Research Article

International Journal of Advanced Computer Research, Vol 7(31)
ISSN (Print): 2249-7277 ISSN (Online): 2277-7970
http://dx.doi.org/10.19101/IJACR.2017.730022

Fuzzy zero day exploits detector system

Adnan Shaout[1*] and Cameron Smyth [2]
The University of Michigan-Dearborn[1]
The Department of Electrical and Computer Engineering Dearborn, Michigan[2]

Received: 23-February-2017; Revised: 03-June-2017; Accepted: 06-June-2017

Abstract
Intrusion detection systems today are relatively capable of detecting network intrusions by attackers. Unfortunately, these systems operate on a network level and not on a system level. Meanwhile, antivirus software is typically capable of detecting known viruses but cannot easily stop zero day exploits. The paper will propose a fuzzy inference system to detect exploitation of a system using system metrics such as CPU, memory usage and network connections. This system is implemented using the MATLAB fuzzy logic toolbox. The design was tested and provided reasonable results.

Keywords
Intrusion detection system, Fuzzy exploit monitor, Fuzzy inference system, Computer security, Zero day exploits.

1.Introduction
Computer systems have been plagued by viruses for many years. They were first conceptualized as far back as 1949 when John von Neumann theorized about "self-replicating automata" that could reproduce and propagate itself [1]. The first computer virus to appear outside a lab environment is believed to be the "Elk Cloner" virus written by Rich Skrenta, which was originally written as a practical joke [2].

Since then, many far more dangerous viruses such as Blaster, Conficker and "ILOVEYOU" have infected PCs world wide. Many efforts have been made to intercept and prevent viruses from running. They are often marketed and sold in the form of "antivirus" programs. The majority of these programs function by searching for or detecting "signatures" of viruses – by comparing the hash of the file's contents against a database of known virus hashes, and then preventing the code from executing and even removing the file from the file system automatically. Unfortunately, these approaches tend to be "cat and mouse games;" they require viruses to be known and available in the provided database before they can be stopped, often meaning new or "zero day" exploits can go uncaught for a period of time. The goal of the fuzzy exploits monitor is to detect these unknown viruses based on unusual computer conditions.

*Author for correspondence

This can take many forms-high CPU usage, high memory usage, unsolicited network connection attempts, constant disk I/O, or other conditions. A user may then be alerted of the unusual conditions, or the system may autonomously take action if it is confident enough.

The concept of this paper is to model a system utilizing fuzzy logic to identify unusual conditions in a computer system and attempt to classify the degree to which the system is compromised. The model will take a number of inputs, including CPU and memory usage, disk I/O, and network connections. These inputs will be used to determine the relative "normality" of the system's behavior and therefore extended to identify if the system is believed to be compromised.

The inputs must be based on real data and a combination of intuition and inference will be used to design the system. For simplicity, the model is likely to be designed using simple membership functions and relationships. However, neural networks are recognized as a more advantageous approach as they would allow a fuzzy virus detector to learn a normal system's behavior and identify unusual circumstances that may be indicative of a bug or exploit.

One potential challenge facing the research for this paper will be identifying scenarios where there may be high CPU usage, high memory usage and high disk I/O due to a valid process such as video rendering.

While outside the scope of this paper, a neural network could potentially be used to identify valid scenarios based on a usage history and set of known good processes. Note, however, this would not be a bulletproof mechanism – many hacking techniques involve exploiting vulnerabilities in normal applications and masquerading within their processes, so extra care would be needed to develop this neural network to identify unusual behaviors from valid applications.

The paper is organized as follows: related works have been discussed in section 2, methodology and design have been discussed in section 3, proposed system implementation has been discussed in section 4, section 5 has covered test and results and finally conclusion have been included in section 6.

2.Related works

Intrusion detection systems (IDS) are designed to monitor network traffic and alert a system administrator or other responsible individual in the event of potential network intrusion from adversaries. The concept can be traced back to at least 1980, on a proposal by James Anderson which consisted of a set of tools by which system administrators could identify attacks via audit trails [3]. A concept for a more active IDS was first proposed in 1986 by Dorothy Denning based on analyzing system logs and other auditing records [4]. These systems evolved over the next decade to include statistic-based detection (pioneered by Vaccaro and Liepins [5]) and include preventative measures.

Over the most recent decade, numerous studies have attempted to identify fuzzy methods of detecting network intrusions. Abadeh, et al., proposed a fuzzy genetics-based learning algorithm that could be used as a network intrusion detection system [6]. Wang et al., defined a method combining artificial neural networks and fuzzy clustering to improve the capabilities of previously proposed systems to detect low-occurrence attacks [7]. Mkuzangwe et al. [8] presented a fuzzy logic based network intrusion detection system to predict Neptune, which is a type of a transmission control protocol synchronized (TCP SYN) flooding attack. Shanmugavadivu et al. [9] introduced a network intrusion detection system using fuzzy logic. Kudłacik et al. [10] presented an intrusion detection method based on a fuzzy approach. Azad et al. [11] introduced an intrusion detection system which is based on the fuzzy min, fuzzy max neural network and the particle swarm optimization. Ramakrishnan et al. [12] proposed an entropy-based feature selection to select the important features, layered fuzzy control language to generate fuzzy rules, and layered classifier to detect various network attacks namely neptune, smurf, back, and mailbomb. However, no research could be found on fuzzy methods of detecting exploitation within a single system.

The above described intrusion detection systems, both fuzzy and non-fuzzy, are generally capable of detecting and preventing intrusion of an entire computer network and not a single endpoint. Anti-virus software is perhaps the closest comparison to the system proposed here. It is intended to prevent exploitation of a single machine. There are varying mechanisms to do so. Most applications are signature-based, meaning they constantly scan the system for known “signatures” of viruses, generally by comparing the hash of a new file or memory segment to a dictionary of known values.

Heuristic detection techniques may also detect some virus binaries which have been modified to evade signature-based detection techniques while still maintaining the integrity of the executable. Some heuristic techniques may be fuzzy by detecting inexact signature matches. However, these mechanisms still rely on detecting known code bases and exploits and are not capable of detecting zero days.

3.Methodology and design

The following requirements were defined for the proposed intrusion detection system based on the authors experience in the filed:

- The model shall be capable of processing the following inputs:
  - Overall CPU usage, measured as a percentage of idle time
  - Overall memory usage, measured as a percentage of unallocated memory
  - Disk I/O, measured in bytes per second
  - Disk I/O, measured in number of operations
  - Number of total network connections currently active
  - Number of new network connections in the last second
  - Number of new network connections in the previous minute
  - Number of new network connections in the previous 5 minutes
  - Type of network connections (inbound/outbound, port, type of service (if known))
  - Number of active processes

- ○ Number of new processes over the last second
- ○ Number of new processes over the last 5 seconds
- ○ Number of new processes over the last 30 seconds
- ○ Number of threads
- ○ Number of new threads over the last second
- ○ Number of new threads over the last 5 seconds
- ○ Number of new threads over the last 30 seconds
- ○ Moving average of number of processes, measured over the previous 15 seconds
- ○ Moving average of number of new processes, measured over the previous 15 seconds
- Confidence results shall be classified into the following buckets:
- ○ 80% or higher: Compromised
- ○ 60-80%: Potential compromise
- ○ 40-60%: Unsure
- ○ 20-40%: Likely not compromised
- ○ 20% or lower: Not compromised
- The model shall indicate clearly to the user whether the system is believed to be compromised and the percent confidence in that result.
- The percent confidence may be displayed graphically.

Only simple tests were required to verify these requirements. For example, a test to verify the readability requirements simply consists of placing the android device at eye level and four feet from the driver, and verifying that the text is readable. The full test plan is as follows:

- Individual membership functions
- ○ Provide a range of inputs to the model for the individual membership functions, covering the entire relevant function width.
- ○ Verify that the membership function provides a valid and expected membership value for each input.
- Combined result
- ○ Provide a range of membership values for each membership function that feeds the combined relationship
- ○ Verify that the provided membership values provide the expected relationship and confidence result
- Classification/reporting
- ○ Provide inputs to the model that have been previously calculated (manually or otherwise outside the model) to provide a 20% or lower confidence rating
- ○ Verify that the system reports a result of “not compromised” and reports the confidence rating
- ○ Repeat these steps for each of the remaining 4 result buckets

A set of data on normal usage was collected to baseline the system. The input set was captured on a 2014 MacBook Pro with a 2.2GHz Intel Core i7 processor, 16GB of RAM and running Mac OS X 10.11.4. CPU, memory and thread usage statistics were captured using the open-source tool top [13], configured to collect system information once per second. Lsof (list open files, which is used in many Unix-like systems to report a list of all open files and the processes that opened them) was used to capture information on active network connections.

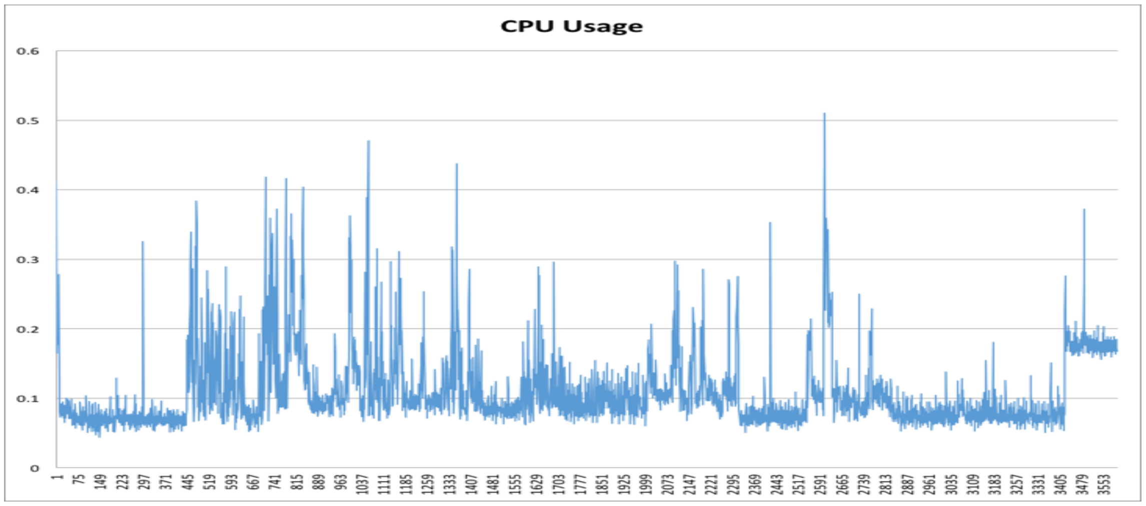


Figure 1 Collected CPU usage

Data was captured on normal usage (web browsing, email, some video streaming, etc.) once per second over one hour. *Figure 1* illustrates the collected CPU usage. *Figure 2* illustrates the network connection data. *Figure 3* shows the memory usage collected data.

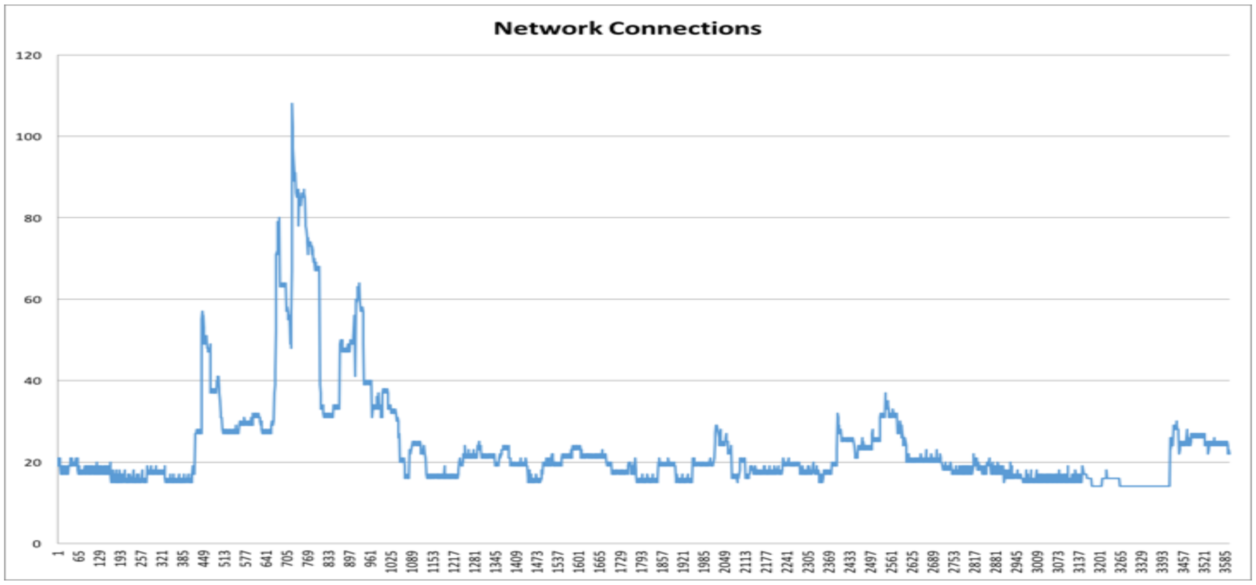


Figure 2 Collected data on number of network connections

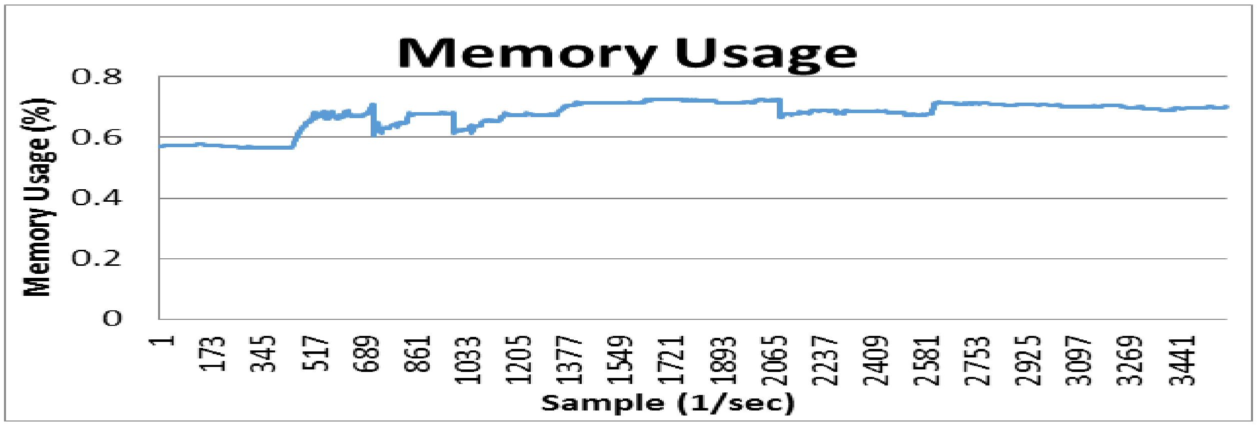


Figure 3 Collected memory usage

The fuzzy inference system to detect exploitation of a system will be designed using system metrics such as CPU, memory usage and network connections. The following are the design steps:

a. Input Selection and Membership Functions

Any number of computer syClRstem measurements may identify a potential compromise. The most obvious to users is CPU usage, memory usage and disk I/O, all of which can quickly cause a PC to appear less responsive, either due to lack of resources or being I/O-bound. The model proposed requirements has defined a wide array of measurements that the system should be capable of monitoring. For this implementation of the proposed system the following subset of five measurements were selected:

- CPU usage
- Memory usage
- Number of active network connections
- Number of new network connections in the previous second
- Number of new threads over the last second

These measurements were chosen based on expert knowledge. An exploit will generally try to invade an existing process and masquerade within a known good process – so monitoring active processes may not always be useful, but monitoring threads could indicate a change to the application. Most exploits will generally try to operate as quickly as possible before being detected, with no regard to existing resource usage-therefore, things like CPU usage and memory usage are useful measurements. Finally, intruders will generally try to reach another system remotely-thus; network connection metrics are highly useful as well.

These inputs were assigned similar membership functions for ease of implementation. Input values were assigned five linguistic values (different classifications): very low, low, moderate, high and very high. Classifications were based on a combination of expert knowledge and some limited data. The resulting membership threshold values are shown in *Table 1*

Table 1 Membership function threshold values

	Very low	**Low**	**Moderate**	**High**	**Very high**
CPU Usage (%)	0-0.2	0.1-0.3	0.2-0.4	0.3-0.5	0.4-1
Memory Usage (%)	0-0.3	0.4-0.6	0.5-0.7	0.6-0.8	0.7-1
# of Network Conns	0-20	10-30	20-40	30-50	40+
# of New Network Conns (1 sec)	0-4	2-6	4-8	6-10	8+
# of New Threads (1 sec)	0-6	3-9	6-12	9-18	12+
Compromise State	Not Comp – 0-0.35	Likely not comp – 0.2-0.5	Unsure – 0.35-0.65	Likely comp – 0.5-0.8	Comp – 0.65-1

b. Model and rule selection and defuzzification

A Mamdani model was selected for this model. The consequence of the rule set is known to be a fuzzy set (varying levels of compromise). The inputs are also fuzzified as they are grouped into classifications (linguistic values) rather than using crisp values.

Rules were then defined for the model based on expert knowledge. A maximum of 7776 (6*6*6*6*6) potential rules could have been defined, given every possible input classification, however not all of these rules are useful. For this project, a set of 15 rules were defined which attempted to cover a range of likely scenarios from low resource usage to high resource usage and from low confidence of compromise to high confidence of compromise. One example includes:

IF CPU Usage $x_1(k)$ is Very High, and
Memory Usage $x_2(k)$ is Very High, and
# of Network Conns $x_3(k)$ is Very High, and
# of New Network Conns $x_4(k)$ is Very High, and
# of New Threads $x_5(k)$ is Very High,
THEN System is COMPROMISED.

Some less detailed rules were defined as follows:

IF CPU Usage $x_1(k)$ is Very High, and
Memory Usage x_2 (k) is Very High, and
# of Network Conns $x_3(k)$ is High,
THEN System is LIKELY COMPROMISED.

The complete sets of rules are as follows:

1. IF (CPU Usage x1(k) is Very Low), AND (Memory Usage x2(k) is Very Low), AND (# of Network Connections x3(k) is Very Low), AND (# of New Network Connections in last 5 seconds x4(k) is Very Low), AND (# of New Threads in last 5 seconds x5(k) is Very Low), THEN (Compromise State y(k) is Not Compromised).; weight = 1
2. IF (CPU Usage x1(k) is Very High), AND (Memory Usage x2(k) is High), AND (# of Network Connections x3(k) is Very High), AND (# of New Network Connections in last 5 seconds x4(k) is Very High), AND
(# of New Threads in last 5 seconds x5(k) is Very High), THEN (Compromise State y(k) is Compromised).; weight = 1
3. IF (CPU Usage x1(k) is Low), AND (Memory Usage x2(k) is Low), AND (# of Network Connections x3(k) is Low), AND (# of New Network Connections in last 5 seconds x4(k) is Low), AND (# of New Threads in last 5 seconds x5(k) is Low), THEN(Compromise State y(k) is Likely Not Compromised).; weight = 1
4. IF (CPU Usage x1(k) is Moderate), AND(Memory Usage x2(k) is Very High), AND (# of Network Connections x3(k) is High), AND(# of New Network Connections in last 5 seconds x4(k) is Moderate), AND (# of New Threads in last 5 seconds x5(k) is Low), THEN (Compromise State y(k) is Likely Compromised).; weight = 0.4
5. IF (CPU Usage x1(k) is Moderate), AND (Memory Usage x2(k) is Very High), AND (# of

Network Connections x3(k) is Moderate), AND(# of New Network Connections in last 5 seconds x4(k) is Low), AND (# of New Threads in last 5 seconds x5(k) is Low), THEN (Compromise State y(k) is Likely Not Compromised).; weight = 1

6. IF (CPU Usage x1(k) is Very High), AND (Memory Usage x2(k) is High), AND (# of Network Connections x3(k) is High), THEN (Compromise State y(k) is Likely Compromised).; weight = 0.8
7. IF (CPU Usage x1(k) is Moderate), AND(Memory Usage x2(k) is Very High), AND (# of Network Connections x3(k) is Low), AND (# of New Network Connections in last 5 seconds x4(k) is Very Low), THEN (Compromise State y(k) is Likely Not Compromised).; weight = 0.8
8. IF (CPU Usage x1(k) is Moderate), AND (Memory Usage x2(k) is Very High), AND (# of New Network Connections in last 5 seconds x4(k) is High), AND (# of New Threads in last 5 seconds x5(k) is High), THEN (Compromise State y(k) is Likely Compromised).; weight = 0.6
9. IF (CPU Usage x1(k) is Moderate), AND (Memory Usage x2(k) is Moderate), AND (# of Network Connections x3(k) is Moderate), AND (# of New Network Connections in last 5 seconds x4(k) is Very High), AND (# of New Threads in last 5 seconds x5(k) is Very High), THEN (CompromiseState y(k) is Likely Compromised).; weight = 0.7
10. IF (CPU Usage x1(k) is Low), AND (Memory Usage x2(k) is Moderate), AND(# of Network Connections x3(k) is Low), AND (# of New Network Connections in last 5 seconds x4(k) is Moderate), AND (# of New Threads in last 5 seconds x5(k) is Low), THEN (Compromise State y(k) is Not Compromised).; weight = 0.7
11. IF (CPU Usage x1(k) is Moderate), AND (Memory Usage x2(k) is Low), AND(# of Network Connections x3(k) is Low), AND (# of New Network Connections in last 5 seconds x4(k) is Low), AND (# of New Threads in last 5 seconds x5(k) is Low), THEN(Compromise State y(k) is Likely Not Compromised).; weight = 0.8
12. IF (CPU Usage x1(k) is Moderate), AND (Memory Usage x2(k) is Moderate), AND (# of Network Connections x3(k) is Moderate), AND(# of New Network Connections in last 5 seconds x4(k) is Moderate), AND(# of New Threads in last 5 seconds x5(k) is Moderate), THEN(Compromise State y(k) is Unsure).; weight = 0.8
13. IF (CPU Usage x1(k) is High), AND (Memory Usage x2(k) is High), AND(# of Network Connections x3(k) is Very Low), AND(# of New Network Connections in last 5 seconds x4(k) is Very Low), AND(# of New Threads in last 5 seconds x5(k) is Very Low), THEN(Compromise State y(k) is Likely Not Compromised).; weight = 0.8
14. IF (CPU Usage x1(k) is Moderate), AND (Memory Usage x2(k) is Moderate), AND(# of Network Connections x3(k) is High), AND (# of New Network Connections in last 5 seconds x4(k) is Very High), AND(# of New Threads in last 5 seconds x5(k) is High), THEN(Compromise State y(k) is Compromised).; weight = 0.6
15. IF (CPU Usage x1(k) is Very Low), AND (Memory Usage x2(k) is Very Low), AND (# of Network Connections x3(k) is High), AND (# of New Network Connections in last 5 seconds x4(k) is Very High), AND(# of New Threads in last 5 seconds x5(k) is High), THEN (Compromise State y(k) is Unsure).; weight = 0.6

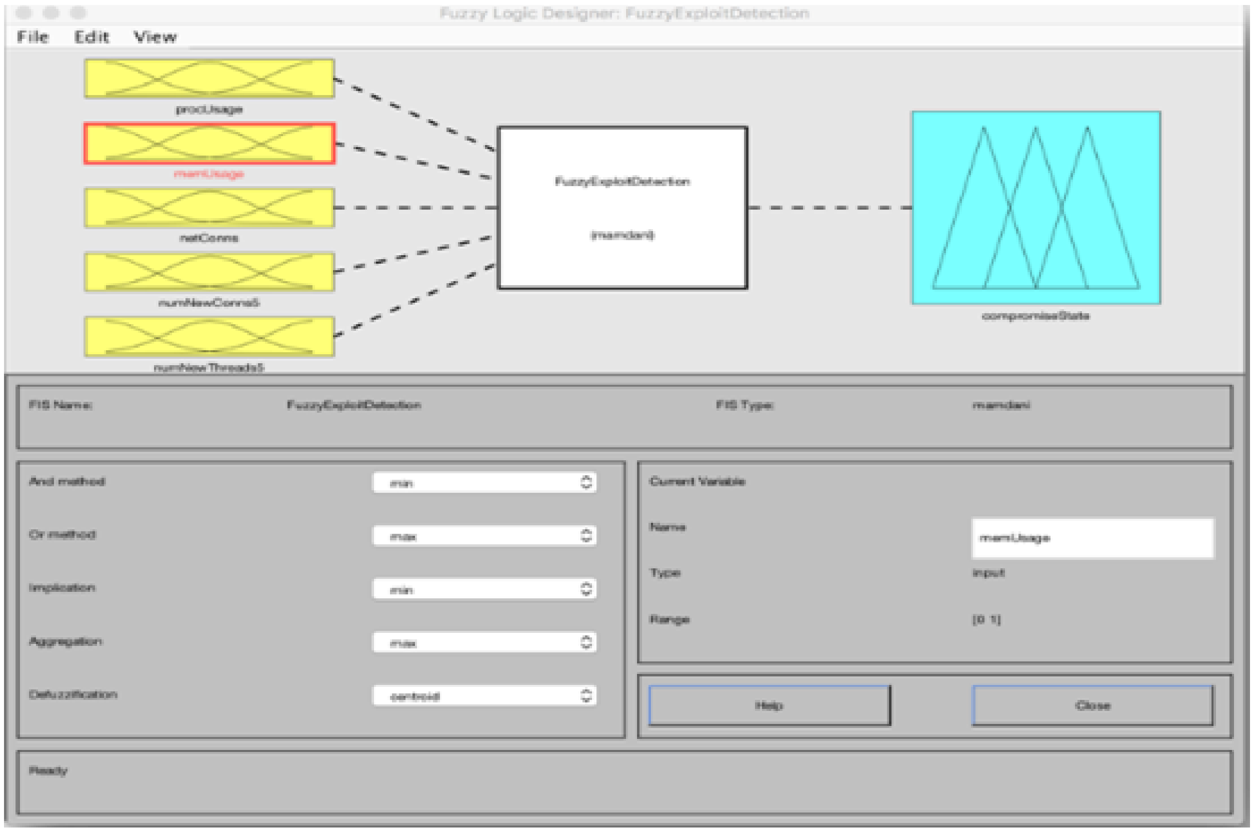
The centroid method was used for defuzzification.

4.Implementation

The design was implemented as a model in MATLAB using the fuzzy logic toolbox. *Figure 4* demonstrates the fuzzy inference system (FIS) generated in the fuzzy logic designer. The five inputs were each defined, as well as the output (compromiseState). Min and max were used to define the AND and OR calculation methods, respectively. Likewise, the min method is used to evaluate implications, and the max method used to aggregate the rules. As noted previously, the centroid method was used as the defuzzification technique.

Pi, sigmoidal and Gaussian membership functions were used for the inputs. They were designed to center around the middle of the previously defined threshold values. *Figure 5* demonstrates the membership function for CPU usage. Triangular and trapezoidal membership functions were used to capture the output and were also centered on the threshold values defined as shown in *Figure 6*.

Rules were defined based and were assigned weights based on expert knowledge in confidence of the rules. Weights ranged from 0.4 to 1-rules with less confidence in consequents were weighted lower and rules with complete confidence in consequents were weighted as a 1.

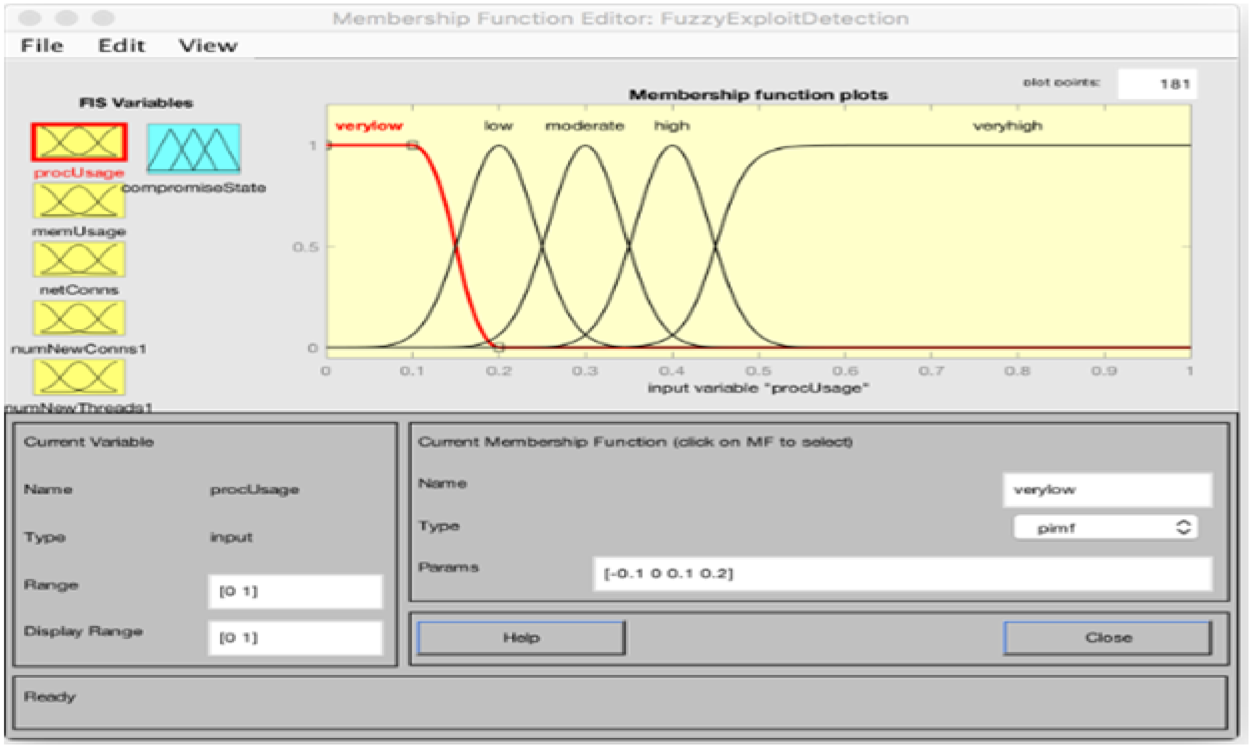


FIS Name: FuzzyExploitDetection
FIS Type: mamdani

And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid

Current Variable
Name: memUsage
Type: input
Range: [0 1]

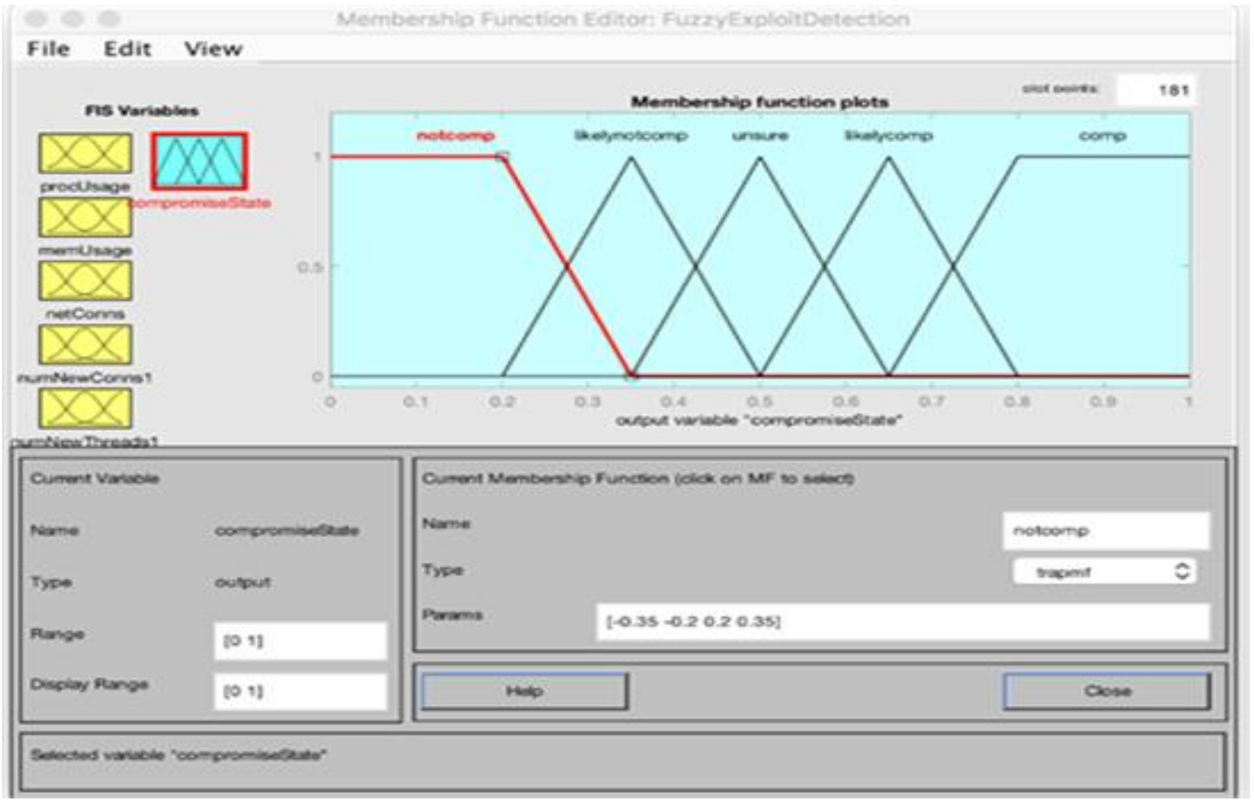
Figure 4 MATLAB fuzzy inference system implementation



Current Variable
Name: procUsage
Type: input
Range: [0 1]
Display Range: [0 1]

Current Membership Function (click on MF to select)
Name: verylow
Type: pimf
Params: [-0.1 0 0.1 0.2]

Figure 5 Membership function for CPU usage



Current Variable

Name compromiseState

Type output

Range [0 1]

Display Range [0 1]

Current Membership Function (click on MF to select)

Name notcomp

Type trapmf

Params [-0.35 -0.2 0.2 0.35]

Selected variable "compromiseState"

Figure 6 Compromise state (Output) membership function

5.Testing and results

Some sample points from the collected data were used to test the system. Points were selected that indicated various periods of usage within the system. While the system was not actually compromised during the period of the test, the points were selected to test the functionality of the model. A future test should include actual compromise of a controlled environment, and demonstrate the true effectiveness of the model. *Table 2* shows the used test points.

Table 2 Test data and results

CPU usage	**MEM usage**	**# of network conns**	**# of new network conns**	**# of new threads**	**Comp. state**
0.5103	0.6921	24	0	49	Potential (0.649)
0.1172	0.7087	108	41	0	Compromised (0.83)
0.0628	0.7276	19	0	4	Likely not compromised (0.35)
0.2779	0.5753	19	0	325	Likely not compromised (0.377)
0.0440	0.5765	19	2	0	Likely not compromised (0.276)
0.3840	0.6271	38	0	34	Potential (0.648)
0.3258	0.5682	17	0	289	Likely not compromised (0.354)
0.0607	0.6893	19	2	1	Likely not compromised (0.323)
0.3507	0.6229	21	1	27	Likely not compromised (0.368)
0.0786	0.6765	49	16	0	Potential (0.646)

The test results have shown some interesting feedback. A large number of network connections become tied to a more likely compromise state. This could be indicative of certain exploits such as a botnet, which may be attempting numerous outbound tries to connect. High memory usage is not necessarily indicative of compromise, as shown by the output – the PC may simply be caching information in memory from the hard disk. High CPU usage with a large number of new threads may indicate a compromised state – if a new process (say, an exploit) is trying to start up and perform CPU-intensive operations. A large number of new threads on its own, however, may not indicate an exploit. This last test example was a likely event during a video streaming operation completed. The resulting surface of this relationship is shown in *Figure 7*. The surface shown in *Figure 8* demonstrates the relationship between the number of network connections and the number of new threads, generally showing a positive correlation.

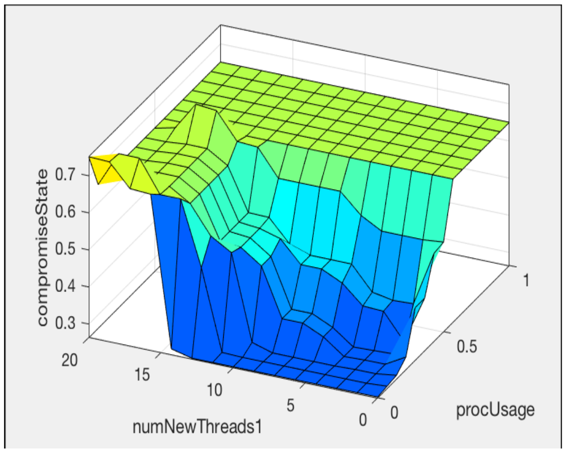


Figure 7 Surface of CPU usage and new threads

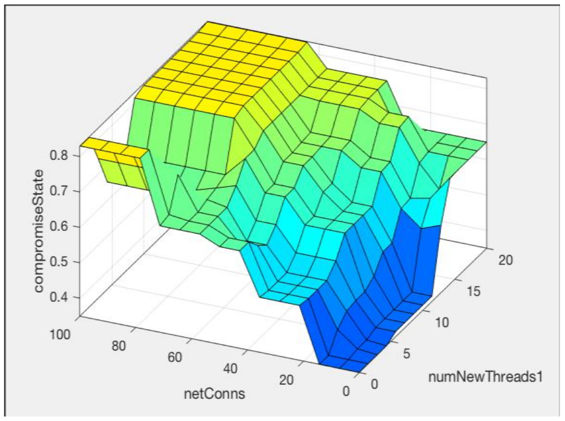


Figure 8 Surface of network connections and new threads

6.Conclusion

A fuzzy inference system has been proposed to detect computer system exploits based on system metrics. The initial model was defined with 5 inputs and a single output. Future versions of the model may be capable of capturing additional inputs such as disk I/O. The design was tested and provided reasonable results, but further evaluation is needed, especially around fine tuning the impact of network connections. Finally, future improvements to the design are proposed which include the following:

- A number of next steps can be taken to further refine the design and make it more accurate. The first would be to evaluate a number of different systems. This should range across different levels of hardware, different operating systems, different system configurations, and more. This model only evaluates the overall system. The system can be subject to a number of variables, including processes such as image/video processing that normally use large amounts of memory and CPU time. Individual processes could be examined and carefully monitored to check for unusual changes in CPU usage, memory usage, network handles are used by a given process, etc.
- The system could be further fined tuned to utilize different membership functions and thresholds for each process. This would be able to more accurately monitor processes that typically use large amounts of system resources. It could also be used to specifically target certain processes prone to exploitation such as browsers, word processing and rich document viewing applications, email applications and others.
- Analysis is also needed for a lightweight implementation of this type of system. A fuzzy exploit detection monitor would need to be consistently running in the background. However, it cannot impact the normal operation of the system as this would create an undesirable user experience.
- Finally, the system could also be improved through the use of neural networks. A periodic learning state could be used to gather information on the system’s typical load for a given user via unsupervised learning. A pre-production supervised learning exercise could also be used to pre-train the system. The system would also need to be somewhat self-monitoring to detect the intrusion of its own processes.

Acknowledgment
None.

Conflicts of interest
The authors have no conflicts of interest to declare.

References

[1] Chen WW. Statistical methods in computer security. CRC Press; 2004.

[2] Jesdanun A. School prank starts 25 years of security woes. http://www.nbcnews.com/id/20534084/#.V5bI8GXZpg1. Accessed 4 April 2016.

[3] Anderson JP. Computer security threat monitoring and surveillance. Technical Report, James P. Anderson Company, Fort Washington, Pennsylvania; 1980.

[4] Denning DE. An intrusion-detection model. IEEE Transactions on Software Engineering. 1987; SE-13(2):222-32.

[5] Vaccaro HS, Liepins GE. Detection of anomalous computer session activity. In IEEE symposium on security and privacy, proceedings 1989 (pp. 280-9). IEEE.

[6] Abadeh MS, Habibi J, Lucas C. Intrusion detection using a fuzzy genetics-based learning algorithm. Journal of Network and Computer Applications. 2007; 30(1):414-28.

[7] Wang G, Hao J, Ma J, Huang L. A new approach to intrusion detection using artificial neural networks and fuzzy clustering. Expert Systems with Applications. 2010; 37(9):6225-32.

[8] Mkuzangwe NN, Nelwamondo FV. A fuzzy logic based network intrusion detection system for predicting the TCP SYN flooding attack. In Asian conference on intelligent information and database systems 2017 (pp. 14-22). Springer, Cham.

[9] Shanmugavadivu R, Nagarajan N. Network intrusion detection system using fuzzy logic. Indian Journal of Computer Science and Engineering. 2011; 2(1):101-11.

[10] Kudłacik P, Porwik P, Wesołowski T. Fuzzy approach for intrusion detection based on user's commands. Soft Computing. 2016; 20(7):2705-19.

[11] Azad C, Jha VK. Fuzzy min–max neural network and particle swarm optimization based intrusion detection system. Microsystem Technologies. 2017; 23(4):907-18.

[12] Ramakrishnan S, Devaraju S. Attack's feature selection-based network intrusion detection system using fuzzy control language. International Journal of Fuzzy Systems. 2017; 19(2):316-28.

[13] http://www.unixtop.org. Accessed 4 April 2016.

Dr. Adnan Shaout is a full professor and a Fulbright Scholar in the Computer Science Department at the Electrical and Computer Engineering Department at the University of Michigan-Dearborn. At present, he teaches courses in AI, Embedded Systems, Software Engineering, Computer Architecture, Cloud Computing, Fuzzy Logic and Engineering Applications and Computer Hardware Design. His current research is in applications of software engineering methods, cloud computing, embedded systems, fuzzy systems, real time systems and artificial intelligence. Dr. Shaout has more than 34 years of experience in teaching and conducting research in the Computer Science, Electrical and Computer Engineering fields at Syracuse University and the University of Michigan - Dearborn. Dr. Shaout has published over 210 papers in topics related to Computer Science, Electrical and Computer Engineering fields. Dr. Shaout has obtained his B.S.c, M.S. and Ph.D. in Computer Engineering from Syracuse University, Syracuse, NY, in 1982, 1983, 1987, respectively.
Email: shaout@umich.edu

Cameron Smyth is a cyber-security engineer at Ford Motor Company. He provides recommendations on securing new features and defines security solutions for Ford's connectivity, mobility and autonomous vehicle platforms. He has filed patents on secure communication mechanisms and holds two patents from a prior role on systems to detect and alert drivers of vehicle clearance concerns. Cameron obtained his M.S.E. in Computer Engineering in 2017 and B.S.E.s in Computer Engineering and Electrical Engineering in 2012 from the University of Michigan – Dearborn.