Building a Neural Network for Misuse Detection

Alan McAvinney   Ben Turner

December 16, 2005

Abstract

One of the fastest-growing areas of computer science research is in the area of security, specifically in intrusion detection. Much research has been done to attempt a functional and useful intrusion detection system, but so far no satisfactory solution has emerged. Recently, attention has focused on using artificial intelligence to train an intrusion detection system (IDS), rather than trying to build one from scratch. A machine learning intrusion detection system has many potential advantages over both human-engineered rule-based expert systems and purely probabilistic approaches. Of the many types of machine learning systems, neural networks offer one of the most promising methods for creating intrusion detection systems that are accurate and manageable. In this paper we present our intrusion detection system, which uses the audit logs generated by the `Audlib` software to train a simple recurrent network to flag system events as either part of an attack, or not part of an attack.

`Keywords: intrusion detection, misuse detection, machine learning, neural networks.`

1 Introduction

1.1 Intrusion Detection Systems

An intrusion detection system (IDS) attempts to identify the occurrence of unusual, illegal, and/or undesired accesses to a computer or network of computers. The IDS creates alerts which can direct a system administrator or security professional (which might theoretically be another computer program) to records of attacks, so that data can be recovered, security improved, and, potentially, legal action taken against the intruder. Thus it is important that the IDS have both a high rate of *true positives* (that is, few attacks go undetected), and a low rate of *false positives* (that is, few non-attacks are mistakenly labeled as attacks).

A great deal of work has been done in analyzing the performance of various kinds of IDSs; see Section 2 for a detailed discussion. After a review of the literature, we believe that one of the most fruitful approaches for IDSs in the near future will be in machine learning systems. In this spirit, we implement a neural-network-based IDS. A neural network, specifically a type of neural network called an Elman Network or a Simple Recurrent Network [Elm90], is a learning system which is well suited to tasks which require some temporal or contextual knowledge to complete[1]. Our project was to train a neural network to accurately classify system events as attacks or not attacks.

There are essentially two components to any IDS: a record of potentially relevant events on the computer under attack, and an indication of which events correspond to attacks. Most IDS research focuses on the latter, and rightly so; but it is worth noting that the IDS can only be as good as the data it is fed. (For a discussion of how this is an especially important concern in a neural-network-driven IDS, see Section 1.3.) For our system, we chose to use the `Audlib` audit

[1]For instance, a neural network can learn to predict that the next bit in the series `01010101` is probably `0`.

log generating package [Kup04]. `Audlib` replaces standard library calls with new versions which record the calling process, the arguments to the library call, and other useful information before letting the intended library call go through normally. We then pre-processed the data by parsing `Audlib`'s log files into lists of real values to be fed directly to the neural network as input.

1.2 Theory of Neural Networks

A neural network is based on the idea, inspired by biological neuron-based brains, that many simple nodes, densely connected, can produce complex output. Each node takes a set of real-valued inputs and outputs a single real-valued output; nodes can take their inputs from raw data or from other nodes, and their output can likewise be fed to other nodes or can represent a final result. The weights (rules by which nodes change their inputs into outputs) can gradually be updated through a process known as back-propagation, which trains the network's output towards some specified ideal.

A neural network's nodes are organized into layers, in which each node in layer `n` is connected to each node in layer `n+1`. When the concept of a context layer is introduced—that is, when part of the network's input on each time step is the values of its hidden-layer nodes from the previous time step—the network gains the ability to 'remember' what its weights were in the past, and the computational power of the network becomes truly impressive—good enough to learn a variety of tasks that would be extremely difficult, if not impossible, to engineer by hand.

It is our belief that among the tasks that can be learned by a neural network is that of deciding when an attack is taking place against a computer system. By feeding the network input that represents some important features of a particular event on the system, we can train the network to recognize events which are likely part of an attack. Furthermore, thanks to its contextual memory, the network can take the events seen previously into account when making its classification. The primary task for us as researchers, then, is to properly select features from the thousands of metrics for determining what is happening on a computer system, and present them to the network in such a way that it can make meaningful abstractions and learn the characteristics that define an attack.

1.3 Pros and Cons of Neural Networks

A neural network has several advantageous characteristics that make it an attractive choice for the intrusion detection problem. It is highly robust—resistant to the noise which will inevitably crop up in any real dataset. It is also fast enough (once its training is complete) to conceivably run in real-time on top of a real computer system. Its outputs are not limited to a simple "yes" or "no": they can be probabilistic, and they can be used to sort inputs into arbitrarily many separate categories. Most importantly, perhaps, a neural network has the ability to detect novel attacks, that is, attacks it was never been exposed to during its training. Because it works by creating and refining abstractions from raw data, a neural network learns not just what is an attack and what is not, but what makes an attack an attack.

The neural network approach does have a few significant disadvantages. First, its ability to learn a task is entirely dependent on the input data; as the saying goes: garbage in, garbage out. But this problem is made far worse by the notorious "black box" nature of neural networks: because it is difficult for a human analyst to explain the neural network's behavior in logical terms, it is possible that the network is not learning the same problem that its users think it is.[2] This black box problem can only be over-

[2]For example, if the intent is to learn to distinguish nouns from verbs in English sentences, but all the nouns in the training data happen to start with an 'S,' then the network is likely to learn to distinguish words that start with 'S' from other words—and perform very poorly when exposed to new data.

come through careful feature selection and rigorous training on a wide variety of data.

On the whole, we believe a neural network to be well-suited to the intrusion-detection task. The key benefit of a neural network over other types of IDSs, even other machine-learning-based IDSs, is its ability to abstract away from its particular inputs to learn their general characteristics and thus correctly classify new inputs that it has never seen before. In the world of computer security, when new attacks–which are, crucially, not much different from old attacks except in particulars—constantly arise, the ability of a neural network to learn to solve this type of problem is enormously appealing.

2 Previous Research

Intrusion detection systems have in recent years been divided into two groups: those which perform anomaly detection—also called behavior-based systems and those which perform misuse detection–also called knowledge-based systems [DDW99]. Misuse detection systems attempt to determine whether an attack has occurred by scanning the system's audit data for occurrences of known attacks. Anomaly detection systems rely instead on a statistical analysis of the system's behavior, signaling out anomalous activity as likely attacks. Each approach has its drawbacks, the most important being that misuse detection systems cannot detect intrusions which do not match the profiles in their set of known attacks, while anomaly detection systems can potentially erroneously classify acceptable behavior as an attack, or conversely may mistake attacks for legitimate use. Machine learning approaches have been suggested for both types of intrusion detection system.

In 1997 Lane and Bradley attempted to build a learning system for anomaly detection [LB97]. Their objective was to learn a profile for each legitimate user in the system, and then examine future actions by the users, using the learned profile to determine if an anomaly had occurred. User profiles were comprised of sequences of actions (information about command names, behavioral switches, and number of other arguments for each command each user entered at the shell prompt). Once the profiles were built, each subsequent sequence was compared against the appropriate user's profile; a similarity function determined if the new sequences were normal or abnormal. While the authors may be right to believe their system shows promise, their results are based on data from only four users. It is also worth noting that the system is not truly a learning system, as it makes no attempt to generalize from its input—it simply compares sequences it has seen to new sequences, in a deterministic way that is defined by the authors, not the system itself.

Ghosh, Schwarzbard, and Schatz's 1999 paper examines three anomaly detection systems, the last of which is a true learning system which employs a neural network to detect anomalies [GSS99]. The authors collected audit data from their network using Sun Microsystem's Basic Security Module, a built-in auditing tool on Solaris machines. They then tested three techniques on their ability to correctly identify anomalous data: an equality matching algorithm, a simple feed-forward backpropagation neural network, and a simple recurrent network (Elman net). Each system performed better than the one before, and the authors claim that the Elman net could detect 77.3% of intrusions with no false positives, and 100% of intrusions with "significantly fewer false positives than either of the other two systems." This result demonstrates the potential of a neural network solution to the intrusion detection problem and encourages further research into the performance of Elman nets at the intrusion detection task.

Another application of neural networks is presented in Cannady's 1998 paper [Can98]. The author mentions the primary shortcoming of rule-based systems for misuse detection, namely that their set of known attacks is extremely unlikely to be complete, and proposes a misuse detection system which utilizes a neural network to learn what characteristics are present in an attack, and then flag future events as attacks or not. The system was trained on a set of data representing network packets, some of which were known to be legitimate and some of which were known to be attacks. When it was presented with another set of similar data that had not been available to it during training, the network correctly identified packets as attacks or not. This result suggests that not only can a neural network learn to identify anomalous behavior, it can also learn the characteristics that are shared by various attacks and apply that knowledge to detect system misuse.

3 Our Experiment

3.1 The Network

To implement our neural network, we used the tools provided by `Pyrobot`, a Python library [BKea]. We set up a standard simple recurrent network with three fully connected layers: input, hidden, and output. (See Figure **??**.) The input layer consisted of three components: the audit data, a standard contextual memory layer (i.e. a copy of the previous step's hidden layer) and a per-process contextual memory layer (a copy of the hidden layer from the last time step in which the data came from the same process it is currently coming from). The hidden layer had N nodes each, and thus each context layer had N nodes also. The output consisted of a single node, which we refer to as "the classification bit" (although in truth it is a real value between 0 and 1, not a binary 0 or 1). The classification bit indicated the presence or absence of an attack; values below a parameterized threshold corresponded to "not an attack," while values above another threshold meant "attack."

3.2 Data Gathering and Pre-Processing

The first stage of our experiment involved collection of data using the `Audlib` tool. Two datasets were created: a training set and a testing set. The training set consisted of data that would be used to train the neural network to identify attacks. The testing set consisted of data that would be used to test the performance of the neural network on novel data. Each dataset contained examples of both normal system usage and attacks. To generate attack data, we used a suite of attack tools from `www.metasploit.com`.

Next we parsed the raw data to process it into a form suitable for use by a neural network. We transformed the data into a set of real-valued inputs. The features available to the neural network were:

- Library Call Name (hashed, normalized to range [0,1])
- For each argument to the system call:
  - argument size
  - argument type
  - argument value (strings are hashed; all values normalized)
- PID of calling process (normalized to range [0,1])
- PID of calling process' parent process (normalized)
- Real UID of calling process (normalized)
- Effective UID of calling process (normalized)
- Saved UID of calling process (normalized)
- Real GID of calling process (normalized)
- Effective GID of calling process (normalized)
- Saved GID of calling process (normalized)

To avoid inadvertently teaching the neural network to learn that specific UIDs are associated with attacks, we made sure that there was no UID which appeared only in attack data. In the real world, this might not be the case; there might in fact be some user id which was only used by an attacker. But it is far more likely that an attacker would impersonate a legitimate

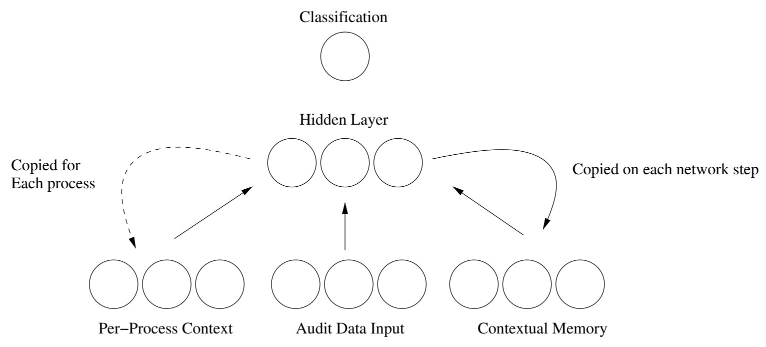

Figure 1: The network architecture.

user (or superuser) and we had to be sure that our IDS could detect this type of activity.

3.3 Training

Once this pre-processing was complete, we could begin training the neural network. Using the timing information from the `Audlib` tool, we sent input data to the network in chronological order (note that the network itself had no access to the time stamps). The network was trained to output a 1.0 on events known to be an attack and a 0.0 on events known to be not an attack.[3] To train the confidence bit, we simply told the network to try for a value of 1.0 when its classification bit matched the expected value, and 0.0 otherwise. The network was trained until its performance on the training data reached an acceptable level of accuracy. To prevent overtraining (the phenomenon of the network learning its training data too specifically, thus making it unable to evaluate novel data), we periodically turned off learning and tested the network on the testing data; if its performance on this data was better than ever before, we saved the network's nodes' weights to a file. Thus the file saves only the weights which performed best on the independent testing data.

[3]There is a small but non-zero chance that some of our events were mislabeled in the training data, because we have no guarantee that an attack was not taking place against the system while we were collecting data. There is no evidence of such an attack taking place, however. We believe that the only attacks present in the data are those that we created ourselves.

4 Results (or lack thereof)

We defined an "epoch" as the amount of time it took to train the network once on each piece of data in the training set. We trained the network for as many epochs as possible before stopping it to evaluate our results. When we did begin to evaluate the results, we found that the network reported a 100% accuracy on its input after the first epoch—that is, the network thought that it perfectly classified each piece of data. Obviously this result is highly unlikely. We are forced to conclude that either our network implementation or our data is flawed, and therefore have no meaningful results to report.

It should be noted that, even had this unexpected bug not occurred, any results from our current data would be preliminary at best. We are confident in the quality of our normal (i.e. non-attack) data, but our attack data, which consists of merely running (unsuccessfully) some standard attack tools downloaded from the Internet, is probably not sufficiently varied to provide a useful basis for the network to learn about attacks in general.

5 Directions for Future Work

This experiment demonstrates nothing conclusively about the viability of neural network-trained misuse detection systems. A more diverse set of attack data is required before a strong conclusion can be made. Thus the first step toward extending this experiment in the future would be a thorough collection of attack data from as many sources as possible. The term "critical mass" is a useful one: a neural network must have enough good input data to learn something meaningful about the general classes of data it encounters, but after a certain point, no new data must be collected because the neural network will have learned everything it can about the problem. So we are hopeful that the classic computer security problem of the hackers remaining "one step ahead" can be overcome with enough effort. Collecting new attack data also has the side benefit of stress-testing the relatively new `Audlib` system. Additionly it would be much better to collect enough attack data to make a significant percentage of the training data attacks, in order to prevent catastrophic forgetting[4].

An ideal way to collect this attack data would be through use of a Honeypot[5]. This provides a diverse set of attack data with minimal effort from the researcher, as well as data on common types of misuse, and has the benefit of recording the actual actions of hackers, worms, and viruses in the wild, rather than the simluated attacks we used in this experiment.

Once a suitable corpus of input data is gathered, the experiment should be run again in the same manner as described in this paper. Ideally multiple neural networks could be trained simultaneously with varied network parameters, since most of the ideal values for any given parameter of the network (such as the number of hidden nodes, the learning rate, and the momentum, if any) are determined experimentally on a case-by-case basis.

Finally, the viability of a neural network which provides more detailed information about the attacks it identifies should be explored. In the current experiment, the network outputs only one value, giving a "yes/no" answer for the question "Is this library call part of an attack?" But a neural network could theoretically be trained to perform a much more complicated task, such as classifying the inputs into N distinct categories based on what kind of attack they were associated with. An IDS which not only signals that an attack has taken place but also identifies something about the nature of the attack would, one imagines, be quite useful, and the current experiment should be extensible to this problem with relative ease—provided that the input data is carefully classified as it is collected, either by a human expert who knows what type of behavior to expect from a given attack, or by another IDS which has proven to be adept at identifying the most common attacks. Again, the collection of attack data is a time-consuming one, but, thanks to the neural network's power of abstraction, the task will have a definite (though perhaps not well-defined) end.

6 Acknowledgements

Thanks to Prof. Ben Kuperman for the `Audlib` tool and numerous helpful suggestions and comments; Prof. Lisa Meeden for the Pyrobot library and for introducing us to neural networks; the Swarthmore College class of 2006 Computer Science Majors; and everyone who allowed their activity on the system to be logged for our use.

References

[BKea] Douglas Blank, Deepak Kumar, and et al. Pyro: A python-based versatile

[4]A pheonomenon in which a neural network learns a task, then after training on a sufficiently large number of inputs of a different nature, no longer succeeds at the original task

[5]A Honeypot is a computer with no legitimate use—any activity on it must therefore be an attack

programming environment for teaching robotics.

[Can98] J. Cannady. Artificial neural networks for misuse detection. In *Proceedings of the 1998 National Information Systems Security Conference (NISSC'98) October 5-8 1998. Arlington, VA.*, pages 443–456, 1998.

[DDW99] Herve Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31, 1999.

[Elm90] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[GSS99] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pages 51–62, April 1999.

[Kup04] Benjamin A. Kuperman. *A Categorization of Computer Security Monitoring Systems and the Impact on the Design of Audit Sources.* PhD thesis, Purdue University, West Lafayette, IN, 08 2004. CERIAS TR 2004-26.

[LB97] T. Lane and C. E. Brodley. An application of machine learning to anomaly detection. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 366–380, 1997.