INL/CON-09-15944
PREPRINT

Neural Network Approach to Locating Cryptography in Object Code

Emerging Technologies and Factory Automation

Jason L. Wright
Milos Manic

September 2009

Neural Network Approach to Locating Cryptography in Object Code

Jason L. Wright
Idaho National Laboratory
2525 Freemont Avenue
Idaho Falls, ID 83415, USA
jlwright@ieee.org
jason.wright@inl.gov

Milos Manic
Department of Computer Science
University of Idaho at Idaho Falls
1776 Science Center Dr., Ste. 306
Idaho Falls, ID 83402, USA
misko@ieee.org

***Abstract*—Finding and identifying cryptography is a growing concern in the malware analysis community. In this paper, artificial neural networks are used to classify functional blocks from a disassembled program as being either cryptography related or not. The resulting system, referred to as NNLC (Neural Net for Locating Cryptography) is presented and results of applying this system to various libraries are described.**

***Index Terms*—cryptography, neural networks.**

I. Introduction

Finding and identifying cryptography is a growing concern in the malware analysis community. The current state of the art is to locate the cryptographic routines manually and identify them based on the constants used by specific algorithms[1]. The *Neural Net for Locating Cryptography* (NNLC) system described in this paper examines the instructions or opcodes that make up each function in a given binary and and determines the likelihood that the function contains cryptography. NNLC uses the classification power of artificial neural networks to accomplish this goal.

A. Background

This work was inspired by two pieces of previous work: *findcrypt[2]/findcrypt2[1]* and the compromise of the Mifare smartcards[3]. *findcrypt* and *findcrypt2* locate various constants used in the initialization of cryptographic algorithms and further provide identification of the specific algorithm. This work differs because it examines the instructions that make up the algorithm and not the data it uses (for initialization or otherwise).

The crack of the Mifare smartcards[3] involved reverse engineering the hardware by examining the distribution of logic gates. The authors looked for the properties of cryptographic algorithms that make them stand out from normal functionality. Specifically, the authors looked for a high density of XOR gates in a given area of the chip. They also looked for blocks of gates that were strongly interconnected, but where the functional block itself was loosely connected to the rest of the chip. In other words, the authors were looking for the pipelining components (strong interconnection) of the cryptographic algorithm, and its inputs and outputs (loose coupling to the rest of the chip).

Other works have attempted to classify files. In [4], n-grams of bytes are used to determine the type of a file (document, spreadsheet, etc.). In [5] and [6], opcodes of executable files were analyzed to determine whether the executable was malware. Both of these works used various analysis techniques (Bayes, frequency analysis, etc.) in an attempt to determine if the distribution of opcodes differs substantially in malware from that of a normal executable file. NNLC differs primarily in the fact that it uses opcode frequency analysis in an effort to determine whether an individual function within a binary is cryptographic, not whether the program as a whole is malware.

In [7], simplistic frequency analysis was used to make a similar judgment. In this case, the classification of individual functions was examined to determine whether the function was cryptographic in nature. The simplistic nature of the classification algorithm was limited in flexibility and was expert driven instead of relying on computational intelligence.

B. Paper Organization

The rest of this paper is organized as follows. Section II describes the primary application of this solution and the motivation for the research. Section III describes the general method used for the solution, and Section IV provides the detailed solution to the problem. Results of tests are given in Section V, and Section VI provides the conclusion.

II. Problem Statement

It is not uncommon for a piece of malware to employ encryption techniques in its communications. In particular botnets employ encryption to help conceal their communications[8], [9]. When analyzing malware, a researcher must determine which function is performing the encryption and then determine which particular algorithm is in use.

This manuscript has been authored by Battelle Energy Alliance, LLC under Contract No. DE-AC07-05ID14517 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

The goal of this work is primarily in classifying whether a function is cryptographic in nature or not, and providing a method of quick location of the most cryptographic-like functions found when examining a given binary. To do so, a neural network is used because of its suitability to supervised learning when a training set can be defined in advance.

III. Method

For NNLC, a neural network is trained with Error Back Propagation (EBP)[10], [11]. A hyperbolic tangent activation function is used for each of its neurons, and this results in two variables to be adjusted when training the network: the learning constant (α) and the gain of the hyperbolic tangent (k). The learning constant affects the speed at which the network learns. The gain of the hyperbolic tangent affects the decision boundary, allowing for a smoother transition from positive to negative decisions.

The architecture of the neural network also has great influence over the correctness of the solution. The inputs (feature extraction), number of layers, number of neurons in each layer, and outputs are all a part of the architecture of the network.

Also for a neural network solution, an appropriate set of training data must be available. This data is used to generate the various weights used by the network. For this work, the C library from the OpenBSD operating system was used. This version of the C library includes several cryptographic functions: SHA1, MD5, RMD160, DES, SKIPJACK, RJINDAEL (AES), and BLOWFISH. These functions were used to positively train the network, and the rest of the C library was used to negatively train the network. Additionally, a selection of algorithms from [12] were used (TEA, IDEA, GOST, RC5, and LUCIFER) in training the network.

To test the algorithm, various optimization levels of the C compiler were used to recompile the C library. Optimization level should have little effect on the results if the neural network is trained appropriately. Also, the network was applied to the OpenSSL library to see how it classifies the algorithms found therein. The results of these tests are given in Section V.

IV. ANN Approach to Locating Cryptography

For NNLC, the inputs to the artificial neural network (ANN) are the total number of instructions with the following opcodes: XOR (exclusive or), SHL (logical shift right), SHL (logical shift left), ROR (rotate right), and ROL (rotate left). Additionally, the densities of these instructions are applied to five more inputs. Density is defined here as the number of the specific opcodes divided by the total number of opcodes in the function.

Various architectures were tried for the neural network. The architecture that yielded the lowest total error consisted of 10 input neurons, 5 neurons in a single hidden layer, and a single output neuron. The inputs are shown in Table I; there are two inputs for each type of instruction: total (σ) and density (ρ). Additional layers increased the total error. The number of hidden-layer neurons was kept purposefully low to avoid over-specializing the network. The goal is not to identify particular implementations of the specific algorithms, but to keep the network general enough to spot unknown implementations of known algorithms or even unknown algorithms.

Input	Description
σXOR ρXOR	Exclusive Or
σROR ρROR	Rotate right
σROL ρROL	Rotate left
σSHR ρSHR	Shift logical right
σSHL ρSHL	Shift logical left

TABLE I
NNLC NEURAL NETWORK ARCHITECTURE

Bipolar neurons are used with a hyperbolic tangent activation function. The output neuron reports a value $o \in (-1, 1)$. Values close to 1 are considered to be cryptography related and values close to -1 are not.

The network was generated with random weights on each neuron, and error back propagation was used to train the network. Each function in the default C library was assigned a target output value (o, above) where 1 is a cryptography function and -1 is not. One hundred iterations of the training set was used with a hyperbolic tangent gain of $k = 0.2$ and a learning constant, $\alpha = 0.5$.

The particular instructions chosen are often used in modern cryptography. The shift and rotate instructions are commonly used to perform the diffusion part of the confusion and diffusion used in modern cryptography. XOR is typically part of the confusion.

V. Test Results

Several tests of the NNLC system were performed. Section V-A gives the result of training the network on the C library. Section V-B examines the effect of compiler optimization on the accuracy of the NNLC system, and finally Section V-C gives the results of running NNLC against a completely independent library.

A. Default C Library

The OpenBSD C library in version 4.4 consists of 2076 functions, everything from *printf()* to *gethostbyaddr()*. Of those, 22 are cryptography functions, implementations of SHA1, SHA2 (SHA256 and SHA512), RMD160, MD4, MD5, SKIPJACK, BLOWFISH, CAST, and DES. In addition to the OpenBSD C library, implementations of the TEA, IDEA, GOST, RC5, and LUCIFER algorithms were used to train the network. The compiler used was *GCC* version 3.

Total error is the sum of the desired output and the computed output of the network as in Equation 1. In the equation, i runs through the output of each of the n test cases.

$$TE = \sum_{i=1}^{n} (d_n - o_n)^2 \quad (1)$$

When computed this way, the minimum total error achieved for the artificial neural network is $TE = 60.92$. Put another way, 18 functions are misclassified. Eight functions are classified as cryptography but are not (false positive) and nine are not classified as cryptography but are (false negative).

The false positive functions are:

- `abs-0x38()`
- `div-0x3a()`
- `htonl-0x3a()`
- `htons-0x3a()`
- `labs-0x3a()`
- `ldiv-0x3a()`
- `ntohl-0x3a()`
- `ntohs-0x3a()`

These functions are short a contain a large number of XOR opcodes. Therefore the density of XOR opcodes in these functions is very high. All of these functions are optimized versions of their respective functionality and perform bitwise operations.

The following functions are correctly identified as cryptography:

- `Blowfish_decipher()`
- `Blowfish_encipher()`
- `cast_decrypt()`
- `cast_encrypt()`
- `cast_setkey()`
- `MD4Transform()`
- `MD5Transform()`
- `RMD160Transform()`
- `SHA1Transform()`
- `SHA256_Transform()`
- `SHA512_Transform()`
- `skipjack_backwards()`
- `skipjack_forwards()`

A common factor in each of these algorithms is that they are all Feistel networks (a cipher that is iterated over an internal round function). A Feistel network is a subset of product ciphers where each round consists of simple transformations such as substitution, permutation, and modular arithmetic[13]. XOR is a modular arithmetic operator, shift and rotate instructions are commonly used in permutation operations.

In Figure 1, the classification success of NNLC is depicted. The samples are ordered along the horizontal axis by their output values and plotted on the vertical axis by their absolute output value. Only the false positive, false negative, and correctly identified cryptography points are displayed. What this shows is that there is one outlier cryptographic algorithm and the false positive algorithms generally classify between this outlier and the cluster of correctly identified algorithms. The false negative classifications cluster very close to non-cryptography.

Table II summarizes the results in a different way. The total percentages for false possible and negative are given. It shows that out of the 2076 functions in the C library, 99.13 are classified correctly as either being cryptography or not.

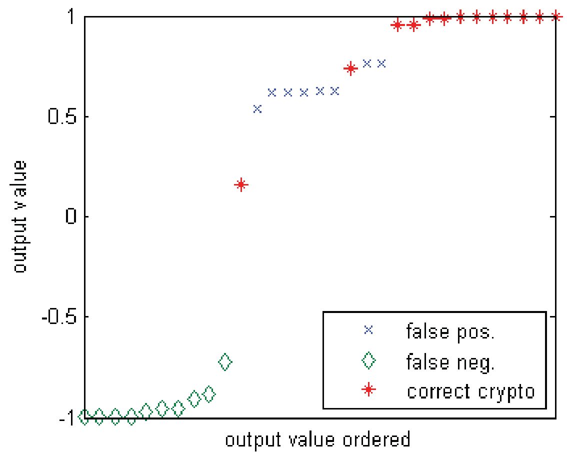

Fig. 1. Classification visualization

Total Functions	2076	
Correct classification	2058	99.13%
False negative	8	0.39%
False positive	10	0.48%

TABLE II
OPTIMIZATION EFFECT ON TOTAL ERROR

B. Effect of Optimization

As a comparison, the trained network was used on a recompiled C library with various optimization levels. Table III shows the total error versus the optimization level of the compiler used. O0 is a way of specifying that the compiler attempt no optimization, and O3 specifies all optimizations be considered. Generally speaking, the transition from O0 to O3 additively enables further optimization.

Of particular note from the table is that the total error for optimization level 3 is much lower than that of the other levels. Only 10 functions are misclassified (2 false negative, 8 false positive). This is probably related to loop-unrolling and other optimizations allowed at this level.

Also, the percentage of correctly classified functions does not change drastically. It remains above 99% for all optimization levels. This lends some weight to the validity of the approach used in NNLC.

Opt. Level	TE	False Pos.	False Neg.	Correct
–O0	60.68	8 / 0.39%	11 / 0.53%	2057 / 99.03%
–O1	60.90	8 / 0.39%	10 / 0.48%	2058 / 99.13%
–O2	60.92	8 / 0.39%	10 / 0.48%	2058 / 99.13%
–O3	30.17	8 / 0.39%	2 / 0.10%	2066 / 99.51%

TABLE III
OPTIMIZATION EFFECT ON TOTAL ERROR

C. OpenSSL library

The trained network was also run on the OpenSSL library from the OpenBSD 4.4 reference machine. This library has many cryptographic functions and most of the implementations are independent of the versions in the C library. The network correctly classifies: MD5, AES, SHA1, DES, ACSS, RMD160, MD4, CAST, and BLOWFISH.

The library also contains Ecliptic Curve Cryptography algorithms and public key algorithms (RSA, DSA, etc.). These algorithms are not identified and the reason for this is that they are simple operations (like modular exponentiation) performed on large (512 bits or greater) numbers. It may be possible to identify the micro operations (big number, modular arithmetic), but classifying the macro operation (RSA) would be difficult from the information considered by NNLC.

VI. Conclusion

An artificial neural network for classifying algorithms as being cryptographic in nature or not is presented. The network takes advantage of error back propagation to train a network to differentiate cryptography algorithms from other functions. The primary use for such a tool is in malware analysis where a researcher must locate the cryptography used by a particular sample in order to begin the process of identification.

The primary direction for future work is to decrease the total error (increase the correctness of classification). It is believed that more vectors can be used in this capacity. In particular, the use of jump instructions is not common in the functions that are currently misclassified. Also, floating point operations should be considered as a negative indicator. Cryptography requires bit-for-bit symmetry on encryption and decryption and this is not guaranteed with floating point operations across processor architectures.

The current implementation of NNLC is targeted at the Intel IA32 instruction set. Initial testing on the SPARC version 9 architecture have proved promising. This leads the authors to believe that NNLC could be applied to other architectures with minimal modification. Also, the tests so far have been done with GNU Compiler Collection (*GCC*), but results from other compilers need to be included.

Finally, the EBP training method used was chosen primarily for its ease of implementation in this initial research. Training the network was rather slow (order of 10 minutes per architecture tried). Other training methods like hypersonic training, Levenberg-Marquardt, etc. and the resulting networks will be compared in the future.

References

[1] I. Guilfanov, "FindCrypt2," February 2006, http://hexblog.com/2006/02/findcrypt2.html.

[2] ——, "FindCrypt," January 2006, http://hexblog.com/2006/01/findcrypt.html.

[3] K. Nohl, D. Evans, Starbug, and H. Plötz, "Reverse-engineering a cryptographic RFID tag," in *USENIX Security Symposium*, July 2008, pp. 185–194.

[4] W.-J. Li, K. Wang, S. J. Stolfo, and B. Herzog, "Fileprints: Identifying file types by n-gram analysis," in *IEEE Workshop on Information Assurance*, USMA, West Point, NY, June 2005.

[5] D. Bilar, "Opcodes as predictor for malware," *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 2, pp. 156–168, 2007.

[6] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, "Unknown malcode detection using opcode representation," in *Proceedings of the 1st European Conference on Intelligence and Security Informatics EuroISI*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 204–215.

[7] J. L. Wright, "Finding cryptography in object code," in *Security Education Conference Toronto (SecTOR)*, October 2008.

[8] J. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: overview and case study," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–1.

[9] K. Chiang and L. Lloyd, "A case study of the rustock rootkit and spam bot," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. Berkeley, CA, USA: USENIX Association, 2007, pp. 10–10.

[10] D. E. Rumelhart and J. L. McClelland, *Parallel distributed processing: explorations in the microstructure of cognition*. Cambridge, MA: MIT Press, 1986, vol. 1.

[11] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley-Interscience, January 1994.

[12] B. Schneier, *Applied Cryptography*. John Wiley and Sons, 1996.

[13] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, 5th ed. CRC Press, August 2001.