Journal of Information Assurance and Security 4 (2009) 247-255

Using Qualia and Hierarchical Models in Malware Detection

Bobby D. Birrer [1], Richard A. Raines, Rusty O. Baldwin, Mark E. Oxley, and Steven K. Rogers[2]

[1]Air Force Institute of Technology,
2950 Hobson Way, Bldg 642, WPAFB, OH, USA 45433-7765
john.doe@afit.edu

[2] Air Force Research Laboratory,
2241 Avionics Cir., WPAFB, OH, USA 45433-7765
john.doe@wpafb.af.mil

Abstract**:** Detecting network intruders and malicious software is a significant problem for network administrators and security experts. New threats are emerging at an increasing rate, and current signature and statistics-based techniques are failing to keep pace. Intelligent systems that can adapt to new threats are needed to mitigate these new strains of malware as they are released.

This research discusses a proposed system that uses hierarchical models, contextual relationships, and information across different layers of abstraction to detect malware based on its qualia, or essence. By looking for the underlying concepts that make a piece of software malicious, this system avoids the pitfalls of static solutions that focus on predefined signatures or anomaly thresholds. If successful, this type of qualia-based system would provide a robust detection system that would be able to more readily identify novel attacks with less human interaction and supervision than current techniques.

Keywords: malware detection, anti-virus, latent dirichlet allocation, qualia

Introduction

Current malware detectors are in danger of losing their ongoing struggle with virus authors. In a week-long study by Symantec in November 2007, over 60% of programs being downloaded were characterized as malware, showing that malware was being downloaded at a faster rate than legitimate programs [1]. This prevalence of malware is a serious problem as current commercial techniques to detect and identify malware are reactive processes. Signature-based detection, the most common form of protection, requires a human analyst to generate a unique signature for each new attack released. With the increased number of viruses and worms being released daily, it is becoming more difficult to generate signatures quickly enough to keep up with new attacks. This problem is being further compounded as metamorphic viruses that are able to automatically change themselves to avoid detection become more common.

As this trend continues, it appears unlikely that anti-virus companies will be able to keep signatures current to protect against all attacks, leaving systems vulnerable to exploitation. A potential method to counter the rapid development and appearance of new malware is through the use of intelligent, adaptive detection systems that are not reliant on static signatures.

Most of the current work into adaptive systems has focused primarily on anomaly detection systems which classify programs based on their "distance" from either predetermined malicious or benign program training samples. The classification system adapts by adjusting the decision boundary used to classify programs. While this technique allows for the detection of some novel attacks, it is unable to properly classify malicious programs that are similar to benign programs and can have false positives if legitimate programs deviate widely from the observed norm.

This research argues that as signature-based detection falls behind, anomaly detection systems are unable to provide reliable protection without unreasonable false positive rates. These models are inherently unable to make intelligent classification decisions because they do not address the fundamental problem of malware detection. Current implementations do not consider concepts such as context, intent, and end behavior. They simply look for symptoms or signatures of the malware rather than the relationships and behaviors that actually make a specific piece of code malicious. Fingerprints and DNA can be used to indentify criminals who have been caught in the past, but are useless in predicting future offenders. Likewise current malware detection techniques are unable to predict and identify future malware.

The goal of this research is to develop a new, robust malware detection technique that addresses these shortcomings by looking beyond static signatures and decision boundaries and instead examine the malware and the target system as a whole and identify malware based on the fundamental factors that make a piece of software malicious. This research draws upon tenets from Qualia Exploitation of Sensor Technology (QUEST), which attempts to gain an engineering advantage over current pattern recognition and object classification techniques through the use of qualia-based systems [2]. According to QUEST, qualia are the mental representations that "intelligent" systems use to process data from the sensed environment into a useful world model that can be used to make decisions. These representations are not simply physical measurements, but are rather an abstraction of those measurements into meaningful information that takes into consideration history, current state, and context.

Received June 10, 2009

This paper outlines the current state of malware detection and describes how a successful QUEST-based solution would provide an engineering advantage over these techniques. Further, the paper describes an initial hierarchal model and mathematical models that can be used to automatically train the representation without assistance from human analysts. The paper then closes with ideas for future work and how the researchers plan to further develop the model into a usable system.

Background

2.1. Current Malware Detection Techniques

Because malware is such a significant problem, multiple techniques have been developed to help identify both known and novel malicious software. This section describes some of the most common malware detection strategies currently being investigated and their limitations.

2.1.1. Signatures

Signature-based detection is the most common form of malware detection and works by taking a known malware sample and looking for specific byte sequences unique to that strain of malware. Future programs are then searched for that byte-sequence or signature. If the byte-sequence is not found in the program, then the program is considered to be benign. However, an absence of known byte-sequences is not sufficient to guarantee a program is malware-free. New malware strains must be manually identified by a human analyst who can then develop a signature. However, until a signature is developed and distributed to vulnerable systems, the malware can propagate undetected. Further, many malware strains are capable of transforming and changing themselves, making it extremely difficult to develop signatures that work for all the transformations. Signature-based systems are commonly used across the networking environment and include anti-virus systems such as those marketed by Symantec.

2.1.2. Normalized Signatures

One focus of current research is to create normalized signatures that can handle metamorphic malware that modifies itself to avoid detection. This research [3-5] demonstrates malware detection systems that can perform basic code transformations to programs and reduce them to a normalized form, limiting the need for multiple signatures for variants of the same strain of malware. While these systems are effective in mitigating some metamorphic viruses, they still rely on having seen the malware previously and having the appropriate signature.

2.1.3. Semantic Signatures

A similar technique to normalized signatures is the use of signatures that are based on program semantics. [6] describes a technique for identifying viruses based on the semantic meaning of code sections rather than specific syntax and byte sequences. Semantic templates are formed by abstracting out symbolic information such as register names and memory addresses which can be changed through obfuscation or the addition/removal of functionality from the malware. However, these templates are created manually for each strain of malware and their usefulness is limited versus instruction replacement obfuscations.

2.1.4. Control Flow Graphs

A different method being researched is the use of control flow graphs to fingerprint and identify malicious software. These methods [7, 8] describe a technique in which the authors generate control flow graphs for executable code traveling through a network and compare them with graphs of known malicious software. The graphs consist of nodes representing basic blocks, with edges to represent control structures such as branches and jumps. Graph coloring is applied to each node based on the general type of instructions located in the basic block to provide additional granularity without over-constraining the fingerprints. These graphs are then broken into k-subgraphs for comparison to known worms and other traffic. If similar subgraphs are visible in a large number of network flows, it could be seen as an indication of a worm outbreak.

2.1.5. Application Program Interface (API) and system call sequences

While control flow graphs can provide fingerprints of the structure and function of a program, they are a very low-level representation and can be fooled by various instruction-level changes. To counter this weakness, [9-12] attempt to characterize programs based on system call and API sequences. APIs and system calls are the interface between processes and the operating system. Programs must use these interfaces to access or modify any other part of the system. Therefore, monitoring the system calls allows a detection system to observe what the process is doing without having to follow the assembly-level implementation of the program. Many of these techniques take an anomaly detection approach to determine whether a program or system has been compromised and issue an alert if the system call sequences of a running program suddenly change or begin to resemble a known malicious program.

2.1.6. Byte/Opcode Distributions

Researchers are also examining whether inherent byte distributions and opcode frequencies can be used to identify malicious software. [13] examines K-nearest neighbors (KNN), Naïve Bayes, Support Vector Machines, and other methods to classify the executables based on the n-gram byte sequences, achieving a success rate of over 99% for several methods. [14] uses a disassembler to find opcodes within programs, as the opcodes specify what actions to perform, providing a way of determining the functionality of the program. However, their research only focuses on the frequency of the opcodes and not their ordering or relationships with one another.

[15] uses sequences of opcodes to classify programs. The authors state that sequences provide additional context not available with opcode distributions alone. The instruction arguments are ignored in the analysis to provide a more generalized feature set. The authors applied eight classifiers to feature sets consisting of sequences between one and six opcodes long. The technique succeeded in correctly classifying up to 99% of programs with Boosted Decision Tree and Boosted Naive Bayes classifiers. [15]

[16] uses similar features, but it uses relationships between instructions within basic blocks rather than strict instruction

sequences. This allows the technique to handle instruction reordering and/or insertion of "garbage" instructions, two common techniques malware writers use to avoid signature detectors [17]. The authors report that the system correctly classified 93% of the test programs [16].

2.1.7. Limitations

While the techniques described above use a wide variety of measurements and classification algorithms, they can be broken into two main categories: signature detection and anomaly detection. Signature detection relies on specifying a priori what attributes or features indicate an attack. While classical signature detection is the most common form, normalized signatures, semantics signatures, misuse detection, and control graph matching are all forms of signature detection. They rely on having previously seen the exact or similar malware in the past, leaving them prone to false negatives when faced with novel attacks.

Anomaly detection methods do not require a priori knowledge of attacks for detection, potentially leading to fewer false negatives when faced with novel malware. However, anomaly detection methods assume that malware will behave significantly different than normal system operation, which is not always true. Malware writers can craft their attacks to mimic normal actions or pace their execution to avoid crossing the detection threshold. A truly skilled attacker can even slowly escalate their actions in an attempt to expand the system's definition of "normal behavior," allowing attacks that would have been caught initially to go undetected.

This research suggests the decision boundary model fails primarily because malicious and benign programs can perform the same basic actions. At the assembly level, they use the same instruction set to manipulate data, perform calculations, and execute behavior. Both types of programs use the same set of API and system calls to interact with the operating system and access resources. Even the high-level functionality can be similar. For example Microsoft Windows Update connects to a remote site, downloads code, and then makes modifications to the operating system—the same behavior that a Trojan installing a rootkit would exhibit [18].

This research argues that it is not individual or even sequences of behaviors alone that makes a program malicious. It is the combination of behavior with the program's end effects and the intent or purpose of the malware author that causes it to be considered malware. However, current decision boundary models cannot capture this type of context, and these model limitations make the decision boundary model inadequate for performing an accurate classification. [9, 11, 12, 19]

2.2. QUEST

One initiative to overcome the shortfalls of current identification and recognition techniques is QUEST [2]. The primary goal of QUEST is to gain an engineering advantage over current classification and prediction techniques through the use of qualia (singular quale). According to QUEST, qualia are the subjective mental representations that are used to model information about the environment in a meaningful way that can be exploited. Qualia are inherently subjective and relative, depending on not just physical measurements, but history, current state, and context. It is possible that presenting the same measurements at different times can evoke different sets of qualia. For each set of input data, multiple qualia compete to determine which forms the best plausible narrative and should be evoked.

QUEST lists a set of tenets which describe an intelligent system that can process and use a qualia representation to provide an engineering advantage in classification, prediction, and decision-making. Any qualia-based system will adhere to the following tenets and use them as part of the design process [2]:

1. Qualia
2. Physiologically Motivated Information Processing (PMIP)
3. Learning
4. Architecture
5. Theory
6. Driver Problems

Malware detection is one of the driver problems for QUEST because malware is a concept that is hard to define in objective terms. Because it is a subjective concept, it is difficult to capture and characterize with standard techniques. Malware is extremely context dependent, with different sources defining it differently. Even anti-virus companies cannot agree on a definite taxonomy. This is further complicated by the fact that one user can use a program on a system legitimately, while another user can use the exact same program on the exact same system for malicious purposes. If one was to ask a system administrator or an anti-virus analyst for a definition of what they looked for in a malicious program, it is unlikely they would reply with a more concrete answer than "I know it when I see it."

QUEST wants to create a computer system capable of making that type of judgment call based on examining the entire situation and current context. Such a system would provide a distinct engineering advantage in the malware domain as it would reduce the need for human generated signatures, allowing it to keep up with the release of new, novel malware.

2.3. Latent Dirichlet Allocation (LDA)

One of the major goals of QUEST is to develop a system that can learn and adapt with minimal human input. In order to achieve this goal, the system must be able to automatically adjust and train a world model and reasoning kernel. A mathematical model being explored as a method of training the system is the LDA model described in [20]. LDA is a generative model that explains sets of observations through unobserved groupings. It is commonly used in topic modeling of documents, where observed words are explained through mixtures of topics in a document. In the model, each document is assumed to have a Dirichlet distribution of topics which drives the distribution of words for that document. The authors provide algorithms for both inference and parameter estimation and applied their techniques to an 8,000 document corpus with a vocabulary of 16,000 words. The LDA model successfully clustered words from the documents into meaningful topics that could then be used as features for document classification. [21] applied LDA to video sequences of busy metropolitan scenes and clustered similar movements such as cars driving down a street and pedestrians using crosswalks. Further, they clustered different co-occurring groups of similar movement to form interactions such as traffic blocking crosswalks. The system was able to detect abnormal behavior such as jaywalking.

Design

3.1. Design Goals

The goal of this research is to design a qualia-based system that can make intelligent decisions that can be applied to the realm of malware detection. After considering the QUEST tenets, this research argues that the system should be designed with the following capabilities:

1. Predicts future behavior and events based on past behavior and events
2. Identifies/classifies objects based on relationships and behavior
3. Weighs multiple predictions against each other
4. Uses information from multiple abstraction levels
5. Incorporates provided knowledge
6. Learns from experience
7. Can make decisions based on incomplete data or assumptions
8. Reconciles observed behavior with predicted behavior
9. Reevaluates prior decisions using current knowledge

3.2. Basic Architecture

From the QUEST architecture tenet, the system consists of a set of sensors, a world model, and the kernel that operates on the model. The sensors provide data that the system reads, converts, and stores in the model. The kernel examines the world model and makes decisions and predictions. This decision-making process is constantly adapting and adjusting as the system experiences additional situations and the kernel must interpret the changes to the world model. The first stage of the research focuses on the model and the kernel as they are the components most vital to making decisions. Once they are shown to be effective, the research will shift to converting the sensor data to the world model format.

3.3. Initial Model Design

The model stores the system's knowledge and interpretation of the environment. It must not simply store reams of physical measurements, but instead represent information through the use of relative, subjective qualia. These qualia will be unique to that system and will change as it incorporates new information that changes its perception of the world. Further, it must store these qualia in a way that allows the kernel to efficiently access and interpret them to make predictions about future events.

Applying the QUEST tenets [2] and [22], the proposed model is made up of entities and the relationships between them. Because QUEST stresses the importance of context and relationships between entities, a reasonable way of modeling and interpreting the world is through the use of graphs and graph theory. Entities are represented graphically as nodes while the relationships are represented as directional linkages or edges between nodes. Graph theory provides standard terminology and algorithms which can potentially be leveraged for this application. Further, a large number of tools for working with graphs exist, which will be useful when developing a prototype of the system.

Nodes or vertices represent individual instantiations of entities observed in the environment and are labeled with unique identifiers such as “Bob”, “Lamp 1”, or “Attack 2”. Each node will have certain attributes associated with it such as height, length, and weight. These attributes could be used with standard pattern recognition techniques to provide additional input, or can be used in the determination of relationships between objects such as “smaller than”.

Additionally, each node has a “color”, or category/class, that corresponds to a basic category or concept such as “person”, “small, movable object”, or “network event”. An entity can belong to a large number of categories or subcategories, meaning each node could have multiple “colors”. Being able to model the categories or concepts that an entity falls under is important as objects in different classes often behave differently or have different relationships with other entities. These differences can be used to either predict future behavior based on an entity's class and relationships, or can be used to classify unknown entities based on their behavior and relationships.

Linkages or edges between nodes represent relationships between the objects such as “on top of”, “inside”, “moves”, “smaller than”, or “before”. These directional linkages represent how entities are tied together in the environment spatially, temporally, and functionally. Simple relationships such as “Object 1 is on-top-of Object 2” can be represented with a one-to-one edge. However, more complex relationships such as “Puppet 1 hides Object1 from Puppet 2 inside Object 2” require the use of hyperedges that are capable of connecting multiple vertices. Each edge has an associated label specifying what relationship it represents, and can have a weight that indicates the strength of the relationship or other information.

3.4. Initial Kernel Design

While the model stores the information about the environment, the kernel is responsible for processing and interpreting the model, making decisions and predictions regarding future states and behavior. The kernel must use both learned and provided knowledge to examine not only the current state of the model, but also past states. It should use context and look at the relationships of all the represented objects and not focus on isolated pieces of data.

The initial system design uses a graph of vertices and edges to provide a snapshot of its perception of the world at an instant of time. As time passes, links are added, removed, or modified based on changes in the world. The system can store each of these snapshots to allow for playback of events or to revisit past decisions.

The changes in edges and vertices form what this research refers to as patterns. These patterns can either be learned or provided a priori. The simplest patterns consist of before and after states, showing the state of a set of nodes and links at a specified time along with the new state after a set transition time. Conceptually, this type of pattern states "if in State 1, expect State 2 next." Such a pattern is no different than a simple "if-then" statement, but it provides a basic foundation for more advanced and robust patterns. Further, these patterns can be provided as a form of knowledge or learned over time as a form of experience.

Patterns can be given an associated prediction accuracy, which states how often the predicted state follows the observed "before" state. For example, a pattern that predicts the impact location of a free-falling object would likely have high prediction accuracy. Likewise, predicting which direction a car is going to turn at a four-way intersection would probably have a lower accuracy. Associating prediction accuracy with each pattern allows the system to assign a level of certainty to its predictions and provides a metric to weigh competing predictions.

Modifying the prediction accuracy of patterns can be used to implement specification and generalization in the system. This research describes generalization as taking a number of distinct observations and using them to create a pattern that can be applied to a variety of situations even if the measurements do not match any of the original observations. A general pattern attempts to find the commonalities between distinct, but similar patterns. Specification is the reverse of generalization as it adds extra information to a general pattern in hopes of providing better prediction. It examines and isolates the differences between patterns and uses those distinctions to make different predictions. Generalization allows the system to predict behavior in a larger number of circumstances, while specification allows the system to improve prediction by incorporating additional information.

3.5. Hierarchical Design

One of the intended contributions of this research is to show that using information from multiple layers of abstraction will provide an engineering advantage over signatures from a single layer. The proposed model will provide functionality to model multiple layers, with information from each layer being passed both up and down, as well as within a specific layer in the hierarchy. Entities, relationships, and patterns from lower layers can be combined to form higher-level patterns that can represent more complex or meaningful concepts.

Patterns of behavior at lower layers can be passed up to other layers as relationship links. For example, if a set of behaviors matches a pattern for “edits a file”, then an “edits” link could be evoked at the next level. This allows the upper level to incorporate the fact that a file has been edited into its own patterns and decision making, while abstracting away the specific implementation details. Abstracting away low-level information allows higher levels to focus only on the information that is useful to them, reducing the complexity of patterns and providing a more robust, qualia-like solution. Additionally, this information from lower levels can be combined with other evoked links also passed up from lower layers or measurements and data from the current layer. This allows the creation of complex and compound concepts and patterns of behavior.

Having multiple layers allows the system to make decisions at whichever layer of abstraction is the most useful for a given situation. It also allows the system to correlate observations across the different levels. If data from the lower level does not correspond with observations and predictions made at a higher level, it could indicate a measurement error or an incorrect prediction. This could be especially useful in the malware domain, as many strains of malware perform low-level actions in an attempt to hide themselves from higher-level detection processes.

Further, examining data from multiple layers of abstraction provides a more complete view of what is happening in the environment or system. Data that is meaningless at one layer might be extremely useful when combined with information from another layer. For example, a program generating a large number of outgoing connections is not particularly meaningful until it is combined with a large number of similar connections being observed across a network. Combining these two pieces of information could be used as an indication of a worm propagating over the network, a conclusion that could not be drawn by either piece of data by itself.

In addition to the bottom-up benefits described above, using a hierarchical approach also allows a top-down feedback loop to be added to the model. Decisions made at higher levels of abstraction can be used to drive low-level sensors in order to gain specific information that could potentially help classification. For example, if a system administrator watching network traffic suspects a worm is propagating through the network, this information could be used to tell host machines to analyze any processes accessing the network for abnormal behavior. This newly acquired data could then be used to improve the administrator’s decision of whether a worm is truly present. Further, information from the upper levels could be fed down into the system to affect the normal bottom-up processes of forming compound qualia, providing a way for the system to adjust its observations and decisions at individual layers based on higher level context.

Using the previous example, if a worm is suspected at the network layer, host level processes could be adjusted to be more likely to classify border-line suspicious behavior as malicious. Specifically, any programs or processes performing unusual network accesses would be examined more closely than normal. Likewise, any processes deemed to be suspicious at the host layer could then be examined again at the program layer, adjusting decision boundaries to incorporate the data from the upper layers. While this is a simple example, the ability to feed information up and down the hierarchy to influence and drive both the sensors and decision-making processes offers a potentially powerful engineering advantage over current techniques.

3.6. Defining a Cyber Model

For the system to be effective in detecting malware, a model for a computer system and network must be defined. Defining entities and relationships in the cyber realm is difficult. The cyber domain stretches from the user down through applications and software to the physical implementation of the system. Further, individual systems can be connected in large networks to share information and resources. Because of this complexity, the entities, relationships, and layers must be carefully considered.

The first design consideration is given to dividing the domain into useful layers. A proposed layering setup is:

physical, program, host, and network layers. These layers are chosen because they form well-defined boundaries that are meaningful to a human observer. The physical layer consists of the hardware used to implement the system including disk drives, memory, processors, and input devices. The program layer focuses on individual programs and software running on the system. The host layer consists of the relationships between programs and other components that form the file and operating systems of the host. The network layer follows the interactions of multiple hosts and their communications across the internet. While additional research may indicate the need for additional layers such as a processor layer that examines the operation of the processor such as cache hits/misses, core load, temperature, etc, the layers described above provide a starting set of layers for the model.

Table 1 shows the proposed list of entities and relationships which will provide the basis for modeling at each layer of the hierarchy. It should be noted that the top three layers have entities that have their own corresponding lower-level model. The network layer is composed of multiple hosts, each of which has an associated set of programs and threads, themselves having their own model at the corresponding layer. This interconnectivity between the layers can be used to either abstract information being passed up, or drill down to determine what combination of low-level actions is creating a specific high-level behavior.

A cyber example involving the Blaster worm is provided in [23]. This example demonstrates how the proposed qualia model and kernel could be used at the host layer to identify worm-like behavior without relying on byte-sequence signatures. The example follows the execution of the worm, showing how the entities and relationships within the host layer change over the time sequence. The paper also describes how a system using this type of model would be better able to handle metamorphic variants of the same strain of malware, as the model would be able to abstract beyond the implementation details. Further, the writing illustrates how incorporating data from all the conceptual layers would help improve detection at the host layer. Figure 1 shows the host layer model at the end of the Blaster example, including the major entities and their relationships.

Layer	Entities	Relationships
Network	Hosts and subnets	Connections (including port and protocol)
Host	Programs/threads, files and directories, registries and settings, libraries, sockets	Read/Write/Execute and API/System Calls
Program	Instruction blocks, data structures, API/System calls, registers	Read/Write
Physical	CPU, memory, I/O devices and storage, etc	Read/Write

Table 1. List of model entities and their relationships

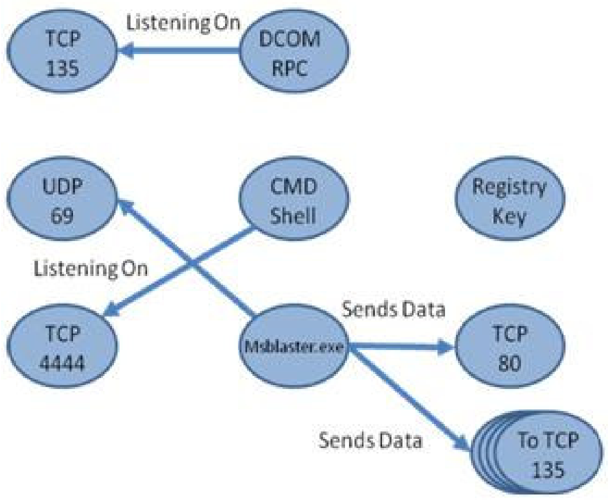

Figure 1. Host-layer model of Blaster.exe example

3.6.1. Model Training

Having established a starting model, a methodology is needed to populate the layers of the model. As discussed in Section 2.3, LDA offers potential methods of finding the desired model parameters. The model has algorithms for autonomous parameter estimation and can generate hierarchical models that fit with this research's design goals. LDA provides a way of clustering co-occurring events or entities into topics or interactions. This clustering mechanism supplies a method to combine common occurrences at the lower levels into new entities and relationships at a higher level.

To use LDA for clustering, the design of the words, topics, and documents must be slightly adjusted. LDA in its basic form is a unigram mixture model, meaning that word ordering and sequences are not considered. Word sequences, timing, and other temporal or spatial relationships are important considerations for any qualia-based model, and must be accounted for when creating the model. For example, changing the ordering of computer instructions greatly changes the output of the program. To account for this, the possible word vocabulary must be changed to accept k-gram sequences of words.

At the program layer, k-gram sequences of instructions can be defined as the vocabulary for the program documents. The k-gram sequences correspond to blocks of instructions that perform basic operations such as moving data structures. The latent topics would correspond to combinations of these blocks that perform more complex tasks. Combining a basic block that opens and reads a file to with a basic block that writes to a new file would provide the complex behavior of copying a file.

Another possible extension of the LDA method is to define chains of entity-relationship-entities as k-grams. The LDA method would then group common relationships between entities into topics. These topics can then be fed into the higher abstraction levels as entities. These entities could have their own associated relationships and be combined with sensor information from their new layer to provide a more complete model. Using the example above, the complex behavior of copying a file could be combined with a DVD-burner access that could indicate the user backing up their system files to a DVD. This functionality allows the abstraction of lower implementation details, focusing only on the general purpose of the program.

One final possible application of LDA is model changes in the model state. The vocabulary could be set as k-gram word sequences. Pieces of the sequences would include a set of entities joined by relationships representing the state of the world at a specific time. These states can then be joined by temporal relationships such as "before", "during", or "after". This provides the pattern functionality described in Section 3.4, allowing the system to learn likely transitions and incorporate temporal information into the model. These learned topic groups could then be used at higher levels as new relationships between entities. When coupled with the abstraction entities, this should provide a reduced feature set that is faster to process and can still represent the major properties of the system.

Preliminary Results

At the time of this writing, the authors have begun testing LDA as a method to perform malware detection at the program layer. Similar to [15], sequences of up to ten opcodes are set as the words in the LDA model, with a corpus of documents consisting of 300 executables from the System32 folder of the Windows XP operating system and 628 malicious executables from the VX Heavens malware database [24]. The authors consider ten opcodes to be a reasonable starting size for basic functionality blocks, providing over 150,000 repeatedly occurring opcode sequences to use as a word dictionary. However, [15] indicates that opcode sequences of length two or three may be sufficient for a classification methodology. Further testing will determine whether smaller sequences can be effective with LDA clustering.

Using LDA to cluster the sequences into 200 topics representing basic program functionalities provides a reduced feature set that can then be used with a number of classification techniques. This research differs from [15] as it uses not only the individual opcodes but also sequences that occur together in the same program. This provides an additional layer of context not available with individual sequences alone. The corpus of programs was classified into benign and malicious programs using a support vector machine (SVM) with 10-fold validation. The SVM correctly classified 81.6% of the programs with a 5.6% standard deviation.

While these results are not ideal, they show that LDA can be used to cluster opcodes into a reduced feature-set that allows basic classification in a cyber domain. The authors contend that replacing the SVM method with a qualia kernel will show improvement in the classification rate. The LDA methodology provides a way to automatically generate the hierarchical and compound entities and relationships needed to create a qualia-based model. By creating a model that can be trained with unlabeled data, the researchers hope to create a system that can reduce the need for human analysts in malware detection and allow current defenses to better keep pace with the increasing amount of malicious software.

Future Work

The next goal of the authors is to expand the testing of the LDA technique to a multi-level hierarchy of the program layer to determine how much improvement a hierarchical model will offer over the previous results. Further, the technique can be expanded to incorporate select data from multiple levels of abstraction, allowing for context from other levels to affect the classification at the current level.

After establishing the technique for a single conceptual layer, the methodology can be generalized to work across all layers including not only the program layer, but also the physical, the host, and the network layers. The SVM technique currently being used as a temporary classification kernel will be replaced with a qualia-processing kernel that incorporates temporal information and other context. The SVM method is only being used until a suitable qualia-based kernel can be developed and tested, as this writing argues an effective qualia system needs both a model and kernel operating in tandem to provide an engineering advantage. Another research area will be how to integrate this approach with other malware detection techniques to improve detection rates. Results from other detectors can be used to populate the model of the system state, or the kernel can drive the operation of multiple detectors based on its determination of which would be best given a specific situation.

Conclusions

This paper outlined some of the difficulties facing researchers and system administrators as they try to detect, identify, and prevent malware and network attacks. Current techniques focus on very low-level features that are devoid of context. This research asserts that malware is a very subjective and difficult to define concept, with many external factors determining whether a particular is considered malicious or benign or a specific system. This work further argues that identification techniques that function without considering context and other information beyond the program being examined will always struggle with malware. However, if a subjective, qualia representation that incorporates context and history can be developed, it will offer a way of describing and identifying "maliciousness".

Performing identification at this level of generalization provides greater flexibility than traditional techniques because it abstracts away implementation details and focuses on what makes a program undesirable. Detection at higher levels of abstraction allows for the system to detect malicious programs that perform similar functions even with extremely different implementations. Metamorphic malware that is difficult to detect with signatures would be more effectively detected by a system focusing on higher-level concepts. This would reduce the need to generate new signatures for every new variant of malware released, helping to mitigate the effectiveness of novel attacks that would normally be impossible to detect until a signature is created.

This research describes a proposed hierarchal model for representing a computer system and the software running on it. The paper examined LDA as a mathematical technique that can be used to generate and train the system model without prior labeling or human intervention. Further, the paper provides initial results indicating LDA's viability in creating compound abstractions from lower-level data and describes future research that can take advantage of this functionality to create a qualia-based system that will provide an engineering advantage over current detection techniques.

References

[1] Hines, M., *Malware flood driving new AV*, in *InfoWorld*. 2007.

[2] Rogers, S.K., et al., *Computing Machinery and Intelligence Amplification*, in *Computational Intelligence, The Experts Speak (Chapter 3)*. 2003, IEEE Press: New Jersey.

[3] Bruschi, D., L. Martignoni, and M. Monga, *Using code normalization for fighting self-mutating malware*, in *Conference on Detection of Intrusions and Malware and Vulnerability Assessment*. 2006, IEEE Computer Society.

[4] Christodorescu, M., et al., *Malware normalization*. 2005, University of Wisconsin, Madison.

[5] Walenstein, A., et al., *Normalizing Metamorphic Malware Using Term Rewriting*, in *Sixth IEEE International Workshop on Source Code Analysis and Manipulation*. 2006.

[6] Christodorescu, M., et al., *Semantics-aware malware detection*, in *IEEE Security and Privacy Symposium*. 2005.

[7] Krugel, C., *Behavioral and Structural Properties of Malicious Code*. 2007, Secure Systems Lab, Technical University.

[8] Krugel, C., et al., *Polymorphic worm detection using structural information of executables*, in *RAID*. 2005.

[9] Li, W., L.-c. Lam, and T.-c. Chiueh, *Accurate Application-Specific Sandboxing for Win32/Intel Binaries*, in *Third International Symposium on Information Assurance and Security*. 2007.

[10] Preda, M. D., et al., *A semantics-based approach to malware detection*. *ACM Trans. Program. Lang. Syst.* 30, 5 (Aug. 2008), 1-54. 2008.

[11] Rabek, J.C., et al., *Detection of Injected, Dynamically Generated, and Obfuscated Malicious Code*, in *2003 ACM Workshop on Rapid Malcode*. 2003.

[12] Tokhtabayev, A.G. and V.A. Skormin, *Non-Stationary Markov Models and Anomaly Propagation Analysis in IDS*, in *Third International Symposium on Information Assurance and Security*. 2007.

[13] Kolter, J.Z. and M.A. Maloof, *Learning to detect malicious executables in the wild*, in *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2004, ACM Press, New York, NY: Seattle, WA, USA.

[14] Bilar, D. Opcodes as predictor for malware. *Int. J. Electron. Secur. Digit. Forensic* 1, 2 (May. 2007), 156-168. 2007.

[15] Moskovitch, R., et al., *Unknown Malcode Detection Using OPCODE Representation,* in *Proceedings of the 1st European Conference on Intelligence and Security Informatics.* Dec. 2008.

[16] Dai, J., R. Guha, and J. Lee., *Efficient Virus Detection Using Dynamic Instruction Sequences*, Journal of Computers, Vol. 4, No. 5. 405-414. 2009.

[17] Collberg, Christian, Clark Thomborson, and Douglas Low. "A Taxonomy of Obfuscating Transformations". Technical report, Department of Computer Science, University of Auckland, 1997.

[18] Shin, J. and D.F. Spears, *The basic building blocks of malware*. 2006, University of Wyoming.

[19] Christodorescu, M., S. Jha, and C. Kruegel, *Mining specifications of malicious behavior*, in *6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* 2007, ACM: Dubrovnik, Croatia.

[20] David, M.B., Y.N. Andrew, and I.J. Michael, *Latent dirichlet allocation.* J. Mach. Learn. Res., 2003. **3**: p. 993-1022.

[21] Xiaogang, W., M. Xiaoxu, and W.E.L. Grimson, *Unsupervised Activity Perception in Crowded and Complicated Scenes Using Hierarchical Bayesian Models.* IEEE Trans. Pattern Anal. Mach. Intell., 2009. **31**(3): p. 539-555.

[22] Komatsu, L.K., *Recent views of conceptual structure.* Psychological Bulletin, 1992. **112**(3): p. 500-526.

[23] Birrer, B., et al., *Using Qualia and Multi-Layered Relationships in Malware Detection* in *IEEE Symposium Series on Computational Intelligence in Cyber Security*. 2009, IEEE Press: Nashville, TN.

[24] VX Heavens. http://vx.netlux.org/

Author Biographies

Bobby D. Birrer is a Captain in the US Air Force and is working on a PhD in computer engineering at the Air Force Institute of Technology. His research interests include software protection, intrusion detection and prevention, malware detection, and reverse engineering. He is a member of Tau Beta Pi and Eta Kappa Nu. Contact him at bobby.birrer@wpafb.af.mil.

Richard A. Raines is a professor of electrical engineering in the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base in Ohio. His research interests include computer communication networks, global communication systems, intrusion-detection systems, and software protection. Raines has a PhD in electrical engineering from Virginia Polytechnic Institute and State University. He is a member of Eta Kappa Nu and a senior member of the IEEE. Contact him at richard.raines@afit.edu.

Rusty O. Baldwin is an associate professor of computer engineering in the Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base in Ohio. His research interests include computer communication networks, embedded and wireless networking, information assurance, and reconfigurable computing systems. Baldwin has a PhD in electrical engineering from the Virginia Polytechnic Institute and State University. He is a registered professional engineer in Ohio, a member of Eta Kappa Nu, and a senior member of the IEEE. Contact him at rusty.baldwin@afit.edu.

Mark Oxley is a Professor of Mathematics in the Department of Mathematics and Statistics, Graduate School of Engineering and Management, Air Force Institute of Technology located on Wright-Patterson Air Force Base, Ohio. Dr. Oxley earned the B.S. degree in mathematics from Cumberland College in 1978 (renamed to the University of the Cumberlands in 2005), the M.S. degree in applied mathematics from Purdue University in 1980, and the Ph.D. degree in mathematics from North Carolina State University in 1987. He joined the faculty at AFIT in July 1987. Contact him at mark.oxley@afit.edu.

Steve "Capt Amerika" Rogers is a Senior Scientist at the Air Force

Research Laboratory where he serves as the principle scientific authority for Automatic Target Recognition and Sensor Fusion. Dr. Rogers' research focuses on qualia exploitation of sensor technology, QUEST. After retiring from active duty in the Air Force, Dr. Rogers founded a company (now called iCAD) for developing practical applications of advanced information processing techniques for medical products. The company invented the world's most accurate computer aided detection system for breast cancer. He has over 150 technical publications and 20 patents. Contact him at steven.rogers@wpafb.af.mil.