A neural network-based IDS

Cornel Izbaşa *
cornel@ieat.ro

Institute e-Austria

Abstract. This paper describes the implementation of a neural network-based intrusion detection system. The main components of the IDS are a custom neural networks library that provides a simple API for feed-forward neural networks trained with backpropagation, a multi-string matching library that uses the `Aho-Corasick` algorithm and which was adapted from a module of the `Snort` IDS and a packet-capture module based on `libpcap/libnids`.

1 Origin of idea

The idea of an IDS based on neural networks came from [4], where the authors describe a solution for heuristic virus detection for Win32 with neural networks. I have not improved much on the classification technique but I am however considering networks with multiple hidden layers, and this was one of the reasons to write a custom neural net library, that so far provides only feedforward neural nets, with a very simple API, and ease of implementing neural nets with an arbitrary number of layers.
The main contribution of this work is the application of the techniques used by the team from `IBM` for heuristic virus detection for an antivirus solution in the realm of intrusion detection systems. This is a natural idea since the are plenty of similarities between viral code and shell-code/attack code used in the exploits of various vulnerabilities. Like in the VX realm there are shellcode/attack code templates and generators, polymorphic shellcode of various types, alphanumeric shellcode and a very wide variety of payloads. Many vulnerabilities are used by both virus writers and exploit developers, and it has become a common phenomenon that once there is a working exploit for a vulnerability it takes only little time before it is used in writing a virus/worm. There are also highly automated root-kits that are employed by crackers that are very hard to distinguish from viruses, given that they automatically scan for vulnerabilities and exploit more hosts once inside a system, so even if the two cultures, that of exploit writers/crackers and virus writers (VX'ers) are quite different in their evolution, the dividing line has become very thin lately, and there certainly are people that are highly active in both realms.

* I thank my colleagues at the IeAT for their support, especially my team leader Marius Minea who's granted me the time to develop this project and Daniela Zaharie for introducing me to neural networks and reviewing the neural nets library.

2 The implementation

2.1 The neural nets library

As described in the previous section, the neural nets library was written from scratch and so far offers only feedforward neural nets trainable with backpropagation, but the API is very simple and hopefully extensible. It supports saving and loading of networks with their associated functions. For example, the following snippet trains and saves a neural net with one hidden layer that approximates the xor function. It has a maximum of 20000 training epochs, an error threshold of 0.001 and an initial learning rate of 0.25 and uses `tanhl` as transfer function.

```
initnn(&nn1, 0.25, 0.001, 20000, 2, 2, 4, 1);
initnnf(&nn1, tanhl, tanhld, tanhl, tanhld);
read_ts(&s, "xor.dat");
BACKPROP(&nn1, &s, 1, -1, 1);
savenn(&nn1, "xor.nn");
```

The classifier neural net should be useful because a lot of the attack signatures are correlated, but it may be quite hard to specify all correlations in a quasi-complete manner. For example, if we encounter an input buffer that contains a reference to `/bin/sh` and also a `setuid()` syscall then the transmission is much more suspect then if only one of them were present, but in many cases we could have correlations between more signs of attack and they may be much harder to express.

2.2 The Aho-Corasick library

The Aho-Corasick - see [7]- library was obtained by adapting a module that implements the Aho-Corasick multi-string matching algorithm for integration with this neural net based IDS. The original module, version 2.0, is written by Marc Norton and Dan Roelker for `Snort` - the Open Source IDS of choice - [5]. The Aho-Corasick algorithm is also used by the OpenAntivirus Project - see [6] - for virus signatures matching. The Aho-Corasick algorithm searches n strings within m bytes with runtime O(m).
The ability to use XML files for attack description is provided by the `expat` library.This is a sample XML file for attacks description:

```
<classes>
 <class name="shellcode">Shellcode attacks
  <attack name="setuid" platform="GNU/Linux/x86">setuid syscall
   <bytes>6a 17 58 31 db cd 80</bytes>
  </attack>
  <attack name="//sh" platform="UNIX">reference to //sh
   <bytes>2f 2f 73 68</bytes>
  </attack>
 </class>
</classes>
```

2.3 The capture module

This module is not yet complete. So far it is using `libpcap`, the packet capture library used in a lot of other Open Source projects like `nmap`, `tcpdump` and `ethereal`. I am considering using `libnids` as it provides TCP packet reassembly.
Of course, the idea is that a variety of capture modules could be written, so the neural nets and Aho-Corasick libraries could be employed with equal ease in an antivirus/anti-malware solution that instead of scanning packets scans files for viral code/shellcode/other malware as in [6].

2.4 The scanning process

Once data to be scanned is made available by the capture module, the Aho-Corasick module is used for attacks signatures matching, each signature representing a feature. An input vector for the trained neural net is generated with ones for present features and zeroes for absent features. The network classifies the data according to the presence of the features, and based on its class (e.g. viral/shellcode/clean) reporting and/or blocking actions could be taken.

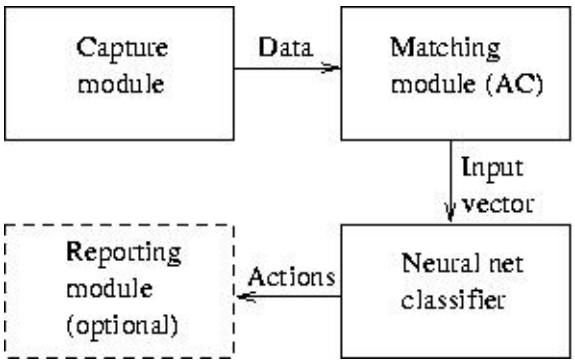

Fig. 1. Scanning process

2.5 Future work

This needs automated feature extraction badly, since otherwise a lot of the use of neural nets is wasted. This may prove to be quite hard in practice and one of the main reasons is the difficulty in legally using the study materials (viral code, shellcode, exploits etc.) and this situation is very bad for all areas of information systems security.

References

1. Daniela Zaharie, *Curs de rețele neuronale*, web resource.
2. Gerald Tesauro and Jeffrey O. Kephart and Gregory B. Sorkin, *Neural Networks for Computer Virus Recognition*, IEEE Expert vol. 11, no. 4, Aug. 1996, pp. 5-6.
3. Jeffrey O. Kephart, Gregory B. Sorkin, Morton Swimmer, and Steve R. White, *Blueprint for a Computer Immune System*, Virus Bulletin International Conference, October 1-3, 1997.
4. William Arnold and Gerald Tesauro, *Automatically generated Win32 heuristic virus detection*, Virus Bulletin conference, September 2000.
5. Martin Roesch and Chris Green, *The Snort Users Manual*, web resource.
6. Kurt Huwig and Rainer Link, *OpenAntivirus Project Presentation*, Linux Kongress 2003, Saarbrücken, Germany.
7. Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, *Data Structures and Algorithms*, Addison Wesley, January 1, 1983.
8. Mark Ludwig, *The Big Black Book of Computer Viruses*, American Eagle Publications, 1995.