tue	0	0	9,663	0	0	719,181		
Test Wk 1 Wed	0	0	715	0	0	416,752		
Test Wk 1 Thu	0	0	1,192	0	0	520,950		
Test Wk 1 Fri	1,013	35,486	693	0	952	600,323	0.975	0.999
Test Wk 2 Mon	0	0	9,296	0	0	520,025		
Test Wk 2 Tue	0	0	960	0	0	559,699		
Test Wk 2 Wed	100	3,566	16,287	0	100	36,259	0.995	0.69
Test Wk 2 Thu	0	0	8,478	0	0	898,028		
Test Wk 2 Fri	0	0	7,879	0	0	625,445		
Totals	3,321	117,565	57,601	0	3,180	7,118,764	0.974	0.992

Table 5. Performance of rule 1205312089 with strict density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.22483334 after four cycles of the cellular automaton. Rule 1205312089 had a fitness of 0.8622 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,693	0	0	1,018	177,954	0.973	1.000
Train Wk 3 Wed	1,000	35,728	0	0	1,001	227,047	0.973	1.000
Train Wk 6 Wed	100	3,594	0	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	2	0	94	632,675	0.974	1.000
Test Wk 1 Mon	0	0	0	0	0	712,669		
Test Wk 1 Tue	0	0	6	0	0	719,181		
Test Wk 1 Wed	0	0	1	0	0	416,752		
Test Wk 1 Thu	0	0	2	0	0	520,950		
Test Wk 1 Fri	1,013	35,475	0	0	963	600,323	0.974	1.000
Test Wk 2 Mon	0	0	4	0	0	520,025		
Test Wk 2 Tue	0	0	4	0	0	559,699		
Test Wk 2 Wed	100	3,566	0	0	100	36,259	0.973	1.000
Test Wk 2 Thu	0	0	1,196	0	0	898,028		
Test Wk 2 Fri	0	0	1	0	0	625,445		
Totals	3,321	117,469	1,216	0	3,276	7,118,764	0.974	1.000

Table 6. Performance of rule 1360891913 with strict density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.25925000 after four cycles of the cellular automaton. Rule 1360891913 had a fitness of 0.8126 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,693	192	0	1,018	177,954	0.973	0.999
Train Wk 3 Wed	1,000	35,728	120	0	1,001	227,047	0.974	0.999
Train Wk 6 Wed	100	3,594	42	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	304	0	94	632,675	0.976	1.000
Test Wk 1 Mon	0	0	209	0	0	712,669		
Test Wk 1 Tue	0	0	143	0	0	719,181		
Test Wk 1 Wed	0	0	28	0	0	416,752		
Test Wk 1 Thu	0	0	246	0	0	520,950		
Test Wk 1 Fri	1,013	35,475	137	0	963	600,323	0.974	1.000
Test Wk 2 Mon	0	0	156	0	0	520,025		
Test Wk 2 Tue	0	0	238	0	0	559,699		
Test Wk 2 Wed	100	3,566	171	0	100	36,259	0.975	0.995
Test Wk 2 Thu	0	0	1,490	0	0	898,028		
Test Wk 2 Fri	0	0	101	0	0	625,445		
Totals	3,321	117,469	3,577	0	3,276	7,118,764	0.974	0.999

Table 7. Performance of rule 1386779481 with strict density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.25208333 after four cycles of the cellular automaton. Rule 1386779481 had a fitness of 0.8745 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,693	0	0	1,018	177,954	0.973	1.000
Train Wk 3 Wed	1,000	35,728	0	0	1,001	227,047	0.973	1.000
Train Wk 6 Wed	100	3,594	0	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	0	0	94	632,675	0.974	1.000
Test Wk 1 Mon	0	0	0	0	0	712,669		
Test Wk 1 Tue	0	0	7	0	0	719,181		
Test Wk 1 Wed	0	0	1	0	0	416,752		
Test Wk 1 Thu	0	0	2	0	0	520,950		
Test Wk 1 Fri	1,013	35,475	0	0	963	600,323	0.974	1.000
Test Wk 2 Mon	0	0	4	0	0	520,025		
Test Wk 2 Tue	0	0	2	0	0	559,699		
Test Wk 2 Wed	100	3,566	0	0	100	36,259	0.973	1.000
Test Wk 2 Thu	0	0	1,391	0	0	898,028		
Test Wk 2 Fri	0	0	1	0	0	625,445		
Totals	3,321	117,469	1,408	0	3,276	7,118,764	0.974	1.000

Table 8. Performance of rule −369784237 with relaxed density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.28314583 after eight cycles of the cellular automaton. Rule −369784237 had a fitness of 0.8530 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,701	0	0	1,010	177,954	0.973	1.000
Train Wk 3 Wed	1,000	35,734	0	0	995	227,047	0.974	1.000
Train Wk 6 Wed	100	3,594	0	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	4	0	94	632,675	0.974	1.000
Test Wk 1 Mon	0	0	0	0	0	712,669		
Test Wk 1 Tue	0	0	12	0	0	719,181		
Test Wk 1 Wed	0	0	1	0	0	416,752		
Test Wk 1 Thu	0	0	3	0	0	520,950		
Test Wk 1 Fri	1,013	35,476	0	0	962	600,323	0.974	1.000
Test Wk 2 Mon	0	0	4	0	0	520,025		
Test Wk 2 Tue	0	0	4	0	0	559,699		
Test Wk 2 Wed	100	3,566	2	0	100	36,259	0.973	1.000
Test Wk 2 Thu	0	0	1,279	0	0	898,028		
Test Wk 2 Fri	0	0	1	0	0	625,445		
Totals	3,321	117,484	1,310	0	3,261	7,118,764	0.974	1.000

Table 9. Performance of rule −1144617577 with relaxed density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.421625003 after four cycles of the cellular automaton. Rule −1144617577 had a fitness of 0.7659 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,702	0	0	1,009	177,954	0.973	1.000
Train Wk 3 Wed	1,000	35,735	2	0	994	227,047	0.974	1.000
Train Wk 6 Wed	100	3,594	0	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	6	0	94	632,675	0.974	1.000
Test Wk 1 Mon	0	0	2	0	0	712,669		
Test Wk 1 Tue	0	0	32	0	0	719,181		
Test Wk 1 Wed	0	0	2	0	0	416,752		
Test Wk 1 Thu	0	0	3	0	0	520,950		
Test Wk 1 Fri	1,013	35,476	1	0	962	600,323	0.974	1.000
Test Wk 2 Mon	0	0	5	0	0	520,025		
Test Wk 2 Tue	0	0	12	0	0	559,699		
Test Wk 2 Wed	100	3,566	47	0	100	36,259	0.974	0.999
Test Wk 2 Thu	0	0	1,450	0	0	898,028		
Test Wk 2 Fri	0	0	31	0	0	625,445		
Totals	3,321	117,486	1,593	0	3,259	7,118,764	0.974	1.000

Table 10. Performance of rule −1023231863 with relaxed density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.5233542 after four cycles of the cellular automaton. Rule −1023231863 had a fitness of 0.8730 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,746	469	0	965	177,954	0.975	0.997
Train Wk 3 Wed	1,000	35,759	559	0	970	227,047	0.975	0.998
Train Wk 6 Wed	100	3,594	1,204	0	100	471,757	0.980	0.997
Train Wk 7 Fri	108	3,415	2,037	0	92	632,675	0.984	0.997
Test Wk 1 Mon	0	0	1,739	0	0	712,669		
Test Wk 1 Tue	0	0	11,530	0	0	719,181		
Test Wk 1 Wed	0	2	1,716	0	0	416,752		
Test Wk 1 Thu	3	0	2,385	0	0	520,950		
Test Wk 1 Fri	1,013	35,488	1,739	0	950	600,323	0.976	0.997
Test Wk 2 Mon	0	0	11,020	0	0	520,025		
Test Wk 2 Tue	0	0	2,061	0	0	559,699		
Test Wk 2 Wed	100	3,566	20,474	0	100	36,259	0.996	0.639
Test Wk 2 Thu	0	0	16,359	0	0	898,028		
Test Wk 2 Fri	0	0	8,963	0	0	625,445		
Totals	3,324	117,570	82,255	0	3,177	7,118,764	0.974	0.989

Table 11. Performance of rule 1205312089 with relaxed density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.290145838 after four cycles of the cellular automaton. Rule 1205312089 had a fitness of 0.8622 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,701	0	0	1,010	177,954	0.973	1.000
Train Wk 3 Wed	1,000	35,734	0	0	995	227,047	0.974	1.000
Train Wk 6 Wed	100	3,594	0	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	4	0	94	632,675	0.974	1.000
Test Wk 1 Mon	0	0	0	0	0	712,669		
Test Wk 1 Tue	0	0	15	0	0	719,181		
Test Wk 1 Wed	0	0	1	0	0	416,752		
Test Wk 1 Thu	0	0	2	0	0	520,950		
Test Wk 1 Fri	1,013	35,476	0	0	962	600,323	0.974	1.000
Test Wk 2 Mon	0	0	4	0	0	520,025		
Test Wk 2 Tue	0	0	4	0	0	559,699		
Test Wk 2 Wed	100	3,566	2	0	100	36,259	0.973	1.000
Test Wk 2 Thu	0	0	1,938	0	0	898,028		
Test Wk 2 Fri	0	0	1	0	0	625,445		
Totals	3,321	117,484	1,971	0	3,261	7,118,764	0.974	1.000

Table 12. Performance of rule 1360891913 with relaxed density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.27354167 after four cycles of the cellular automaton. Rule 1360891913 had a fitness of 0.8126 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,693	297	0	1,018	177,954	0.973	0.998
Train Wk 3 Wed	1,000	35,728	199	0	1,001	227,047	0.974	0.999
Train Wk 6 Wed	100	3,594	94	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	441	0	94	632,675	0.977	0.999
Test Wk 1 Mon	0	0	331	0	0	712,669		
Test Wk 1 Tue	0	0	238	0	0	719,181		
Test Wk 1 Wed	0	0	79	0	0	416,752		
Test Wk 1 Thu	0	0	385	0	963	520,950		
Test Wk 1 Fri	1,013	35,475	237	0	0	600,323	1.000	1.000
Test Wk 2 Mon	0	0	325	0	0	520,025		
Test Wk 2 Tue	0	0	415	0	100	559,699		
Test Wk 2 Wed	100	3,566	297	0	0	36,259	1.000	0.992
Test Wk 2 Thu	0	0	1,656	0	0	898,028		
Test Wk 2 Fri	0	0	254	0	0	625,445		
Totals	3,321	117,469	5,248	0	3,276	7,118,764	0.974	0.999

Table 13. Performance of rule 1386779481 with relaxed density evaluation on back attack packets. A packet was selected as an attack packet if it had a ones density less than or equal to 0.333624998 after four cycles of the cellular automaton. Rule 1386779481 had a fitness of 0.8745 by the fitness function used for this report.

Data Set	Back Attack Initial	Back Attack Cont	False Pos	False Neg Initial	False Neg Cont	True Negative	Sensi-tivity	Speci-ficity
Train Wk 2 Fri	1,000	35,702	0	0	1,009	177,954	0.973	1.000
Train Wk 3 Wed	1,000	35,735	0	0	994	227,047	0.974	1.000
Train Wk 6 Wed	100	3,594	30	0	100	471,757	0.974	1.000
Train Wk 7 Fri	108	3,413	20	0	94	632,675	0.974	1.000
Test Wk 1 Mon	0	0	9	0	0	712,669		
Test Wk 1 Tue	0	0	30	0	0	719,181		
Test Wk 1 Wed	0	0	73	0	0	416,752		
Test Wk 1 Thu	0	0	3	0	0	520,950		
Test Wk 1 Fri	1,013	35,476	0	0	962	600,323	0.974	1.000
Test Wk 2 Mon	0	0	9	0	0	520,025		
Test Wk 2 Tue	0	0	24	0	0	559,699		
Test Wk 2 Wed	100	3,566	40	0	100	36,259	0.974	0.999
Test Wk 2 Thu	0	0	1,556	0	0	898,028		
Test Wk 2 Fri	0	0	5	0	0	625,445		
Totals	3,321	117,486	1,799	0	3,259	7,118,764	0.974	1.000

Table 14. Comparison of strict and relaxed rules. For each rule tested, the table shows the difference in false positive results and false negative results for the strict and relaxed density rules. False negatives are for back attack continuation packets only; there are no false negatives for initial packets for any rule.

	—— False Positives ——			– False Negative Continuation –		
Rule	Strict	Relaxed	Diff	Strict	Relaxed	Diff
-369784237	1,215	1,310	95	3,276	3,261	-15
-1144617577	1,399	1,593	194	3,276	3,259	-17
-1023231863	57,601	82,255	24,654	3,180	3,177	-3
1205312089	1,216	1,971	755	3,276	3,261	-15
1360891913	3,577	5,248	1,671	3,276	3,276	0
1386779481	1,408	1,799	391	3,276	3,259	-17

Table 15. Sensitivity to perturbation. Bits in two test packets were complemented randomly with replacement. The table shows the percentage of bits that could be modified before the density of the cellular automaton exceeded the upper density bound. Testing was with the relaxed density rule. Testing stopped at 0.25 (25%) of bits modified.

Rule	Perturbation Tolerated
-369784237	0.25
-1144617577	0.25
-1023231863	0.04
1205312089	0.25
1360891913	0.04
1386779481	0.25

Table 16. Rules tested with their density and fitness values. “Cycles” is the number of cellular automaton cycles used in the tests. Packets were considered attack packets if their ones densities were less than or equal to the values given.

Rule	Cycles	Fitness	Strict Density	Relaxed Density
-369784237	8	0.8530	0.2275833	0.28315
-1144617577	4	0.7659	0.3494167	0.42163
-1023231863	4	0.8731	0.5104167	0.52335
1205312089	4	0.8622	0.2248333	0.29015
1360891913	4	0.8127	0.25925	0.27354
1386779481	4	0.8746	0.2520833	0.33362

Table 17. Experimental data collected.

Actual IMAP attack packets (detected by string comparison)
Actual back attack initial packets (detected by string comparison)
Actual back attack continuation packets (detected by string comparison)
IMAP attack packets detected by cellular automaton
Back attack initial packets detected by cellular automaton
Back attack continuation packets detected by cellular automaton
False positive packets
False negative IMAP attack packets
False negatice back attack initial packets
False negative back attack contiuation packets
Total packets read
IP packets read
TCP packets read and processed by cellular automaton

Appendix B

Figures

Figure 1. Operation of cellular automaton rule 1205312089 with the first 96 bits of data from packet 1,819 of the Friday week two training data. The initial data and four cycles of the cellular automaton cell values are shown as five rows in the figure. The ones density does not decrease below the density threshold for this rule. This packet would not be selected as a match.

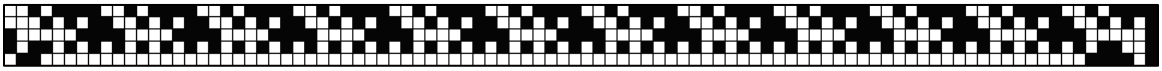

Figure 2. Operation of cellular automaton rule 1205312089 with 96 bits 0x2f characters, representing a back attack packet. The initial data and four cycles of the cellular automaton cell values are shown as five rows in the figure. A pattern of four zero bits and four one bits emerges in cycle two. In cycle three, the bits alternate between zero and one except near the boundaries. In cycle four, nearly all bits are zero. The ones density is 0.073, indicating that this packet matches a pattern in the set.

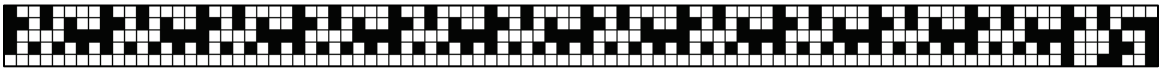

Figure 3. Operation of cellular automaton rule 1205312089 with 96 bits 0x90 characters, representing an IMAP attack packet. The initial data and four cycles of the cellular automaton cell values are shown as five rows in the figure. Cycles two, three, and four exhibit the same patterns seen in Figure 2. At the end of cycle four, the ones density is 0.042, indicating that this packet matches a pattern in the set.

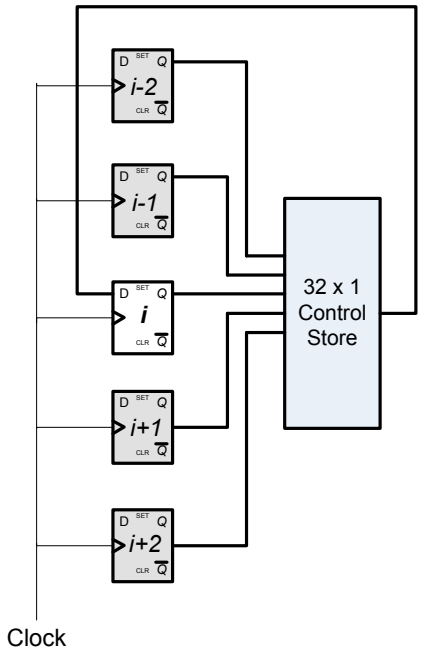

Figure 4. A conceptual diagram showing how the next state of a cellular automaton can be computed using the table look-up mechanism with a control store.

The next state of cell *i* is to be computed. The computation relies on the current states of cells *i*–2, *i*–1, *i, i*+1, and *i*+2. The contents of those five cells form a five bit address into the control store.

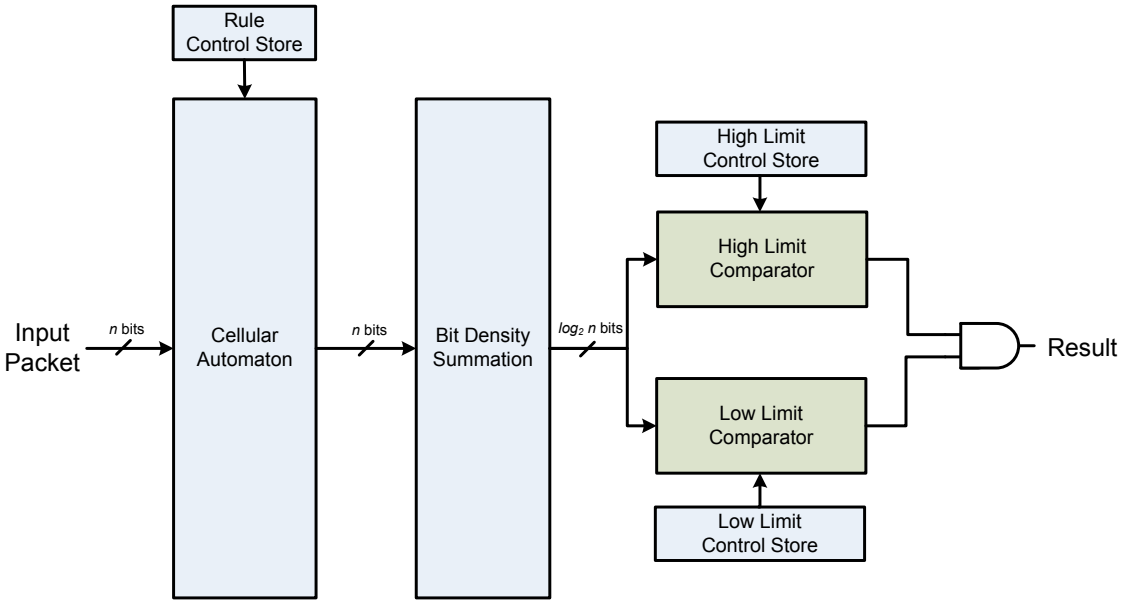

Figure 5. A conceptual diagram showing the implementation in hardware of a pattern recognizer based on the use of cellular automata, with accepting state defined by bit density bands.

References

Alon, N., & Bruck, J. (1994). Explicit Constructions of Depth-2 Majority Circuits for Comparison and Addition. *SIAM J. Discret. Math., 7*(1), 1-8.

Baskerville, R., Pries-Heje, J., & Venable, J. (2009). Soft design science methodology. *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*. New York: ACM. (pp. 1-11)

Berghel, H. (2003). Malware month. *Communications of the ACM, 46*(12), 15-19.

Bishop, M. (2003). *Computer Security: Art and Science*. Boston: Addison-Wesley Professional.

Blelloch, G. E. (1993). Prefix sums and their applications. In J. H. Reif (Ed.), *Synthesis of Parallel Algorithms* (pp. 35-60). San Mateo, CA: Morgan-Kauffman Publishers.

Brewer, G. (2008). *Spiking Cellular Associative Neural Networks for Pattern Recognition* (Unpublished doctoral dissertation.) University of York, Uork, U.K.

Brooks, R., Orr, N., Zachary, J., & Griffin, C. (2002). An interacting automata model for network protection. *Fifth International Conference on Information Fusion.* Annapolis, MD: IEEE (pp. 1090-1097).

Cannady, J. (1998). Artificial neural networks for misuse detection. *21st National Information Systems Security Conference.* Gaithersburg, MD: National Institute of Standards and Technology / National Computer Security Center. (pp. 368-381).

Chady, M., & Poli, R. (1997). Evolution of cellular-automaton-based associative memories. *Cognitive Science Research Papers.* University of Birmingham, U.K.

Chaudhuri, P., Chowdhury, D., & Nandi, S. (1997). *Additive cellular automata: theory and applications*: Piscataway, NJ: Wiley-IEEE Computer Society Press.

Chen, T. M. (2010). Stuxnet, the real start of cyber warfare? [Editor's Note]. *Network, IEEE, 24*(6), 2-3.

Chowdhury, D., Gupta, I., & Chaudhuri, P. (2002). A low-cost high-capacity associative memory design using cellular automata. *Computers, IEEE Transactions on, 44*(10), 1260-1264.

Chua, L. O., & Yang, L. (1988). Cellular neural networks: applications. *IEEE Transactions on Circuits and Systems,35*(10), 1273-1290.

Clark, C., Lee, W., Schimmel, D., Contis, D., Koné, M., & Thomas, A. (2004). A hardware platform for network intrusion detection and prevention. *Proceedings of Third Workshop on Network Processors & Applications (NP3),* St. Louis, MO: Washington University in St. Louis. (pp. 68-74).

Cook, M. (2004). Universality in elementary cellular automata. *Complex Systems, 15*(1), 1-40.

Crowcroft, J., Hand, S., Mortier, R., Roscoe, T., & Warfield, A. (2003). QoS's downfall: at the bottom, or not at all! *Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS: What have we learned, why do we care?*, New York: ACM. (pp. 109-114).

Crutchfield, J. P., Mitchell, M., & Das, R. (1998). The evolutionary design of collective computation in cellular automata. In J. Crutchfield & P. Schuster (Eds.), *Evolutionary Dynamics: Exploring the Interplay of Selection, Accident, Neutrality, and Function* (pp. 361–411). New York: Oxford University Press.

Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys, 44*(2), 1-42.

Erdogan, O., & Cao, P. (2007). Hash-AV: fast virus signature scanning by cache-resident filters. *International Journal of Security and Networks, 2*(1), 50-59.

Florêncio, D., & Herley, C. (2011). Sex, Lies and Cyber-crime Surveys. *Proceedings of the The Tenth Workshop on Economics of Information Security.* Fairfax, VA: George Mason University.

Forrest, S., Hofmeyr, S. A., & Somayaji, A. (1997). Computer immunology. *Communications of the ACM, 40*(10), 88-96.

Fuentes, F., & Kar, D. C. (2005). Ethereal vs. Tcpdump: a comparative study on packet sniffing tools for educational purpose. *J. Comput. Small Coll., 20*(4), 169-176.

Ganguly, N., Maji, P., Das, A., Sikdar, B., & Chaudhuri, P. (2002). Characterization of Non-Linear Cellular Automata Model for Pattern Recognition. *Advances in Soft Computing—AFSS 2002*. Berlin/Heidelberg: Springer-Verlag. (pp 141-150)

Ganguly, N., Maji, P., Sikdar, B., & Chaudhuri, P. (2004). Design and characterization of cellular automata based associative memory for pattern recognition. *Systems, Man, and Cybernetics, Part B: IEEE Transactions on Cybernetics, 34*(1), 672-678.

Haines, J. W., Rossey, L. M., Lippmann, R. P., & Cunningham, R. K. (2001). Extending the DARPA off-line intrusion detection evaluations. *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX) 2000.* Los Alamitos, CA: IEEE Computer Society Press. (Vol. 1, pp. 35-45)

Hald, S. L. N., & Pedersen, J. M. (2012). An updated taxonomy for characterizing hackers according to their threat properties. *Fourteenth International Conference on Advanced Communication Technology (ICACT)*. Piscataway, NJ: IEEE. (Vol. 2, pp. 81-86)

Hall, M. J. (2013). *Improving Software Remodularisation.* (Unpublished doctoral dissertation.) University of Sheffeld, Sheffield, U.K.

Jaeschke, R., Guyatt, G. H., & Scackett, D. L. (2006). Determining the Presence or Absence of Disease: Reporting the Performance Characteristics of Diagnostic Tests. In T. A. Lang & M. Secic (Eds.), *How to report statistics in medicine annotated guidelines for authors, editors, and reviewers* (pp. 125-158). Philadelphia: American College of Physicians.

Katz, R. H. (1994). *Contemporary logic design*. Redwood City, California: Benjamin/Cummings.

Kauffmann, C., & Piche, N. (2008). Cellular automaton for ultra-fast watershed transform on GPU. *Proceedings of the19th International Conference on Pattern Recognition (ICPR)* Piscataway, NJ: IEEE (pp. 1-4)

Kundu, A., & Roy, D. (2010). An efficient approach to Web page classification using non-linear cellular automata. *Proceedings of the First International Conference on Parallel Distributed and Grid Computing (PDGC)* Piscataway, NJ: IEEE. (pp. 313-318)

Latif, A., Dalhoum, A., & Al-Dhamari, I. (2010). fMRI brain data classification using cellular automata. In Mastorakis, N., Mladenov, V., & Bojkovic, Z. (Eds.) *New Aspects of Applied Informatics, Biomedical Electronics &Informatics and Communications, 10th WSEAS International Conference on Applied Informatics and Communications,* Taipei, Taiwan: WSEAS (pp. 348-352).

Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., … Zissman, M. A. (2000). Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX) 2000.* Los Alamitos, CA: IEEE Computer Society Press. (Vol. 2, pp. 12-26)

Löf, A., & Nelson, R. (2010). Comparing Anomaly Detection Methods in Computer Networks. *Proceedings of the Fifth International Conference on Internet Monitoring and Protection (ICIMP)* Red Hook, NY: Curran Associates (pp. 7-10).

Mahoney, M. V., & Chan, P. K. (2003). An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In G. Vigna, C. Kruegel & E. Jonsson (Eds.), *Recent Advances in Intrusion Detection Vol. 2820.* Berlin / Heidelberg: Springer (pp. 220-237).

Maji, P., Ganguly, N., & Chaudhuri, P. (2003). Error correcting capability of cellular automata based associative memory. *Systems, Man and Cybernetics, Part A: IEEE Transactions on Systems and Humans, 33*(4), 466-480.

Mano, M. M., & Kime, C. (2007). *Logic and Computer Design Fundamentals (4th Edition)*. Boston: Prentice Hall.

McCumber, J. R. (1991). Information systems security: a comprehensive model. *Fourteenth NIST-NCSC National Computer Security Conference.* Gaithersburg, MD: National Institute of Standards and Technology / National Computer Security Center. (pp. 328-337)

McHugh, J. (2000). Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information Systems Security, 3*(4), 262-294.

Mitchell, M., Crutchfield, J., & Das, R. (1996). Evolving cellular automata with genetic algorithms: A review of recent work. In Goodman, E. D. (Ed.), *Proceedings of the First International Conference on Evolutionary Computation and its Applications (EvCA'96)* Moscow: Presidium of the Russian Academy of Sciences (pp. 1-14).

Mitchell, M., Crutchfield, J., & Hraber, P. (1994). Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D: Nonlinear Phenomena, 75*(1-3), 361-391.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys, 33*(1), 31-88.

Otey, M., Parthasarathy, S., Ghoting, A., Li, G., Narravula, S., & Panda, D. (2003). Towards NIC-based intrusion detection. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining,* Washington, DC (pp. 723-728).

Pfleeger, C. P., & Pfleeger, S. L. (2006). *Security in Computing Fourth Edition*. Upper Saddle River, NJ: Prentice Hall Professional.

Porter, R., & Bergmann, N. (1999). Evolving FPGA based cellular automata. *Simulated Evolution and Learning*, 114-121.

Reeves, C. R., & Rowe, J. E. (2003). *Genetic Algorithms: Principles and Perspectives A Guide to GA Theory*. Boston: Kluwer Academic Publishers.

Rhodes, B., Mahaffey, J., & Cannady, J. (2000). Multiple self-organizing maps for intrusion detection. *Proceedings of the 23rd National Information Systems Security Conference,* Gaithersburg, MD: National Institute of Standards and Technology / National Computer Security Center. (pp. 16-19).

Saha, S., Maji, P., Ganguly, N., Sikdar, B. K., & Chaudhuri, P. P. (2002, 6-9 Oct. 2002). Evolving cellular automata model for pattern recognition and classification. In El Kamel, A., Mellouli, K. & Borne, Pierre (Eds.) *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics,* New York: IEEE (pp. 114-119).

Salmela, L., Tarhio, J., & Kytöjoki, J. (2007). Multipattern string matching with q-grams. *Journal of Experimental Algorithmics, 11*, 1-19.

Sanders, J., & Kandrot, E. (2011). *CUDA by example; an introduction to general-purpose GPU programming*. Boston: Addison-Wesley.

Sarkar, P. (2000). A brief history of cellular automata. *ACM Computing Surveys, 32*(1), 80-107.

Sekar, R., Guang, Y., Verma, S., & Shanbhag, T. (1999). A high-performance network intrusion detection system. *Proceedings of the 6th ACM conference on Computer and Communications Security*, New York: ACM (pp. 8-17).

Shonkwiler, R. (1993). Parallel genetic algorithms. *Proceedings of the 5th International Conference on Genetic Algorithms,* San Francisco: Morgan Kaufmann Publishers (pp. 199-205)

Singaraju, J., Bu, L., & Chandy, J. (2005). A signature match processor architecture for network intrusion detection. In Arnold, J. & Pocek, K. (Eds.) *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, CA: IEEE (pp. 235-242).

Smith, A. (1971). Two-dimensional formal languages and pattern recognition by cellular automata. *Conference Record of the 12th Annual Symposium on Switching and Automata Theory,* East Lansing, MI: IEEE (pp. 144-152).

Solomon, M. G., & Chapple, M. (2005). *Information Security Illuminated*. Sudbury, MA: Jones and Bartlett Publishers.

Sommerhalder, R., & Westrhenen, S. C. (1983). Parallel Language Recognition in Constant Time by Cellular Automata. *Acta Informatica, 19*(4), 397-407.

Stallings, W., & Brown, L. (2008). *Computer Security Principles and Practice*. Upper Saddle River, NJ: Pearson Prentice Hall.

Stallings, W., & Brown, L. (2012). *Computer Security Principles and Practice Second Edition*. Upper Saddle River NJ: Pearson Prentice Hall.

Sutton, W. F., & Linn, E. (1976). *Where the Money Was, The Memoirs of a Bank Robber* (2004 ed.) New York: Broadway Books.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications,* Ottawa, ON, Canada: IEEE (pp. 1-6).

Tavallaee, M., Stakhanova, N., & Ghorbani, A. A. (2010). Toward Credible Evaluation of Anomaly-Based Intrusion-Detection Methods. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 40*(5), 516-524.

Vinod, P., Jaipur, R., Laxmi, V., & Gaur, M. (2009). Survey on Malware Detection Methods. *Proceedings of the Proceedings of the 3rd Hackers' Workshop on Computer and Internet Security,* Kanpur, UP, India: Prabhu Goel Research Centre for Computer & Internet Security (pp. 74-79).

Wolfram, S. (1994). *Cellular Automata and Complexity: Collected Papers*. Reading, MA: Addison-Wesley.

Wolfram, S. (2002). *A New Kind of Science*. Champaign, IL: Wolfram Media.

Wood, D. (1987). *Theory of Computation*. New York: John Wiley and Sons.

Yu, F., Katz, R. H., & Lakshman, T. V. (2004, 5-8 Oct. 2004). Gigabit rate packet pattern-matching using TCAM. *Proceedings of the 12th IEEE International Conference on Network Protocols,* Piscataway, NJ: IEEE (pp. 174-183).

Zhang, J., Duan, H., Wang, L., Guan, Y., & Wu, J. (2008). A Fast Method of Signature Generation for Polymorphic Worms. In Xie, Y., Li, W., & Zhou, J. (Eds.), *Proceedings of the 2008 International Conference on Computer and Electrical Engineering,* Piscataway, NJ: IEEE (pp. 8-13).

Zhang, Q., Reeves, D. S., Ning, P., & Iyer, S. P. (2007). Analyzing network traffic to detect self-decrypting exploit code. In Deng, R. & Samarati, P. (Eds.), *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security,* New York: ACM (pp. 4-12).

Zhuge, J., Holz, T., Song, C., Guo, J., Han, X., & Zou, W. (2009). Studying malicious websites and the underground economy on the Chinese web. In Johnson, M. E. (Ed.) *Managing Information Risk and the Economics of Security* (pp. 225-244) New York: Springer U.S.

Zu, Y., Yang, M., Xu, Z., Wang, L., Tian, X., Peng, K., & Dong, Q. (2012). GPU-based NFA implementation for memory efficient high speed regular expression matching. *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming.* New York: ACM (pp. 129-140).