Optimizing Anomaly Detector Deployment under Evolutionary Black-box Vulnerability Testing

Hilmi Güneş Kayacık, Nur Zincir-Heywood, Malcolm Heywood, Stefan Burschka

Abstract— **This work focuses on testing anomaly detectors from the perspective of a Multi-objective Evolutionary Exploit Generator (EEG). Such a framework provides users of anomaly detection systems two capabilities. Firstly, no knowledge of protected data structures need to be assumed (i.e. the detector is a black-box), where the time, knowledge and availability of tools to perform such an analysis might not be generally available. Secondly, the evolved exploits are then able to demonstrate weaknesses in the ensuing detector parameterization. Therefore, the system administrator can identify the suitable parameters for the effective operation of the detector. EEG is employed against two second generation anomaly detectors, namely pH and pH with schema mask, on four UNIX applications in order to perform a vulnerability assessment and make a comparison between the two detectors.**

I. Introduction

BUFFER OVERFLOW ATTACKS represent one of the most effective and widespread examples of a host based attack. Intrusion detection techniques for recognizing (and therefore initiating a defence against) buffer overflow attacks generally fall into one of two basic forms: misuse or anomaly. The misuse paradigm focuses on recognizing the attack, whereas the anomaly paradigm concentrates on modelling 'normal' behaviour and flags everything else as 'anomalous.' Signature detectors represent a widely utilized example of the former (e.g. Snort [1]), whereas Stide was an early example of the latter [2]. Given the ease with which current examples of misuse detection can be evaded [3], in this paper, we concentrate on the case of anomaly detection.

The previous work of Kayacık *et al.* [4] on evolving attacks has shown that it is possible to evolve exploits against the original Stide anomaly detector when feedback from the detector is limited to the (publicly available) anomaly rate [4]. Conversely, the norm in the literature is to assume access to privileged data structures of the detector while trying to evade it. Indeed, such an approach simplifies the search for candidate exploits [5], [6], [7]. Moreover, such works only consider the problem of crafting an exploit, where this is only half of a valid attack. A complete attack requires the combination of a preamble and exploit [8]. The role of a preamble is to set up the vulnerability such that the exploit can be launched. Specifically, anomaly detectors would receive the preamble component of an attack before seeing the exploit. Given that the detector is attempting to compare behavioural properties of candidate instruction sequences with a concept of normal behaviour, the preamble places additional limits on the content of exploits that are likely to evade detection.

Unlike the original Stide anomaly detector, the recent process Homeostasis (pH) model of anomaly detection both detects and instigates counter measures against suspicious processes [9]. Thus, processes with above normal anomaly rates receive a corresponding exponentially weighted delay. Detection is performed relative to sequences of system calls made by an application. Unlike the earlier Stide model of anomaly detection pH does not solely rely on a single sliding window (of process system calls) to establish a model of normal behaviour, but instead abstracts information from the sliding window of system call (SC) information into a tuple: $< SC(t), SC(t-1), relative\_position(SC(t-1)) >$. Somayaji [9] established that this 'look-ahead' method was more efficient to store and deploy and was quicker to train to a normal profile than Stide. Moreover, the resulting anomaly detector was shown to perform significantly better than Stide [2]. A recent pH variant [10] employed schema masks to increase the depth of the detector sliding window without increasing the computational complexity of detection. This is achieved by taking values from the pre-determined locations of the sliding window.

The goal of this work is to perform a vulnerability assessment of the original pH and the pH with schema mask anomaly detectors under multiple applications using a black-box approach and to compare the effectiveness of both detectors (i.e. whether incorporating a schema mask makes it more difficult to evolve attacks). Specifically, the only detector information used to 'guide' the search for better exploits is the alarm rate, where this is public information available to a user. Conversely, in order to establish better criteria for evolutionary exploit generation (EEG) we make use of a Pareto multi-objective fitness function. We demonstrate that such a framework presents users with a wider characterization of the scope of potential vulnerabilities for a given anomaly detector, as well as quantifying the impact of the detector parameterization. In common with all previous work on automating exploit generation we assume the availability of the preamble [5], [6], [7], but explicitly include the impact of this within the evaluation, so that exploits are identified within context.

II. Methodology

As indicated in the Introduction, we are interested in evolving buffer overflow attacks against pH anomaly detector and its variant – pH with schema mask – where the detectors have

Hilmi Güneş Kayacık, Nur Zincir-Heywood and Malcolm Heywood are with the Faculty of Computer Science, Dalhousie University, Canada (email: {kayacik, zincir, mheywood}@cs.dal.ca). Stefan Burschka is with Swisscom Innovations Inc., Switzerland (email: Stefan.Burschka@swisscom.com).

the capacity to delay the process associated with suspicious system call sequences. Moreover, given that an attack is composed of a preamble and an exploit components [8], it would also be useful to reward the EEG for reducing anomaly rates associated with the attack as a whole as opposed to the exploit alone. Given the multiple criteria associated with this task it is reasonable to assume a Pareto framework for designing our exploit generator [4]. Before we do so, however, we first need to establish the basic components of the evolutionary model; in particular the representation and model of deployment. To this end we assume the framework established by Kayacık *et al.* [4] and summarize this in Section II-A. The design of the multi-criteria fitness function is considered relative to the this framework in Section II-B.

A. Basic Evolutionary Exploit Generator

The basic framework for EEG automates the design of attacks using Genetic Programming (GP). The GP paradigm differs from most machine learning methodologies in that a 'population' of candidate solutions are maintained concurrently throughout the search process [11]. Each candidate solution, or individual, takes the form of a program, which is represented, in the case of this application, as a sequence of system calls. Furthermore, programs are variable length and consist of instructions. The specific representation utilized in the GP employed in the EEG defines instructions as integers where each integer corresponds to a system call.

This work expands upon the basic EEG that we have developed before [4]. The basic framework for EEG under a black-box detector assumption was considered to comprise of two basic steps [4]: identification of a representative set of system calls, and design of the fitness function, as follows:

System Calls: It is necessary to *a priori* specify a set of instructions (system calls) from which candidate exploits are built. The approach adopted by Kayacık *et al.* was to first run the vulnerable application (*not* the anomaly detector) under normal operation and then review the most frequently executed system calls during execution. That is to say, a diagnostic tool – in this case Strace[1] – is deployed to establish the most frequent 20 system calls of the application (accounts for upwards of 90 percent of the system call utilization). It is these system calls that are used as the instruction set of the evolutionary exploit generator. The natural assumption behind this is that attacks composed from instructions representing 'normal' behaviour will be more difficult to detect than attacks composed from instructions outside of normal behaviour. In effect the same principle guiding the approach of an anomaly detector in general will be assumed in establishing the instruction set of the exploit. This does not break the black-box assumption as use is made of public information associated with the application, not the detector.

Fitness Function: Two basic criteria were previously identified for guiding evolution towards an effective attack: criteria for a valid attack, and quality of the attack. Criteria for a valid attack assumed that the basic goal of a buffer overflow exploit were to: open a file, write to the file, and close the file. The file in question was the password file, where the arguments to the open-write-close sequence established the specifics of the target file with the objective of creating a super-user account. Naturally, the sequence of these three operations is significant. Thus, individuals were rewarded for both the instructions executed and the order of execution. In the case of exploit quality, use was made of the anomaly rate returned by the detector. Thus if the detector considered the instruction sequence anomalous the ensuing penalty would be greater. Again we note that this is not making use of any privileged detector data structures – such as the data base of normal behaviours. Moreover, the junk code or introns of an individual may now actually perform a useful purpose, such as obfuscating the true intent of the process. The goal is to make the obfuscating material mimic normal behaviour as measured by the detector.

The search process progresses through the iterative application of a selection operator, evaluation of performance associated with the subset of individuals targeted by the selection operator, and application of search operators. Specifically, the selection operator in this work takes the form of a steady state tournament. This means that an even number of individuals (4 in this work) is selected from the population of (many more) individuals with uniform probability. Performance (fitness) of this subset of individuals is evaluated, ranking the individuals participating in the tournament relative to each other. Search operators are then applied to the better performing half of the tournament, resulting in (2) children. The children overwrite the individuals of the worst half of the tournament, taking their place in the original population. Such a scheme is inherently elitist with the best individuals always surviving. Individuals are defined using a variable length format, thus population initialization creates individuals with program varying program lengths. Search operators take three forms: crossover, instruction-wise mutation, and instruction swap. The specific details of each operator are as follows, however, all search operators are applied stochastically relative to a predefined probability of application, Table I.

Crossover: Crossover operator provides a scheme for investigating instruction sequences that currently exist in the population, but in different contexts. Crossover operator selects, with uniform probability, separate crossover points on each parent. Therefore the children can have different lengths than their parents.

Swap: The swap operator provides the opportunity to investigate the significance of different instruction orders within the same individual. The operator is applied to a single individual, selecting two instructions with uniform probability, and interchanging their position.

Instruction-wise mutation: Mutation operator provides a way to introduce new sequences to the individual. Mutation is applied instruction-wise, that is to say, each instruction is tested independently for modification. If the test returns

[1]Strace can be downloaded from http://sourceforge.net/projects/strace/.

true then the instruction is replaced with an alternative instruction, from a predefined list of instructions. Moreover, we also let the probability of applying the mutation operator decay linearly with tournament count [4]; thus lowering the likelihood of introducing instructions that are not currently in the population as the population evolves. This effectively places more emphasis on the crossover operator as evolution progresses, thus reinforcing the reuse of system call sequences that where earlier demonstrated to minimize detection.

TABLE I
EEG TRAINING PARAMETERS.

Parameter	Setting
Crossover	0.9 probability
Mutation	0.01 probability, linearly decreasing to 0 over the tournament limit
Swap	Instruction swap within an individual with 0.5 probability
Selection	Tournament of 4 individuals
Stop criteria	100,000 tournaments or until the convergence criteria is met
Convergence criteria	If the Pareto ranks remain unchanged over 10 tournaments
Population	500 individuals with instruction selection probability proportional to the percentage of the instruction in the normal use cases
Program length	Initialized over 240 system calls, maximum 1000 system calls
Training time	Approximately 2 days
Number of runs	50

In the following we assume the same process for identifying the subset of system calls to compose attacks. However, the criteria for designing the fitness function will now assume an Evolutionary Multi-criteria Optimization (EMO) model in order to address the increased functionality apparent in the new anomaly detectors such as pH, but without referring to knowledge of the detectors operational features.

B. Evolutionary Multi-criteria Exploit Generator

In assuming an EMO basis for evolving exploits we naturally make use of the method of Pareto ranking to combine multiple objectives into a single 'scalar' fitness function [12]. Specifically, a process of pairwise comparison is used to establish to what degree each individual is dominated by other individuals in the population. Thus, one individual (*A*) is said to dominate another (*B*) iff *A* is as good as *B* on all objectives and better than *B* on at least one. However, a wide range of algorithms have been proposed to support such a rank based fitness function [13]. In this work we assume the PCGA framework of Kumar and Rockett [14]. Such a scheme avoids the need to support explicit measures of similarity, where such schemes are generally employed to encourage diversity and/ or measure similarity in the population. Instead diversity is established care of a steady-state style of tournament. Specifically, after the initial population is ranked, parents are chosen using proportional selection and the ensuing children ranked. The rank of the children is compared to that of the worst ranked individuals

1) Count = 0;
2) IF (sequence contains *open*("/etc/passwd)) THEN (Count++)
3) IF (sequence contains *write*("toor::0:0:root:/root:/bin/bash")) THEN (Count++)
4) IF (sequence contains *close*("/etc/passwd")) THEN (Count++)
5) IF ('*open*' precedes '*write*') THEN (Count++)
6) IF ('*write*' precedes '*close*') THEN (Count++)

Fig. 1. Fitness objective quantifying the exploit functionality.

in the population. If the children have a better rank, then they overwrite the (worst ranked) population members. Use of such a EMO model is necessary in this work as it is not possible to establish distance information relative to the original problem domain. That is to say, popular algorithms such as NSGA-II and SPEA all make use of Euclidean based distance functions when building the diversity metric (see for example [13]). In this work the representation takes the form of system call sequences as opposed to points in multi-dimensional space, thus precluding the application of such distance metrics.

Specific objectives measured to establish fitness of each individual under the EMO model take the form of the following three generic objectives:

1) Attack Success: As established in Section II-A the basic functionality of the attack is described in terms of an 'open-write-close' sequence of system calls, with reward established for each of the three instructions and the relevant order. Relevant arguments are also evolved, although pH does not monitor them. A behavioural success function gives a maximum of 5 points the correct 'open-write-close' behaviour, Figure 1.
2) Anomaly Rate: Again as established in Section II-A, the detector anomaly rate represents the principle metric for qualifying the likely intent of a system call sequence; a would be attacker naturally wishes to minimize the anomaly rate of the detector. Again, no inside knowledge is necessary as detectors provide alarm rates as part of their normal mode of operation. Moreover, as indicated in the introduction, the attacker also needs to minimize the anomaly rate of the exploit. Provided that the preamble is not highly anomalous, relatively low attack anomaly rates can be accomplished by utilizing very concise exploits. However this is not always the case, thus we evolve exploit with preamble appended to facilitate identification of the most appropriate content.
3) Attack Length: Attack length is not an immediate concern for the attacker; longer attacks potentially provide more obfuscation. However, attack length appears as an objective to encourage evolution to perform a wider search for solutions i.e., short solutions will be included as well as longer ones under the Pareto methodology.

III. Configuration of the Detectors and the Applications

This work evaluates pH and pH with schema mask (pHsm) detectors on four UNIX applications, therefore detector and application configurations are discussed in this section. EEG is employed on each detector, application pair in order to assess the effectiveness of the detectors on different applications.

A. Configuration of the pH Detector

As previously discussed pH is a second generation anomaly detector in which specific instances of the detector are associated with each application. During training a sliding window is employed over the training set in order to establish the normal behaviour of the application in terms of a three dimensional matrix with dimensions: (1) current system call; (2) previous system call in the sliding window; (3) location of the previous system call in the sliding window. During testing, the same sliding window is employed on the test data (candidate exploits in this case). If a given sliding window sequence produced a look ahead pair that is not in the normal database, a mismatch is recorded. Given a pre-specified window size and system call trace length, the anomaly rate for the trace is calculated by dividing the number of mismatches by the total number of look ahead pairs.

Moreover, the delay property used by pH to penalize suspicious processes is calculated as an exponential function of locality frame count; where locality frame (LF) count aims to identify clusters of anomalies. Specifically, pH simply maintains a count of how many of the past LF (128 by default [9]) system calls were anomalous. Process delays can substantially delay the execution of a program when a cluster of anomalies is observed. In the following experiments, the default pH training parameters were employed, Table II, where the use of such defaults often represent the initial deployment option.

TABLE II
pH Configuration.

Parameter	Setting
Look ahead pair window size	9
Locality frame window size	128
Delay factor	1
Suspend execve after	10 anomalies
Suspend execve duration	2 days
Anomaly limit	30
'Tolerize' limit	12

B. Configuration of the pH with Schema Mask Detector

Inoue and Somayaji [10] proposed an improvement to pH based on the random schema mask concept. Their main motivation was the observation that longer windows improve the detection rates hence there exists a potential to increase the difficulty of generating evasion attacks against pH (and indirectly Stide and variants).

In pH with schema mask (pHsm), a longer sliding window (usually 20 [10]) is maintained and a number of taps (generally 9 which is also the sliding window size of original pH [9]) are taken from sliding window. The locations of the taps are determined randomly before training and this location information constitute the schema mask. Incorporating a schema mask to pH enables pH to monitor a longer time window. Furthermore, it provides an additional detector parameter that the attacker needs to consider. In addition to utilizing the pH configuration parameters detailed in Table II (except the look ahead pair window size), pHsm employs the configuration parameters detailed in Table III.

TABLE III
pH with schema mask (pHsm) configuration parameters

Parameter	Setting
Look ahead pair window size	20
Number of taps taken from the sliding window	9
Tap locations	Determined before training

C. Application Configurations

In the following experiments, four Linux applications are tested: Traceroute, Samba and Restore all of which have known and documented vulnerabilities. These are also the vulnerable applications most frequently used in the attack automation literature [5] [6] [15]. The Traceroute vulnerability can be exploited locally whereas and Samba vulnerability can be exploited remotely. For each application, we define normal use cases, which represent the scenarios of legitimate use and establish the behavioural profile in pH and pHsm.

1) Traceroute Configuration: Traceroute is a network diagnosis tool, which is used to determine the routing path between a source and a destination by sending a set of control packets to the destination with increasing time-to-live values. A typical use of traceroute involves providing the destination IP, whereas the application returns information on the route taken between source and destination. Redhat 6.2 is shipped with Traceroute version 1.4a5, where this is susceptible to a local buffer overflow exploit that provides a local user with super-user access [16]. The attack takes advantage of vulnerability in malloc chunk, and then uses a debugger to determine the correct return address to take control of the program. In order to establish traceroute behaviour under normal conditions we developed five use cases, Table IV, where we also list the number of system calls made by the application (the most frequent 90 percent of which establish the instruction set of the evolutionary exploit generator).

2) Restore Configuration: Restore is a component of UNIX backup functionality, which restores the file system image taken by the dump command. Files or directories can be restored from full or incremental backups. Restore version 0.4b15 on Red Hat 6.2 is vulnerable to an environment variable attack where the attacker modifies the path of an executable and runs restore. This results in executing an

TABLE IV
NORMAL USE CASES OF TRACEROUTE APPLICATION.

Use case	System calls
1. Target a remote server	736
2. Target a local server	260
3. Target a non-existent host	153
4. Target a local host	142
5. Help screen	24

arbitrary command with super-user privileges, which leads to a root compromise. In the published attack [17], attacker spawns a root shell. Table V summarizes five normal use cases that are developed for restore.

TABLE V
NORMAL USE CASES OF RESTORE APPLICATION.

Use case	System calls
1. Restore a small file system dump from a full backup.	2,256
2. Restore a small file system dump from an incremental backup.	1,027
3. Restore a large file system dump from a full backup.	167,207
4. Restore a large file system dump from an incremental backup.	68,185
5. Help screen	53

3) Samba Configuration: The Samba suite provides printer and file sharing for Windows clients and can run on most UNIX variants. Samba sets up printer and network shares that appear as disks and printers under a Windows operating system. Redhat 9.0 is shipped with Samba suite version 2.2.7a, which has a vulnerability [18] that can be exploited over the network to gain super-user privileges. The buffer overflow occurs when a Samba service tries to copy user supplied data into a static buffer without checking. The published attack binds a root shell to a network port. Table VI summarizes the six normal use cases employed to establish the normal behavior on Samba. To best of our knowledge, Samba has not been employed in the previous work for automated attack generation.

TABLE VI
NORMAL USE CASES OF SAMBA APPLICATION.

Use case	System calls
1. Mount a samba share successfully	1,156
2. Invalid password while mounting samba share	680
3. Unmount a samba share successfully	186
4. Find and edit a remote file (ls-cs-ls-pico)	254
5. Find and copy a 38MB remote file to a local directory (ls-cs-cp)	65,648
6. Change samba password remotely	1,527

IV. PERFORMANCE EVALUATION

Each run of the Evolutionary Exploit Generator may provide up to 500 exploits (a total of 50 runs are performed per application/ detector). In order to summarize the following analysis we therefore concentrate on the non-dominated performance characteristics as captured post training. Specifically, this will take the form of three properties: anomaly, sequence length, and delay. Thus a user may select an attack post training on the basis of an individual characterized by one of the three performance properties. By way of a baseline we also include the performance of the original attack. To this end, we downloaded the original attacks from the Security focus website and deployed them against pH and pH with a schema mask (pHsm) as independently configured on each of the four vulnerable applications as detailed in Section III-C.

A. Attacks against pH

Table VII details the characteristics of the non-dominated attacks under the Traceroute application. The last row details the characteristics of the original attack. It is apparent that the anomaly rate of the overall attack can be significantly reduced. Moreover, the attack that minimizes the overall attack anomaly rate also minimizes the exploit anomaly rate. This is expected since the attack and the exploit anomaly rate are correlated. In terms of attack and exploit delays, the attack that achieves the least delay achieves this by deploying a short exploit, hence reducing the delay from 6.39 e+06 to 0.55 seconds. Naturally, this is only achieved while doubling the anomaly rate of the attack (from 16.29% to 30.91%). In this case, the attacker may choose an attack that deploys in seconds over an attack that deploys in days, although this causes an increase in the anomaly rate of the attack.

Table VIII details the attack characteristics of the non dominated solutions and the original attacks on the Restore application. Although the evolved exploits achieve anomaly rates as low as 0.1% for the exploit, the resulting attacks (preamble + exploit) produce approximately 48% anomaly rates. This is due to the fact that the preamble for Restore is fairly large and anomalous, which in turn increases the anomaly rate of the attack.

In case of a long and anomalous preamble, EEG increases the length of the exploit to reduce the anomaly rate. Since the anomaly rate is continuously calculated as the attack progresses, appending a low or zero anomaly exploit will reduce the total anomaly rate in time. This shows that various factors such as the vulnerable buffer size, the target detector and the nature of the exploit may introduce new objectives and constraints that the attacker needs to consider. By employing EEG, intrusion detection researchers can investigate how different factors affect the success of the evasion attacks. In turn, the detectors can be improved accordingly to be sensitive to attack attributes beyond the anomaly rate.

Table IX details the attack characteristics of the non-dominated solutions along with the original attack on the Samba application. Similar to results on Traceroute and Restore, the anomaly rates of the evolved attacks are significantly lower than the original exploit. Furthermore the exploit anomaly rates of the evolved attacks are low, whereas the exploit anomaly rate of the original exploit are much higher. This is interesting given the relatively large duration

TABLE VII

NON-DOMINATED CHARACTERISTICS OF ATTACKS FOR TRACEROUTE APPLICATION AGAINST pH.

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	18.29	11.71	118	6.39 e+06	1.11
Delay	30.91	100	9	0.55	0.02
Length	65.11	66.77	1,000	3.83 e+28	3.75 e+28
Original	66.27	73.91	261	4.39 e+35	4.39 e+35

TABLE VIII

NON-DOMINATED CHARACTERISTICS OF ATTACKS FOR RESTORE APPLICATION AGAINST pH.

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	48.57	0.1	1000	1.90 e+38	9.94
Delay	48.69	0.1	998	1.90 e+38	9.92
Length	48.57	0.1	1,000	1.90 e+38	9.94
Original	87.49	90.7	3,029	1.85 e+39	1.66 e+39

of the preamble (3,868 system calls, as opposed to 83 in Traceroute). Indeed, the large preamble dominates the delay property associated with attacks on Samba, whereas the exploit contribution to the delay is very small.

B. Attacks against pHsm

Table X details the attack characteristics of the non-dominated solutions along with the original attack on the Traceroute application against pHsm. Similar to the attacks evolved against pH, the anomaly rate is lower than the original attack. Furthermore, a comparison with Table VII indicates that the the delays are longer when Traceroute is protected with pHsm, however, where the delay increases from 6.39 e+06 seconds to 2.68 e+07 seconds.

Table XI details the attack characteristics of the non-dominated solutions along with the original attack on the Restore application against pHsm. Although the anomaly rates and the delays of the evolved exploits are fairly low the resulting attacks produce anomaly rates due to the large and anomalous preamble. Given that the preamble encapsulates the set of system calls executed during the break-in, the attacker has limited control of the application and may not be able to prevent preamble from executing the anomalous system calls [4]. On the other hand, during the exploit phase, attacker has the full control of the application, therefore can attempt to avoid the execution of anomalous system calls.

Table XII details the attack characteristics of the non-dominated solutions along with the original attack on the Samba application against pHsm. Anomaly rates and delays of the non-dominated solutions are fairly low, however as discussed in Section IV-A, the anomalous preamble increases anomaly rates to above 7% and delays to above e+20.

What happens when the schema mask on the target detector is unknown?: Attack characteristics detailed in Tables X, XI and XII are obtained when the evolved attacks are tested against pHsm configurations that employ the same schema mask as the attacker. This scenario implies that the attacker knows the schema mask that the target detector employs. In contrast, if the attacker does not know the schema mask that the target detector employs, he/she would have to employ a random schema mask. To test this scenario, the evolved attacks are tested against pHsm configurations that employ schema masks that are different from the schema mask that the attacker employs.

By considering these two scenarios, the goal is to identify the main characteristic that makes the pH with schema mask more resistant against attacks. In other words, is it the longer sliding windows or the schema mask that is unknown to the attacker that makes pHsm resistant against attacks?

Table XIII details the attack characteristics of the non-dominated solutions along with the original attack on the Traceroute application against pHsm when the detector employs a different schema mask. Results indicate that it is more difficult to evolve attacks against pHsm if the schema mask of the detector is not known. On the other hand, the attack that optimizes delay succeeds in deploying with short delays even though the schema mask is different.

Table XIV details the attack characteristics of the non-dominated solutions along with the original attack on the Restore application against pHsm when the detector employs a different schema mask. In the case of Restore application, possessing the schema mask of the detector does not seem to change the anomaly rates since the anomaly rate of the evolved attacks remain at 54-57% (compared with the attack characteristics in Table XI).

Table XV details the attack characteristics of the non-dominated solutions along with the original attack on the Samba application against pHsm when the detector employs a different schema mask. If the evolved attacks that optimize anomaly rate and length is deployed against the pHsm with a different schema mask, the anomaly rates increase. On the other hand, the attack that optimizes delay succeeds in maintaining its anomaly rate and delay even though it is trained on a different schema mask.

TABLE IX

NON-DOMINATED CHARACTERISTICS OF ATTACKS FOR SAMBA APPLICATION AGAINST PH.

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	8.11	0.1	1,000	7.95 e+27	9.94
Delay	10.09	6.67	82	7.95 e+27	0.75
Length	8.11	0.1	1,000	7.95 e+27	9.94
Original	16.02	60.51	528	3.11 e+30	3.11 e+30

TABLE X

NON-DOMINATED CHARACTERISTICS OF ATTACKS FOR TRACEROUTE APPLICATION AGAINST PHSM.

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	2.71	0	1,000	2.68 e+07	0
Delay	43.18	100	9	0.44	3.5 e-05
Length	2.71	0	1,000	2.68 e+07	0
Original	81.79	83.06	261	8.51 e+35	8.51 e+35

TABLE XI

NON-DOMINATED CHARACTERISTICS OF ATTACKS FOR RESTORE APPLICATION AGAINST PHSM.

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	54.52	0.20	999	3.60 e+38	9.87
Delay	54.98	0.10	1,000	3.55 e+38	9.83
Length	54.98	0.10	1,000	3.55 e+38	9.83
Original	96.77	98.3	3,029	4.96 e+39	3.93 e+39

TABLE XII

NON-DOMINATED CHARACTERISTICS OF ATTACKS FOR SAMBA APPLICATION AGAINST PHSM.

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	7.36	0	1,000	1.01 e+21	0
Delay	10.62	100	9	1.59 e+20	3.5 e-05
Length	7.36	0	1,000	1.01 e+21	0
Original	99.95	99.6	528	1.41 e+40	8.96 e+38

TABLE XIII

NON-DOMINATED CHARACTERISTICS OF ATTACKS FOR TRACEROUTE APPLICATION AGAINST PHSM (SCHEMA MASK DIFFERS FROM THE TRAINING).

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	29.28	27.60	1,000	4.48 e+16	1.50 e+14
Delay	61.36	100	9	0.44	3.5 e-05
Length	29.28	27.60	1,000	4.48 e+16	1.50 e+14

TABLE XIV

NON-DOMINATED CHARACTERISTICS OF ATTACKS FOR RESTORE APPLICATION AGAINST PHSM (SCHEMA MASK DIFFERS FROM THE TRAINING).

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	57.92	0.31	999	2.56 e+39	11.1
Delay	54.13	0.10	1,000	4.04 e+38	9.88
Length	54.13	0.10	1,000	4.04 e+38	9.88

TABLE XV

NON-DOMINATED CHARACTERISTICS OF ATTACKS FOR SAMBA APPLICATION AGAINST PHSM (SCHEMA MASK DIFFERS FROM THE TRAINING).

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	15.84	29.23	1,000	1.95 e+28	7.37 e+12
Delay	9.92	100	9	1.53 e+21	3.5 e-05
Length	15.84	29.23	1,000	1.95 e+28	7.37 e+12

V. CONCLUSION

Two second generation anomaly detectors – pH and pHsm – are assessed for effectiveness under a black-box information criterion. To do so, a multi-objective model of EEG is employed. Such an approach enables us to determine the cross-section of (non-dominated) behaviours evolved post training. It is clear that the preamble plays a very significant role in establishing the effectiveness of the overall attack. Attacks that require a large number of system calls to create the context for deploying an exploit are more likely to use instructions that do not conform to the normal behavioural profile established by the anomaly detector. Conversely, if this condition is met, a significant degree of freedom exists in crafting attacks capable of avoiding detection.

The case of Traceroute versus the other applications is of particular interest. The parameterization of pH variants is a trade-off between collecting statistics over longer window sizes (minimize false positive rates) and establishing a good model for normal behaviour versus detection of very short attacks. Under Traceroute the low system call count of the preamble enables the configuration of attacks that on the one hand register a high anomaly rate, yet escape the delay property of pH and pHsm. Furthermore, when the schema mask of pHsm changes, the anomaly rate and delay characteristics of the attacks are affected but the degree of impact depends on the application. For example, Traceroute seems to be sensitive to schema mask changed whereas Restore and Samba applications are less sensitive to schema mask changes.

The results suggest that appropriate parameterization of the detectors is crucial to build effective defenses. When it comes to anomaly detectors, there is no 'one size fits all' solution since the application characteristics and the detector parameters (such as the sliding window sizes and schema mask) can affect the effectiveness of the detectors. Therefore, it is important to determine the suitable parameters for every detector deployment scenario. The EEG proposed in this paper contributes by providing an automated tool for detector parameterization.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of Swiss-Com Innovations SA. (Switzerland), TARA Inc. (Canada), Killam Predoctoral and Precarn Scholarships, and MITACS and NSERC grant programs.

REFERENCES

[1] M. Roesch, "Snort – lightweight intrusion detection for networks," in *Proceedings of Thirteenth Systems Administration Conference (LISA)*, 1999, pp. 229–238.

[2] S. Forrest, S. A. Hofmeyr, A. B. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Proceedings of the IEEE Symposium on Security and Privacy*, 1996, pp. 120–128.

[3] H. G. Kayacik, M. Heywood, and N. Zincir-Heywood, "On evolving buffer overflow attacks using genetic programming," in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*, SIGEVO. ACM, 2006, pp. 1667–1674.

[4] ——, "Evolving buffer overflow attacks with detector feedback," in *Proceedings of the EvoWorkshops (EvoCOMNET)*. Springer, 2007, pp. 11–20.

[5] D. Wagner and P. Soto, "Mimicry attacks on host based Intrusion Detection Systems," in *ACM Conference on Computer and Communications Security*, 2002, pp. 255–264.

[6] K. Tan, K. Killourhy, and R. Maxion, "Undermining an anomaly-based Intrusion Detection System using common exploits," in *Recent Advances in Intrusion Detection Systems (RAID)*, ser. LNCS, vol. 2516, 2002, pp. 54–73.

[7] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Automating mimicry attacks using static binary analysis," in *USNIX Security Symposium*, 2005, pp. 717–738.

[8] H. G. Kayacik and A. N. Zincir-Heywood, "Mimicry attacks demystified: What can attackers do to evade detection?" in *PST '08: Proceedings of the 2008 Sixth Annual Conference on Privacy, Security and Trust*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 213–223.

[9] A. B. Somayaji, "Operating system stability and security through process homeostasis," Ph.D. dissertation, The University of New Mexico, 2002.

[10] H. Inoue and A. Somayaji, "Lookahead pairs and full sequences: A tale of two anomaly detection methods," in *Proceedings of the 2nd Annual Symposium on Information Assurance (Academic track of the 10th NYS Cyber Security Conference)*, June 2007.

[11] W. Banzhaf, F. D. Francone, R. E. Keller, and P. Nordin, *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[12] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Addison Wesley, 1989.

[13] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley and Sons, 2001.

[14] R. Kumar and P. Rockett, "Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution," *Evolutionary Computation*, vol. 10, no. 3, pp. 283–314, 2002.

[15] K. M. C. Tan, J. McHugh, and K. S. Killourhy, "Hiding intrusions: From the abnormal to the normal and beyond," in *IH '02: Revised Papers from the 5th International Workshop on Information Hiding*. London, UK: Springer-Verlag, 2003, pp. 1–17.

[16] SecurityFocus, "Lbnl traceroute heap corruption vulnerability," http://www.securityfocus.com/bid/1739, Last accessed June 2008.

[17] ——, "Redhat linux restore insecure environment variables vulnerability," http://www.securityfocus.com/bid/1914, Last accessed June 2008.

[18] ——, "Samba 'call_trans2open' remote buffer overflow vulnerability," http://www.securityfocus.com/bid/7294, Last accessed June 2008.