Evolutionary Computation as an Artificial Attacker

Generating Evasion Attacks for Detector Vulnerability Testing

Hilmi Güneş Kayacık · A. Nur Zincir-Heywood · Malcolm I. Heywood

Received: date / Accepted: date

Abstract Intrusion detection systems protect our infrastructures by monitoring for signs of intrusions. However, intrusion detection systems are themselves susceptible to vulnerabilities, which the attackers take advantage of to evade detection. In particular, we focus on evasion attacks in which the attacker aims to generate a stealthy attack that eliminates or minimizes the likelihood of detection. Attackers achieve stealth by mimicking normal behaviour while achieving the attack goals, hence bypassing the detector. Previous work focused on generating evasion attacks using the internal knowledge of the detectors, hence adopting a 'white-box' access to the detector. On the other hand, we adopt a 'black-box' approach and propose an evolutionary attacker based on Genetic Programming. The access of our 'black-box' approach is limited to the feedback of the detector such as anomaly rates and delays. We compare our 'black-box' approach with various 'white-box' approaches to investigate its effectiveness. In doing so, the impact of anomalies from the break-in stage of the attacks and the delays based on locality frame counts are also discussed. This is particularly important if the performance comparison is to reflect the real capabilities of detectors.

H. G. Kayacık
Carleton University, School of Computer Science, 1125 Colonel By Drive, Ottawa ON K1S 5B6, Canada E-mail: kayacik@ccsl.carleton.ca

A. N. Zincir-Heywood
Dalhousie University, Faculty of Computer Science, 6050 University Avenue, Halifax NS B3H 1W5, Canada E-mail: zincir@cs.dal.ca

M. I. Heywood
Dalhousie University, Faculty of Computer Science, 6050 University Avenue, Halifax NS B3H 1W5, Canada E-mail: mheywood@cs.dal.ca

Keywords Computer security · intrusion detection · anomaly detection · evasion attacks · evolutionary computation · artificial arms race

1 Introduction

Buffer overflows are the result of software deficiencies due to improper handling of memory, inputs or outputs. For attackers, buffer overflows represent an important potential weakness that may lead to a wide range of outcomes e.g., vandalism, fraud, identity and intellectual property theft. While software testing provides a valuable technique for identifying and eliminating the deficiencies causing buffer overflows, it cannot guarantee that all the existing deficiencies are eliminated. Therefore, it is important to deploy intrusion detection systems to prevent attackers from exploiting the unknown deficiencies.

Intrusion detection systems provide a means to detect buffer overflow attacks either in transit on the network or during run-time on the host. Different detection techniques can be employed to search for the evidence of intrusions. To this end, two main categories exist: misuse and anomaly detection. Misuse detectors employ attack signatures whereas anomaly detectors adopt the opposite approach and utilize normal behaviour models to detect intrusions. Misuse detectors are very effective against the known attacks for which a signature exists but their detection capabilities are limited to the signatures they have. Anomaly detectors, on the other hand, are effective against the unknown attacks as long as the attack behaviour sufficiently deviates from the normal behaviour model. However, retraining may be necessary to ensure the effective detection and low false positive rates.

Naturally, intrusion detection systems, just like other systems, are not infallible. In addition to general software deficiencies and misconfigurations, they are also susceptible to detector specific vulnerabilities such as blind spots or deficiencies in detection techniques. Sophisticated attackers can take advantage of these vulnerabilities to evade detection, hence rendering the detectors ineffective. Thus, detector testing is crucial for identifying and eliminating detector weaknesses before the attackers can exploit them. Vulnerability testing can be considered as ethical hacking, where the objective is to establish the limitations of the detection methodologies. Identifying the limitations can allow us to make better decisions on the design of defences that protect the networks and the connected hosts.

This work focuses on the vulnerability testing of host-based anomaly detectors by generating evasion attacks. In a typical evasion attack, the attacker aims to alter a generic attack template – the core of an attack – so that the evasion attack 'mimics' normal behaviour to evade detection. If an attacker can generate an evasion attack, which produces zero or low anomaly rates, it can evade the detector in question. While several researchers [35] [32] [24] identified several evasion attacks against host-based anomaly detectors, the common trait is to use a 'white-box' methodology, which assumes that the attacker has full access to the detector and its normal behaviour database. In this work, we take an alternative 'black-box' approach to investigate whether an attacker can generate effective evasion attacks without access to the internal knowledge of the detector. To do so, we propose an Evolutionary Exploit Generator (EEG), which interacts with the anomaly detectors purely through the detector outputs such as the reported anomaly rates. The main objective of this paper is to provide a comparison between 'white-box' and 'black-box' detector testing techniques, thus establish to what degree the 'black-box' limitation constrains our ability to detect detector limitations. Our results demonstrate that the Evolutionary Exploit Generator assuming a 'black-box' access to the detector is as effective as the techniques assuming a 'white-box' access. Furthermore, the results suggest that the break-in stage of the attack is crucial in determining the anomaly rate of the overall attack, hence even though the attacker can design a 'stealthy' exploit, an anomalous break-in stage can cause the attack to be detected.

The remainder of the paper is organized as follows. The relevant work on the vulnerability analysis of host-based anomaly detection is provided in Section 2. The anomaly detectors employed in our experiments and the vulnerable applications are detailed in Section 3. Section 4 provides the discussion of the proposed Evolutionary Exploit Generator, which assumes a 'black-box' access to the detector. Section 5 discusses various 'white-box' testing techniques which employ the internal knowledge of the detectors such as the normal behaviour database to generate evasion attacks. Performance comparisons of the 'black-box' and 'white-box' methods are made in Section 6. Conclusions are drawn in Section 7.

2 Relevant Work

Earlier works in vulnerability analysis make extensive use of knowledge regarding the internal design of the detector, with the emphasis being directed purely at the exploit. Wagner et al. [35] investigated an approach to alter the system call sequences of an attack in order to render it undetectable to a specific IDS, namely Stide. Given a minimum sequence of malicious system calls to support execution of a successful attack – the *core* attack – their goal was to find other sequences of system calls that avoid detection by the target IDS yet still achieve the objective of the core attack. This was achieved by manually adding system calls that have no effect on the success of the attack. Similarly, Tan et al. [32] aimed to undermine the anomaly based IDS Stide [11] by identifying weaknesses and modifying the malicious system call sequences to exploit these limitations. To do so, they first modified the attack by hand to change the ownership of a critical file. Secondly, they inserted system calls from the data characterizing normal behaviour into the malicious system call sequence. Vigna et al. [34] described a methodology to generate variations of an attack to test the quality of detection signatures of Snort. Stochastic modification of attack code was employed to generate variants of attacks to render the attack undetectable. Techniques such as packet splitting, evasion and polymorphic shellcode were discussed. Kruegel et al. [24] developed a static analysis tool for Intel x86 binaries in order to automatically identify instructions that can be used to redirect control flow. They used symbolic execution to achieve this. Giffin et al. generated evasion attacks against Stide by applying automatic model checking to prove that no reachable operating system configuration corresponds to the effect of an attack [14]. However, in their approach, the operating system model, application (program) model and system call specifications as well as the attack configuration were still generated manually.

On the other hand, our work contributes to the existing work on evasion attacks in two ways. Firstly, our approach represents an arms race between various anomaly detectors and a 'black-box' attacker i.e., the Evolutionary Exploit Generator (EEG) framework. The arms race is designed to reward the attacker as it demonstrates measurable features that are potential indicators of an effective attack. Such features may take the form of minimizing anomaly rates or the minimization of dynamic counter measures deployed against an attack by the detector e.g., delays. Consequently, the EEG attacker utilizes the information feedback from the detector to build evasion attacks that achieve the objectives of the attacker while minimizing detection from the target detector. No internal knowledge of the detector is required to facilitate the arms race, instead all the feedback is based on behavioural information publicly available to the user of the target detector. The main result of the arms race is a set of evasion attacks, which can evade the target detector (or not). The resulting attacks provide the defenders with crucial information that can be utilized to guide attempts to eliminate weaknesses of the target detector. Needless to say, the exploits produced are entirely a result of the EEG with no hand crafting of the exploits.

Second, the previous work [35] [33] [32] [12] [24] [14] assumed that either (1) at the break-in stage, attackers can gain control of the vulnerable application without raising alarms, or (2) attackers have already gained control of the victim system. Although the attackers can alter their exploit after gaining full control, no consideration was given as to whether the combination of the break-in and exploit would increase the anomaly rate. In this work, we call the break-in stage, during which the attacker tries to gain control of the vulnerable application, the preamble. After the attacker gains control of the application, he/she injects a payload, which is called the exploit, to carry out the attack objectives such as spawning a UNIX shell. Therefore, attacks are comprised of two components: the preamble and the exploit. Thus, in this work, the anomaly rate is calculated not only on the exploit but also on the entire attack, which contains the anomalies from the preamble. As such, the performance evaluation reflects the ability of the detector to detect attacks as a whole (as would be the case in practice) as opposed to limiting evaluation to detecting the exploit alone; the latter biasing the evaluation in favour of the attacker and giving a misleading impression of detector vulnerability to exploits. It is therefore acknowledged that evasion attacks against anomaly detectors may not be as easy to perform in practice due to the attacker's lack of control over the system calls executed before the attacker's shellcode is invoked. Indeed, it readily becomes apparent that only when the preamble component of an attack contributes a significantly lower proportion of the attack code, it is possible to evade the more sophisticated detectors (as discussed in Section 6).

3 Vulnerable Applications and Candidate Detectors

In this section, we introduce the set of vulnerable applications used in our benchmarking study and establish the application specific instruction set employed by EEG for evolving exploits. The set of detectors, against which vulnerability testing is carried out, is then established where we provide a cross-section of detector complexity as well as covering different families of detector architecture.

3.1 Vulnerabilities

In our experiments, four Linux vulnerabilities are employed: traceroute, samba, restore and ftpd, all of which have known and documented vulnerabilities.[1] These are also the vulnerable applications most frequently used in the vulnerability testing literature [35] [32] [33]. The traceroute and restore vulnerabilities can be exploited locally whereas ftpd and samba vulnerabilities can be exploited remotely. As established by the previous work, vulnerable applications are executed under different scenarios to establish the normal behaviour for each application [35] [32] [33] [21].

The attacks against the four Linux applications are buffer overflow attacks, where the attackers take advantage of the vulnerabilities to inject their malicious code. In a typical buffer overflow attack, the attacker injects more data than the vulnerable variable can hold, hence causing the excess data to spill into the unallocated memory space or into other allocated variables. The main goal of a buffer overflow attack is to overwrite the system state information stored in the memory and consequently divert the execution to the attacker's code.

3.1.1 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. Red Hat 6.2 is shipped with traceroute version 1.4a5, where this

[1] See Security Focus Vulnerability archives `http://www.securityfocus.com`

is susceptible to a local buffer overflow exploit that provides a local user with super-user access [1]. The attack takes advantage of the 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, traceroute is executed by supplying different targets, as established by the previous work [19].

3.1.2 Restore Configuration

Restore is a component of Linux 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 arbitrary command with super-user privileges, which leads to a root compromise. In the published attack [2], the attacker spawns a root shell. In order to establish normal behaviour, restore is executed numerous times to process backup files with different sizes [19].

3.1.3 Samba Configuration

The Samba suite provides printing and file sharing facilities for Windows clients and can run on most Linux variants. Samba sets up printer and network shares that appear as disks and printers under a Windows operating system. Red Hat 9.0 is shipped with Samba suite version 2.2.7a, which has a vulnerability [3] 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. To establish normal behaviour, samba is deployed on a host and activity was generated involving the mounting and unmounting of a samba share and various file operations on the samba share including file edit and copy operations [19].

3.1.4 Ftpd Configuration

Red Hat 6.2 is shipped with Washington University Ftp Server version 2.6.0(1), which provides FTP access to remote users. WuFtpD 2.6.0(1) is susceptible to an input validation attack where the attacker can corrupt the process memory by sending malformed commands and overwrite the return address to execute a shellcode. Although the attack is an input validation attack [4], the deployment is similar to a buffer overflow attack. In order to establish a normal behaviour, numerous FTP sessions are formulated where each session involves a login followed by a series of file upload and download operations [19].

3.2 Anomaly Detectors

The anomaly detectors employed in this work monitor system call traces to detect the attacks. System calls are operating system routines, which provide the interaction between the applications and system resources such as memory, disks and peripherals. Given that the operations, which alter the system state are handled through system calls, monitoring the applications at the system call level provides a suitable granularity for detecting the attacks.

Although numerous alternative detectors exist for detecting buffer overflow attacks, there are various traits, which make the anomaly detectors discussed in this section particularly appropriate. First, they employ different detection methodologies while monitoring application system calls, with sliding window, Markov and frequency based pattern recognition detectors all being considered. Second, depending on the complexity of the detector, they provide feedback in the form of anomaly rates and (counter measure) delays, which can be utilized to guide the search for the evasion attacks.

The relevant work on anomaly detection proposed various techniques [28] [12] for extending the publicly available Stide and pH detectors. In this work, however, we employ Stide and pH because, in addition to being publicly available, they do not require the vulnerable applications to be recompiled for deployment. Specifically, the additional system state information (e.g. program counter values, return addresses stored on stack) utilized by the more recent work requires the application being monitored to be compiled in a certain way to make these aforementioned values deterministic. Considering system services such as ftpd and restore depends on many other system libraries, this is likely to require a recompile of the OS, which is beyond the scope of this work.

In this work, detectors are employed with their default parameters as established by earlier studies [11] [29] [15]. Needless to say, identifying suitable deployment parameters is crucial in ensuring the effective operation of a detector. Depending on the application being monitored and false positive requirements, different parameters may be suitable for different deployment scenarios [22].

3.2.1 Stide

Forrest et al. [11] employed a methodology motivated by immune systems. This characterizes the problem as distinguishing 'self' from 'non-self' (normal and abnormal behaviours respectively). An event horizon is built from a sliding window applied to the sequence of system calls made by an application during normal use. The sequences formed by the sliding window are stored in a table, which establishes the normal behaviour model. During the deployment (detection) phase, if the pattern from the sliding window is not in the normal behaviour database, it is considered a mismatch. Stide is employed with the parameterization assumed in earlier studies, listed in Table 1.

Table 1 Stide configuration parameters.

Parameter	Setting
Sliding window length	6

3.2.2 Process Homeostasis (pH)

Process Homeostasis (pH) [29] represents a second generation anomaly detector and is therefore designed to address specific drawbacks of Stide. pH is implemented as an extension to the Linux 2.2 Kernel. Therefore, pH monitors system calls more efficiently by capturing system calls directly at the kernel level as opposed to Stide, which employs Strace[2] to capture system calls. pH monitors the changes in short sequences of system calls by employing look-ahead pairs. While employing the sliding window approach, pH does not store the sliding window patterns but records tuples, which consist of the current and past system calls and the sliding window locations. Somayaji [29] established that the look-ahead method is more efficient to store and could potentially converge to a normal profile more quickly than the sequence method. Additionally, tolerization and sensitization concepts were introduced. Tolerization allows pH to improve false alarm rates by leaving out minimal anomalies, which are likely to be caused by slight changes in normal behaviour. Sensitization prevents abnormal behaviour from leaking into the normal behaviour database [29].

In addition, pH responds to attacks by slowing down the offending process. The delay is an exponential function of the locality frame count (LFC) where the locality frame count aims to identify the clusters of anomalies. To this end, pH simply maintains a count of how many of the past system calls within the locality frame were anomalous. Consequently, even though the attack might minimize anomaly rate, it can still be detected if the remaining anomalies are clustered together. pH was employed with the training parameters, which are listed in Table 2.

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

Table 2 pH configuration parameters.

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

3.2.3 Process Homeostasis with a Schema Mask (pHsm)

Inoue et al. [15] discussed the differences between look-ahead pairs and sequences. In their paper, the authors also proposed an improvement to pH based upon the concept of a random schema mask. Their main motivation was the observation that longer windows improve detection rates hence there exists a potential to increase the difficulty of generating evasion attacks against pH (and indirectly Stide and variants). We call the extended pH as pH with a schema mask (pHsm) in this work. In pHsm, a longer sliding window is maintained and a number of taps are taken from the sliding window. The locations of the taps are determined randomly before training and this location information constitutes the schema mask. The configuration parameters for pHsm are detailed in Table 3.

Table 3 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
Locality frame window size	128
Delay factor	1
Suspend execve after	10 anomalies
Suspend execve duration	2 days
Anomaly limit	30
Tolerize limit	12

3.2.4 The Markov Model-Based Detector

The Markov Model is a statistical modelling technique, which is useful for building probabilistic models of event sequences evolving in time. Markov Models have been utilized within the context of intrusion detection systems [31] [13] as in the related case of a Finite State Automata representation [28]. The Markov Model was selected as an anomaly detector in this work because: (1) it can build probabilistic models using exemplars from only one class (i.e. normal behaviour) and (2) it can capture temporal (i.e. sequence) information without employing a sliding window.

Although higher order Markov Models exist where the current state depends upon a number of previous states, the Markov Model anomaly detector implemented in this work employs a first order Markov Model. In a first order Markov Model, the next state is only dependent upon the current state, where such an assumption is widely employed in these systems to reduce the number of 'free parameters', which require estimation. In order to establish values for the above model parameters, the Baum-Welch model is assumed [7]. The detection decision is based upon a characterization of state transition behaviour, which was employed in the previous Markov Model detector approaches [31] and in Stide [5]. After the Markov Model is trained, a test sequence is presented. If there exists a transition in the test sequence, which was not encountered during training (hence, the probability transition is zero), a mismatch flag is set. A count of the mismatch flag is maintained, and the anomaly rate is defined by the number of mismatch flags divided by the total number of transitions encountered. Such an anomaly rate implies that, if the test sequence follows the training model (i.e. normal behaviour), it will encounter zero or low numbers of mismatch flags. Thus, a low anomaly rate is assigned. The configuration parameters for the Markov Model detector are provided in Table 4.

Table 4 Markov Model parameters.

Parameter	**Setting**
Order	First order
Number of states	223 (Number of system calls)
Training algorithm	Baum-Welch

3.2.5 Auto-Associative Neural Network

The auto-associative neural network is a multi-layer perceptron configured in a 'bottleneck' topology – i.e., the hidden layer is configured with a reduced neuron count relative to the input and output spaces. During training, the same exemplar is presented to both input and output. The architecture attempts to recreate the input at the output under the 'bottleneck' limitation, thus forcing the network to identify the most appropriate encoding to correctly reconstruct the (single) training category [23] [27] [26] [16]. An auto-associative neural network therefore provides the basis for developing models from one-class data with post-training decisions based upon the similarities to, or deviation from, the behaviour that the auto-associative neural network encapsulates. In our experiments, an auto-associative neural network was employed as an anomaly detector. As opposed to the other detectors discussed in the previous sections, which employ sequence information, the input to the auto-associative neural network takes the form of the frequency distribution of system calls. This approach bears similarities to the detector employed by Kang et al. [17], which uses a bag of words representation as the detector input. As such, the resulting frequency distribution constitutes the normal behaviour characteristics. Given the frequency distribution vector for the test trace, detection is therefore based upon the divergence between the frequency distribution vectors of the test trace and the 'normal behaviour'. Since the frequency distribution of a trace is calculated after the trace is complete, the auto-associative neural network can be considered an 'off-line' detector, which provides post-mortem analysis of the system call traces after they are executed.

Given q dimensional data, a multilayer perceptron with p nodes in the hidden layer ($p \ll q$) and q nodes in the output layer is trained. The goal of training is to find an encoding that recreates the training partition. Under test conditions, the network will produce an output similar to the input if the input is similar to what was encountered during training. From the perspective of anomaly detection, if the applied input does not produce an output similar to the input, it is considered anomalous. In order to measure the degree of anomaly and produce an anomaly rate, the Euclidean distance between input and output is compared to a threshold. The calculated distance is scaled to vary between 0 and 100, where larger numbers indicate anomalous behaviour. The training parameters for the auto-associative Neural Network are detailed in Table 5, with the training algorithm taking the form of the very efficient second order conjugate gradient optimization algorithm, making training much more efficient and more accurate than the regular back propagation algorithm [8].[3]

[3] As implemented using the MATLAB Neural Network Toolbox.

Table 5 Auto-associative Neural Network parameters.

Parameter	Setting
No. of neurons in hidden layer	15
Hidden layer transfer function	Hyperbolic tangent sigmoid (tansig)
No. of neurons in output layer	223
Output layer transfer function	Linear (purelin)
Training function	Conjugant gradient backpropagation
Maximum epochs	1000
Minimum mean square error	10^{-6}

4 A 'Black-box' Attacker Scenario: Evolutionary Exploit Generator

The process at the center of the proposed 'black-box' attacker involving the Evolutionary Exploit Generation (EEG) framework is Genetic Programming (GP). The GP paradigm differs from most machine learning methodologies in that a 'population' of candidate solutions is maintained concurrently throughout the search process [6]. Each candidate solution, or individual, takes the form of a program i.e., a sequence of system calls in the case of this work. Although parameters for the system calls are specified, there is no need to support the specification of the internal state i.e., register values. This makes the resulting attacks real exploits as opposed to just system call sequences.

Aside from being able to conduct a stochastic search over a code based representation, GP provides several properties that make it a particularly attractive model for evolving exploits [21]. We consider the top three properties to take the form of:

Representation: Machine learning paradigms generally impose an *a priori* representation on the nature of a solution e.g., neurons in artificial neural networks, kernels in support vector machines or rules in decision tree induction. The representation required by the exploits imposes a system call based representation (although solutions based on the assembly language of the target platform would also be appropriate [20]). This precludes the utility of most machine learning algorithms. Thus, the automated variants of earlier research in this area have relied on exhaustive search, an option made possible by making extensive use of privileged information from the target detector, e.g., the content of the detector's behavioural database post configuration (training) [35] [33] [32]. By assuming the representation of the actual application, the credit assignment process becomes more direct and support for including the preamble is straight forward (merely a question of concatenating preamble and exploit).

Multi-criteria fitness: Expressing exploit generation as a single objective – for example constructing code representing a valid exploit – would not sufficiently encompass the breadth of the task at hand. In particular, exploits need to achieve a malicious objective while simultaneously minimizing the alarm rate and reducing any delay imposed by detectors capable of dynamic/ reactive countermeasures. One approach to doing this might be to merely linearly combine three performance functions into a single scalar value of fitness. However, Evolutionary Computation provides a much better mechanism in the form of Pareto multi-criteria formulations [10]. In particular, assuming a Pareto formulation implies that objectives are ranked relative to the number of other individuals they dominate in the Pareto sense. This avoids any need to impose arbitrary scaling of different objectives and encourages solutions to take the form of a front of non-dominated individuals as opposed to converging to a single possibly sub-optimal solution.

Obfuscation: Solutions from GP take the form of a program expressed in terms of a system call sequence. However, not all system calls comprising a solution necessarily contribute to the underlying operation of the individual, where this artifact is synonymous with the introns of biological genomes. Code bloat or introns are a well known by product of search in GP [6]. Typically, instructions corresponding to intron behaviour are removed post learning. However, when generating evasion attacks, introns are beneficial to the attacker, if their statistical characterization matches normal behaviour as measured by the detector. During the fitness calculation, the feedback from the detector guides evolution toward the intron code that matches the statistical characterization of normal behaviour. Consequently, this represents a scenario, in which introns have a specific measurable contribution to the utility of the individuals.

Figure 1 provides an overview of the EEG framework. At each iteration during training, Pareto ranking calculates the rank of each individual (i.e. an exploit) by employing the Pareto dominance concept as detailed in Section 4.2. Higher ranked individuals correspond to more successful exploits, which implies the exploit minimizes the anomaly rate from the detector and maximizes the fitness, as established in Figure 2. At each iteration, two individuals are selected from the population, in which the probability of selection is proportional to the Pareto ranks. Two children are created by applying the search operators as defined in Section 4.3. The resulting children are converted to executable exploits and injected to the vulnerable application using publicly available code injection methods [1] [2] [3] [4]. The feedback of the exploits takes two forms: First, the anomaly rates (and delays for pH and pHsm) are collected from the detector, which monitors the vulnerable application. Second, attack fitness – in terms of achieving the attack objectives – is calculated by employing the fitness function defined in Figure 2. Pareto ranking, as detailed in Section 4.2, takes the anomaly rate, delay and attack success as the measured parameters and ranks the exploits where the individuals are encouraged to maximize the attack success and minimize the delay and anomaly rate.

The principle EEG design decisions are now limited to defining the instruction set (representation) and search and selection operators (establishes basis for credit assignment), and establishing the appropriate feedback (goals/ objectives) used to guide the process of evolution. The following subsections address each of these independently.

4.1 Representation

The instruction set is defined to be the 20 system calls which are most frequently utilized by each vulnerable application as discussed in Section 3.1. The implicit assumption is that the most frequently employed system calls are less likely to cause alarms than the infrequent system calls. Determining the most frequently utilized 20 system calls involves an attacker to the analyse the system calls on a local copy of the vulnerable application. Such analysis can be performed using publicly available system call profilers such as Strace and does not require the internal knowledge of the vulnerable application.

An EEG instruction is defined to be 4 bytes, where the first 2 bytes are allocated for the function identifier (i.e. the opcode) and one byte is allocated for each function parameters (i.e. operands). Therefore, EEG utilizes a fixed-length representation where the first two bytes determine the instruction and the remaining two bytes can define up to two instruction parameters. A simplified instruction set for EEG is provided in Table 6 in which an instruction is defined to be 1 byte for better readability.

Furthermore, Table 7 provides an example of the genotype to phenotype mapping. Based on the instruction set defined in Table 6, each instruction is converted to a binary representation for decoding. The first four bits determine the instruction and the remaining four can define up to two parameters. If the instruction has no parameters, the last four bits are ignored. For example, the fourth instruction, which is 61 (decimal) in Table 7, is first converted to the binary representation. The first four bits (i.e. '0011') maps to write ($< file >$, $< buffer >$). The last four bits determine the parameters, in which '11' (binary) maps to file3 and '01' (binary) maps to buffer1. As new instructions are introduced as a result of initialization and mutation, the validity of the individual is checked to ensure that the instruction and parameter fields produce a valid instruction.

Table 6 A sample instruction set and parameters.

Instruction Set	
Value	**Instruction**
0001	exit ( )
0010	open ($< file >$)
0011	write ($< file >$, $< buffer >$)
0100	read ($< file >$, $< buffer >$)
0101	close ($< file >$)

File Descriptors $< file >$	
Value	**Descriptor**
01	file1
10	file2
11	file3

Buffers $< buffer >$	
Value	**Buffer**
01	buffer1
10	buffer2
11	buffer3

Table 7 Genotype to phenotype mapping of a sample individual using the instruction set defined in Table 6.

Integer Representation	**Binary Representation**	**Phenotype**
40	0010 10 00	open (file2)
44	0010 11 00	open (file3)
59	0011 10 11	write (file2, buffer3)
61	0011 11 01	read (file3, buffer1)
88	0101 10 00	close (file2)
92	0101 11 00	close (file3)
16	0001 00 00	exit ( )

4.2 Fitness Calculation and Pareto Ranking

Pareto Ranking is a method for combining multiple objectives under the concept of dominance [10]. Specifically, in the case of a problem in which multiple objectives are being optimized, solution A dominates solution B, if and only if A is as good as B in all objectives and A is better than B in at least one objective. An individual, which is not dominated by any other individual is called a non-dominated individual. Pareto

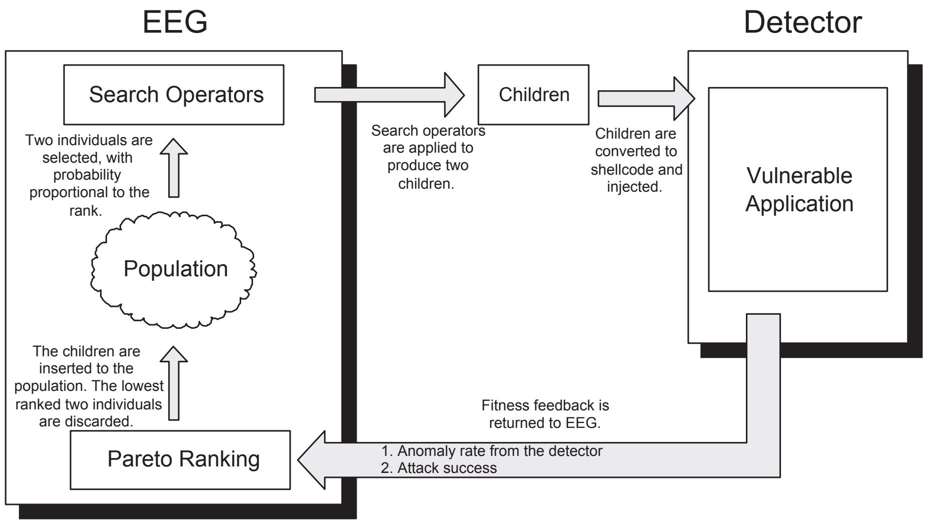

EEG
Detector
Search Operators
Children
Search operators are applied to produce two children.
Children are converted to shellcode and injected.
Vulnerable Application
Two individuals are selected, with probability proportional to the rank.
Population
The children are inserted to the population. The lowest ranked two individuals are discarded.
Fitness feedback is returned to EEG.
Pareto Ranking
1. Anomaly rate from the detector
2. Attack success

Fig. 1 An overview of the EEG framework.

ranking succeeds in reducing the multi-objective vector into a scalar fitness value (i.e. the rank) without combining features or assigning *a priori* weights. The Pareto rank of an individual in our experiments is equal to the number of individuals, which it dominates [25]. In terms of evasion attack characteristics to be optimized, the following objectives are considered:

1. **Attack Success.** The original attack contains a standard shellcode, which uses the *execve* system call to spawn a Linux shell upon successful execution. *Execve* is a system call, which executes the program given as the first argument. Since *execve* is not a frequently used system call for traceroute, restore, samba and ftpd, it is expected that the original attack will be detected easily. To this end, a different strategy is employed for defining the exploit such that the need to spawn a Linux shell is eliminated [20]. Typically, most programs perform I/O operations – in particular to open, write to / read from and close files. Therefore the goal of the attack is altered to involve the following three steps, which aim to gain super-user privileges:
   (a) open the Linux password file ('/etc/passwd');
   (b) write a line, which provides the attacker a super-user account allowing login without a password;
   (c) close the file.

   The objective of the search process conducted by EEG is to discover a sequence of system calls (and appropriate arguments), which perform the above three steps in the correct order (i.e. the attack cannot write to a file, which it has not first opened), while minimizing the anomaly rate from the detector. A behavioural success function rewarding the above behaviour awards a total of 5 'points' for establishing the behavioural steps for the 'core' attack in Figure 2.

(a) Success = 0
(b) IF the sequence contains open ('/etc/passwd') THEN Success += 1
(c) IF the sequence contains write ('toor::0:0:root:/root:/bin/bash') THEN Success += 1
(d) IF the sequence contains close ('/etc/passwd') THEN Success += 1
(e) IF open precedes write THEN Success += 1
(f) IF write precedes close THEN Success += 1

Fig. 2 Fitness function for establishing the objectives of modifying the Linux password file.

2. **Anomaly Rate.** The anomaly rate represents the principal 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.
3. **Delay.** In addition to reporting anomaly rates, various anomaly detectors respond to anomalies by enforcing delays, as discussed in Section 3.2. Therefore, the attacker aims to minimize the delays associated with the attacks.

4.3 Search and Selection Operators

The search process progresses through the iterative application of selection and search operators, Figure 6. The selection operator is applied in two stages; in part 1, two individuals are identified from the population (the parents) with probability of selection proportional to the rank of the individual. Search operators are then applied to the individuals, resulting in two children. The children are appended to the population and the population is Pareto ranked. The worst two individuals (i.e. the individuals with the lowest two ranks) are discarded from the population, hence restoring the population size, or part 2 of the selection operator's role. Moreover, by pursuing a fitness function based on Pareto ranks, we are able to seamlessly incorporate multiple criteria into the performance evaluation without resorting to arbitrary combinations of unrelated objectives [25]. Application of search and selection operations continues until either the convergence or termination criteria are satisfied, Figure 6. The training parameters, which remain the same over all applications, are detailed in Table 8. Individuals are defined using a variable length format, and population initialization creates individuals with varying program lengths.

Search operators take three forms: cut and splice crossover, instruction-wise mutation and instruction swap. Note that all search operators are applied stochastically relative to a predefined probability of application, Table 8. The specific details of each operator are detailed below.

4.3.1 Cut and Splice Crossover

The crossover operator provides a scheme for investigating instruction sequences that exist currently in the population, but in different contexts. The cut and splice crossover operator selects, with uniform probability, separate crossover points on each parent. Therefore, the children can have different lengths from their parents. An example of cut and splice crossover is provided in Figure 3.

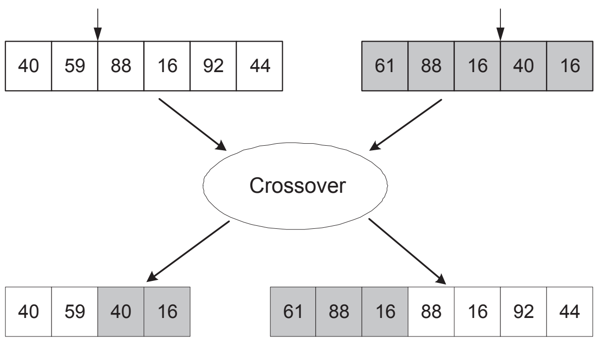


Fig. 3 An example of cut and splice crossover.

4.3.2 Swap

The basic motivation of the swap operator is to provide the opportunity for investigating the significance of different instruction orders within the same individual (the case of a correct instruction mix, but in the wrong order). The swap operator is applied to a single individual, selecting two instructions with uniform probability and interchanging their position, as shown in Figure 4.

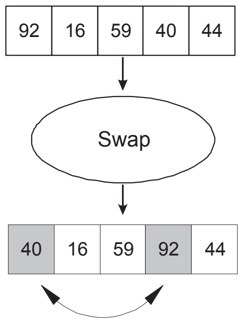


Fig. 4 An example of the swap operator.

4.3.3 Instruction-wise mutation

The 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 true then the instruction is replaced with an alternative instruction from a predefined list of instructions. Moreover, the probability of applying the mutation operator decays linearly with the tournament count, thus lowering the likelihood of introducing new instructions (i.e., not currently in the population) as the population evolves. This places more emphasis on the crossover operator as evolution progresses, thus reinforcing the reuse of system call sequences, which were demonstrated earlier to minimize detection. Figure 5 provides an example of instruction-wise mutation, in this specific case resulting in two instructions being modified.

5 A 'White-box' Attacker Scenario: Exhaustive Search

In order to compare against the 'black-box' approach discussed in Section 4, various 'white-box' evasion attack strategies are considered. A 'white-box' method requires a focused and exhaustive search against the detection mechanisms and can employ internal knowledge

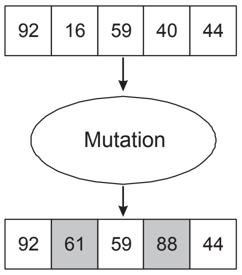


Fig. 5 An example of the mutation operator.

```
1. Initialize the population
2. Pareto rank the population
3. WHILE ! (Stop Criteria) AND ! (Convergence Criteria)
   (a) Select two individuals from the population and copy it
       to $chidren1$ and $children2$
   (b) $ApplyCrossover(children1, children2, P_{xo})$
   (c) $ApplyMutation(children1, P_{mut})$
       $ApplyMutation(children2, P_{mut})$
   (d) $ApplySwap(children1, P_{swp})$
       $ApplySwap(children2, P_{swp})$
   (e) Insert chidren1 and children2 into the population and
       Pareto rank
   (f) Remove the worst ranking two individuals and restore
       the population size
```

Fig. 6 Pseudo-code of the EEG training.

Table 8 Genetic Programming 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 normal use cases
Program length	Initialized over 240 system calls, maximum 1,000 system calls
Replacement	Children replace the lowest ranked two individuals
Training time	Approximately 2 days
Number of runs	50

such as the training data, normal behaviour database or the detection methodology utilized. Thus, for each target detector, a separate 'white-box' method needs to be developed that fits the characteristics and the constraints of the detector. For example, a 'white-box' attacker against the neural network detector, which monitors the frequency distribution of system calls, may not succeed against a detector, which monitors not only the frequencies but sequences of system calls. To this end, the 'white-box' attackers developed against each detector are presented separately in the following subsections.

5.1 The 'White-box' Attacker Against Stide

Wagner et al. [35] employed language theory to formulate two sets: a set of malicious sequences and a set of normal behaviour sequences. If the intersection of these sets is not an empty set, it implies that an evasion attack can be constructed. On the other hand, Tan et al. [32] viewed the problem as increasing the length of the malicious sequence beyond the sliding window length, which means that the sliding window patterns generated on the malicious sequence would be within normal behaviour. The common trait of both methods is to generate an evasion attack, which does not contain an anomalous sub-sequence. In other words, every sliding window pattern that the evasion attack produces should be in the normal database of the detector.

Sharing this goal, the 'white-box' evasion attack generation technique developed for Stide in this work takes the Stide normal database (i.e. the sequences of system calls) and builds a connectivity graph where each pattern in the normal database is represented as a node. Figure 7 provides an example in which a simple Stide normal database is provided. The sliding window length is set to 3 for simplicity. Each number in Figure 7 uniquely maps to a system call. Furthermore, the connectivity graph, which corresponds to the normal database is given. In the connectivity graph, a unidirectional connection from $A = [1\ 2\ 3]$ and $D = [2\ 3\ 7]$ means that the graph node (i.e. sliding window pattern) D can follow A without creating any anomalous subsequences. The exhaustive search was implemented as a depth-first graph search where the search terminates when the resulting sequence contains the malicious system calls (in this case, open - write - close). For example, the depth first search of the path $A \Rightarrow D \Rightarrow G$ produces a sequence of [1 2 3 7 1], which does not produce any anomalies.

5.2 The 'White-box' Attacker Against pH

As with the methodology proposed by Wagner et al. [35], the methodology developed against pH in this work exploits the use of sliding windows in pH and builds attacks from the sliding window patterns encountered during training. Simply put, the approach developed by Stide in Section 5.1 is modified with a sliding window length of 9. This is an acceptable approach

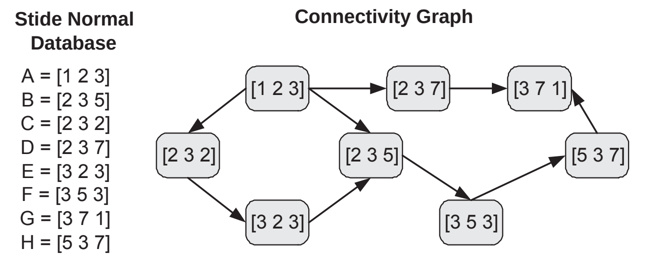


Fig. 7 A example of the Stide normal database (sliding window length of 3) and the corresponding connectivity graph

since a 'white-box' approach implies that the attacker can utilize all and any knowledge about the detector, including the detection methodology. This reduces the search space size of a 'white-box' search, hence shortening the time it takes to find an attack.

5.3 The 'White-box' Attacker Against pHsm

Although, as of this writing, pHsm has not been employed in previous evasion attack research, the methodology discussed in Section 5.1 applies to pHsm. Two attributes of pHsm differentiate it from Stide and pH: (1) the sliding window length is 20 and (2) a schema mask is applied to the sliding window. However, given a 'white-box' access to the detector, it is realistic to assume that the attacker knows both the sliding window length and the schema mask. Thus, the task can be formulated as generating a evasion attack against a detector with a sliding window of 20 system calls. This implies that the methodology detailed in Section 5.1 is modified to reflect a sliding window length of 20. If the generated attack does not raise any alarms, the application of a schema mask will not generate any anomalies since the purpose of the schema mask is to select 9 values from a sliding window pattern of length 20 (which does not produce any anomalies as a requirement of the exhaustive search).

5.4 The 'White-box' Attacker Against Markov Model-Based Detector

Tan et al. [30] utilized a methodology to generate evasion attacks against a Markov Model detector. The main focus of their experiments was to determine the operational limits of Stide, therefore the attacks were generated against Stide and tested against both Stide and the Markov model detector. Given that the Markov Model detector employed in this work is a first order model, as detailed in Section 3.2.4, the current state (i.e. the system call) depends solely upon the previous system call. This differs from Stide, pH and pHsm, where the current state depends upon a history of previous system calls. The 'white-box' search on the Markov Model can be formulated as a graph search where each system call is a state. States A and B are connected in the graph if the detector encounters any transitions from A to B in the training data. The search for evasion attacks now boils down to a depth-first search on the graph. The search terminates if the generated sequence contains the malicious sequence (i.e. open - write - close).

5.5 The 'White-box' Attacker Against Auto-Associative Neural Network

The Neural Network detector differs from the rest of the anomaly detectors utilized in this work since the detection is based upon the frequency distribution as opposed to the sequence of system calls. Such a representation is more compressed than the sequence-based methods, which means the attackers do not have to consider determining sequences that contain malicious system calls. For example, from the perspective of the Neural Network detector, which monitors the frequency distribution of system calls, sequence 2 2 1 3 2 2 1 is equivalent to 1 1 2 2 2 2 3. Furthermore, an evasion attack can evade detectors, which monitor system call sequences by repeating patterns that are in the normal database. However, such an approach may not work for a detector monitoring system call frequencies because although the sequences employed in the evasion attack are in the normal database, the resulting frequency distribution may deviate from the frequency distributions encountered during training. The 'white-box' methodology developed against the Neural Network detector analyses the frequency distributions in the training sets and generates evasion attacks to match the frequency distribution. Again, this conforms to a 'white-box' assumption since the attacker can use internal knowledge including access to the training partition. Given that the detector does not employ sequence information, the methodology rearranges the ordering of the system calls randomly. Thus, as long as the frequency distributions match and the malicious open-write-close sequence exists, the 'white-box' attacker does not care about the ordering.

6 Results

To date, the approach to evasion attack evaluation has been to report the anomaly rates of attacks against the detector for which they were trained. For example, an attack trained for Stide was tested against Stide alone.

Although such a deployment scenario is sufficient to make a comparison between the original and the evasion attack, a natural extension is to deploy the evasion attacks against other detectors (e.g. an attack for Stide tested against pH). In such an extended scenario, it is important to make the distinction between a 'training' detector and a 'test' detector. The training detector is the anomaly detector with which the proposed approach interacted while generating the evasion attacks. On the other hand, a test detector is an alternate anomaly detector that was not employed during the course of crafting the attack. Such a scenario illustrates the case of an attacker using a different detector to generate evasion attacks. Thus, we aim to analyse the anomaly rate of the automatically generated 'black-box' and 'white-box' evasion attacks against a cross-section of test detectors. Such an analysis provides insight into the degree of portability that might be expected in an evasion attack evolved under one context by evaluating under a cross-section of contexts. By doing so, we model the scenario in which an attacker is only able to gain access to a single target detector for the purposes of training their evasion attack. Such an evaluation therefore answers the question as to what degree successful/ unsuccessful performance under one detector implies similar behaviours under the other detectors, or conversely, are there particular detectors that represent better training targets than others (as they result in evasion attacks that are more general)? Naturally, such an evaluation still respects commonality in the application against which the evasion attack is designed.

In addition, as pointed out in the literature review, previous work [35] [33] [32] [12] [24] [14] assumed a 'white-box' access to the anomaly detectors. This implied that the attacker could use the normal database of the detector, the (detector) training sets and knowledge of the detection mechanism to facilitate the 'white-box' attack generation process. On the other hand, the 'black-box' approach taken in this work implies that knowledge of the detector's internal mechanism is 'hidden' from the attacker. Therefore, the attacker has to interact with the detector during its operation and utilize feedback from the detector (e.g., in the form of anomaly rates) to facilitate the 'black-box' attack generation process. Thus, we investigate whether an attacker can generate evasion attacks without using the internal knowledge of the detector.

6.1 Minimizing anomaly rates

Table 9 details the anomaly rate of both 'black-box' and 'white-box' exploits, when the exploits are generated for a particular detector and tested against all five detectors but *without* including the preamble in the estimation of the anomaly rate. The 'black-box' EEG maintains multiple solutions per application/ detector pair, hence multiple exploits are generated. On the other hand, 'white-box' methodologies terminate when an exploit is found, thus, typically only one exploit is generated per application/detector pair. For the comparisons between 'black-box' and 'white-box' approaches, the best EEG exploit is selected based on the least anomaly rate (observed during training).

The first group of results reflects the training detector case of Stide. Comparing 'black-box' versus 'white-box' results (respectively the left versus right hand side of Table 9) indicates, for example, that a 16.67% anomaly rate is returned under the target Stide detector on traceroute for the case of a 'black box' exploit authoring whereas it produces a 81.25% anomaly rate, when deployed against pHsm. Similarly, the 'white-box' exploit for Stide succeeds in executing without any anomalies, i.e. 0% anomaly rate. However, when it is deployed against pHsm, it produces an 18.97% anomaly rate. Hence, the anomaly rates are generally higher when the exploit is deployed against different test detectors from the Stide training target. Comparison of the exploit anomaly rates in Table 9, indicates that 'white-box' methodologies produce exploits with lower anomaly rates. However, 'white-box' assumption comes with the cost of requiring access to the internal detector knowledge such as normal behaviour databases. Requiring such access has the following implications:

- The 'white-box' methodologies generally boil down to performing an exhaustive search. Thus, the utilization of internal knowledge is *required* to limit the exhaustive search so that it is computationally feasible as the total number of all possible system call sequences tends to be very large (i.e. 10^{2341} for a sequence length of 1000 [18]).
- In order to formulate attack generation as an exhaustive search, certain abstractions need to be made such as utilizing sliding window patterns or the formulation of the search on a graph structure, as discussed in Section 5. The success of the resulting attacks heavily depends on the accuracy of such abstractions.
- Since different detectors implement a normal behaviour model in different ways, a separate 'white-box' technique must be developed *per detector*. This is likely to raise scalability issues if multiple detectors, employing different detection mechanisms need to be tested.

'Black-box' Exploits					
	Stide	pH	pHsm	Markov Model	Neural Network
Target: Training conducted on Stide					
traceroute	16.67%	29.63%	81.25%	14.29%	37.24%
restore	0.40%	0.20%	0.81%	0.20%	2.73%
samba	0.50%	0.30%	0.20%	0.20%	57.15%
ftpd	57.14%	33.33%	100.00%	18.18%	34.71%
Target: Training conducted on pH					
traceroute	98.25%	11.71%	13.00%	10.92%	73.03%
restore	42.57%	0.10%	0.20%	0.20%	1.80%
samba	81.63%	0.10%	0.71%	0.30%	36.21%
ftpd	24.30%	0.10%	1.12%	0.20%	11.55%
Target: Training conducted on pHsm					
traceroute	100.00%	86.10%	86.15%	15.08%	100.00%
restore	74.37%	0.60%	11.01%	0.20%	16.12%
samba	99.50%	31.22%	29.23%	25.47%	40.35%
ftpd	100.00%	55.12%	38.73%	33.57%	14.38%
Target: Training conducted on Markov Model					
traceroute	100.00%	85.26%	84.56%	0.10%	100.00%
restore	99.30%	41.69%	51.93%	0.10%	32.06%
samba	100.00%	37.81%	28.50%	0.10%	43.39%
ftpd	100.00%	42.50%	26.07%	0.10%	12.50%
Target: Training conducted on Neural Network					
traceroute	100.00%	99.50%	99.29%	75.62%	2.47%
restore	100.00%	97.28%	94.50%	58.54%	2.90%
samba	100.00%	87.11%	83.10%	51.25%	16.68%
ftpd	97.39%	61.83%	20.82%	30.07%	3.46%

'White-box' Exploits					
	Stide	pH	pHsm	Markov Model	Neural Network
Target: Exhaustive search conducted on Stide					
traceroute	0.00%	5.80%	18.97%	0.00%	100.00%
restore	0.00%	0.00%	0.00%	0.00%	100.00%
samba	0.00%	4.39%	19.07%	0.00%	100.00%
ftpd	0.00%	7.87%	14.66%	0.00%	100.00%
Target: Exhaustive search conducted on pH					
traceroute	0.00%	0.00%	5.26%	0.00%	100.00%
restore	0.00%	0.00%	0.00%	0.00%	100.00%
samba	0.00%	0.00%	4.17%	0.00%	100.00%
ftpd	0.00%	0.00%	0.00%	0.00%	100.00%
Target: Exhaustive search conducted on pHsm					
traceroute	0.00%	0.00%	0.00%	0.00%	100.00%
restore	0.00%	0.00%	0.00%	0.00%	100.00%
samba	0.00%	0.00%	0.00%	0.00%	100.00%
ftpd	0.00%	0.00%	0.00%	0.00%	100.00%
Target: Exhaustive search conducted on Markov Model					
traceroute	96.74%	96.63%	100.00%	0.00%	100.00%
restore	97.78%	75.86%	80.26%	0.00%	100.00%
samba	99.16%	81.90%	81.90%	0.00%	100.00%
ftpd	98.02%	90.82%	91.95%	0.00%	100.00%
Target: Exhaustive search conducted on Neural Network					
traceroute	100.00%	100.00%	71.41%	66.58%	1.87%
restore	11.17%	0.07%	0.07%	0.03%	0.22%
samba	85.67%	42.91%	42.91%	42.80%	2.38%
ftpd	74.78%	49.90%	0.17%	24.96%	0.46%

Table 9 Anomaly rates of the 'black-box' and 'white-box' exploits tested *without preamble* on the Stide, pH, pHsm, Markov Model and Neural Network detectors. The exploit anomaly rates are categorized according to the target detectors.

Table 10 details the anomaly rate of the corresponding attacks, which also includes the anomalies from the preamble. The results show that although 'white box' attackers can craft an exploit with 0% anomaly rate under no preamble conditions (as shown in Table 9) the resulting attacks produce anomalies above 0% when the required preamble is included. For example, all the 'white-box' exploits generated against pHsm in Table 9 produce 0% anomalies, except the cases when they are deployed against the Neural Network detector. On the other hand, the corresponding 'white-box' attacks Table 10 produce anomaly rates between 0.95% and 100%.

A distinct trend emerges with respect to 'black-box' and 'white-box' exploits. While the inclusion of the preamble generally increases the anomaly rate for the 'white-box' attacks, it frequently reduces the anomaly rates for 'black-box' attacks. This is likely due to the fact that EEG utilizes the code bloat property of GP to generate exploits, which are comparably longer than the 'white-box' exploits (detailed later by Tables 12 and 13). On the other hand, 'white-box' techniques utilize exhaustive search, which terminates when a malicious sequence is found therefore, the sequences tend to be shorter (with the exception of Neural Network exploits, Table 12). Given that the break-in phase (i.e. the preamble, the details of which are provided in Table 11) is anomalous, a longer exploit has a better chance of reducing the overall attack anomaly rate.

The anomaly rates in Table 10 indicate that attacks against the Markov Model do not generalize well to the other detectors. Given that a first order Markov Model can be described as a detector with a sliding window length of 2 (i.e. the current state depends only upon the immediately previous state) as opposed to Stide, pH and pHsm with sliding window lengths greater than or equal to 6, it is conceivable that an attack that optimizes for a shorter sliding window may not be applicable to the detectors utilizing a longer sliding window. When the attacks generated against the Neural Network detector were deployed against other detectors, the resulting exploit and attack anomaly rates were typically high for the alternative detectors. Conversely, attacks not specifically trained against the Neural Network tended to be detected by the Neural Network, where this was particularly true of the 'white box' approach. This could be attributed to two factors: (1) as opposed to utilizing system call sequence information, a Neural Network detector utilizes the frequency distribution of system calls, which is a more compressed

'Black-box' Attacks					
	Stide	**pH**	**pHsm**	**Markov Model**	**Neural Network**
Target: Training conducted on Stide					
traceroute	10.96%	33.75%	75.36%	7.95%	15.89%
restore	46.25%	48.69%	59.51%	21.09%	6.44%
samba	3.00%	8.15%	7.41%	5.45%	9.48%
ftpd	19.30%	22.19%	38.21%	6.20%	2.69%
Target: Training conducted on pH					
traceroute	73.25%	18.29%	30.72%	8.14%	44.95%
restore	63.39%	48.57%	59.43%	21.05%	6.11%
samba	19.74%	8.11%	10.06%	5.47%	9.29%
ftpd	20.60%	16.11%	28.18%	4.50%	3.16%
Target: Training conducted on pHsm					
traceroute	96.15%	83.27%	85.12%	14.42%	100.00%
restore	76.46%	48.87%	63.67%	21.10%	9.55%
samba	23.40%	14.45%	15.84%	10.64%	11.76%
ftpd	41.39%	31.19%	38.43%	13.69%	3.64%
Target: Training conducted on Markov Model					
traceroute	95.98%	82.65%	83.97%	0.20%	100.00%
restore	86.63%	65.34%	79.92%	21.05%	14.27%
samba	23.24%	15.72%	15.63%	5.45%	7.40%
ftpd	41.48%	27.76%	35.01%	4.47%	3.92%
Target: Training conducted on Neural Network					
traceroute	96.15%	96.27%	97.97%	71.92%	1.63%
restore	86.88%	87.55%	97.04%	44.52%	5.60%
samba	23.53%	25.84%	26.72%	15.92%	5.77%
ftpd	40.76%	33.08%	41.62%	12.76%	1.26%

'White-box' Attacks					
	Stide	**pH**	**pHsm**	**Markov Model**	**Neural Network**
Target: Exhaustive search conducted on Stide					
traceroute	2.61%	18.85%	36.04%	1.54%	100.00%
restore	73.46%	76.78%	86.23%	33.10%	100.00%
samba	3.51%	9.87%	9.70%	6.45%	100.00%
ftpd	18.29%	21.51%	14.73%	5.84%	100.00%
Target: Exhaustive search conducted on pH					
traceroute	3.95%	22.89%	43.06%	2.20%	100.00%
restore	75.03%	78.31%	87.97%	33.79%	100.00%
samba	3.59%	9.94%	9.14%	6.65%	100.00%
ftpd	18.92%	21.88%	14.49%	6.05%	100.00%
Target: Exhaustive search conducted on pHsm					
traceroute	1.53%	9.36%	15.10%	0.95%	69.03%
restore	75.03%	78.31%	87.97%	33.79%	100.00%
samba	3.59%	9.95%	9.06%	6.65%	100.00%
ftpd	18.84%	21.79%	14.39%	6.02%	35.49%
Target: Exhaustive search conducted on Markov Model					
traceroute	68.15%	73.94%	87.02%	1.33%	100.00%
restore	79.06%	80.83%	90.35%	33.04%	100.00%
samba	6.54%	12.24%	11.27%	6.60%	100.00%
ftpd	22.13%	24.65%	17.25%	5.91%	40.29%
Target: Exhaustive search conducted on Neural Network					
traceroute	94.83%	95.52%	71.04%	62.36%	3.84%
restore	11.74%	0.78%	0.86%	0.34%	0.27%
samba	81.14%	41.09%	41.04%	40.79%	3.12%
ftpd	74.35%	49.68%	0.28%	24.82%	0.95%

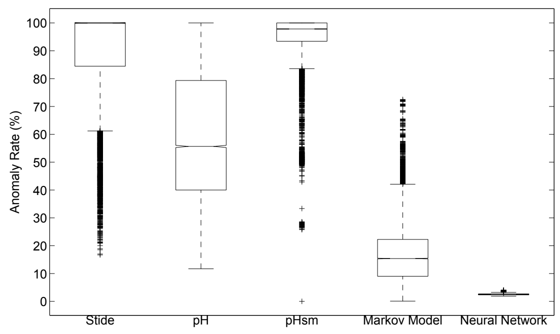
Table 10 Anomaly rates of the 'black-box' and 'white-box' attacks tested *with preamble* on the Stide, pH, pHsm, Markov Model and Neural Network detectors. The attack anomaly rates are categorized according to the target detectors.

representation of system call sequences; (2) in utilizing internal knowledge of a detector 'white box' approaches tie themselves more closely to specific 'families' of detector than the 'black box' approach. Given the dissimilarity between representations employed by Neural Network and the other detectors, less generalization is possible between behaviours appropriate for defeating both classes of detectors. Conversely, the 'black box' approach establishes attack behaviours without resorting to detector specific information, where this appears to confer more generic/ less specific attacks. This also implies that from a detector deployment perspective, multiple detectors with fundamentally different representations should be utilized for maximizing the likelihood of attack detection.

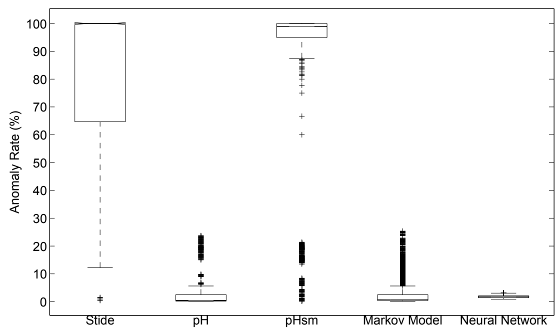
In addition to providing a comparison between 'black-box' and 'white-box' exploits, we also provide the anomaly rates observed for all the exploits evolved by EEG using box plot analysis [9]. Such analysis is not possible for 'white-box' attacks because the search has a very limited focus: finding the first exploit with 0% anomaly rate. This implies that, the 'white-box' methodologies provide one attack per attack, detector pair whereas the EEG provides 25,000 exploits, i.e. 50 runs, 500 per population, as defined by Table 8.

The box plot defines the third quartile, median and first quartile. Whiskers extend from each end of the box to the adjacent values in the data which are within 1.5 times the inter-quartile range from the ends of the box. Outliers are data with values beyond the ends of the whiskers and are displayed with a plus sign.

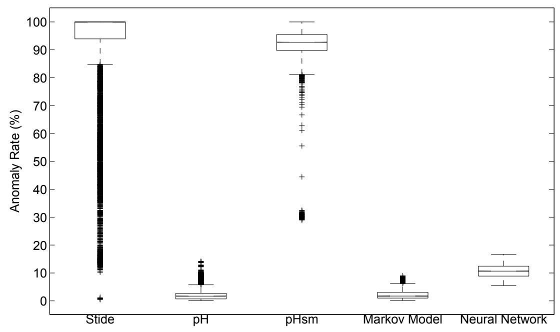
As opposed to Tables 9 and 10, which focus on the best performing attacks, Figure 8 shows the box plot analysis of the 'black-box' exploit anomaly rates. Similarly, Figure 9 provides the box plot analysis of the 'black-box' attack anomaly rates. In addition to showing the distribution of exploit and attack anomaly rates, Figures 8 and 9 further demonstrate that it is easier to evade Neural Network and Markov Model detectors, since the exploits generally achieve lower anomaly rates for them. This is due to the fact that Neural Network detector utilizes a compressed representation of system calls (i.e. sequences are compressed into a frequency distribution) and Markov Model only looks at the previous system call (i.e. window size of 2 as opposed to 6 and 9 for Stide and pH/pHsm, respectively).



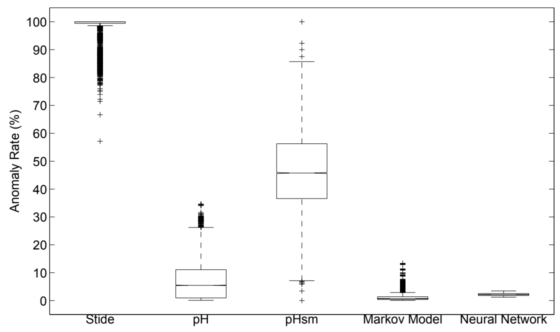
(a) traceroute



(b) restore

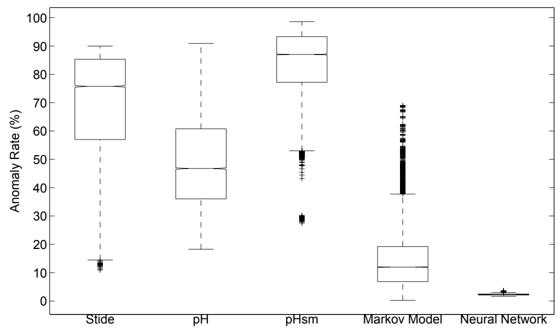


(c) samba

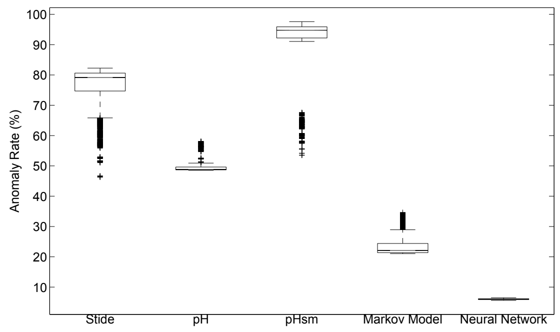


(d) ftpd

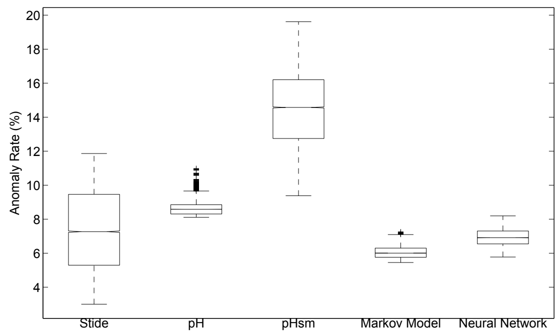
Fig. 8 Box plot analysis of 'black-box' exploit anomaly rates, plotted separately per application.



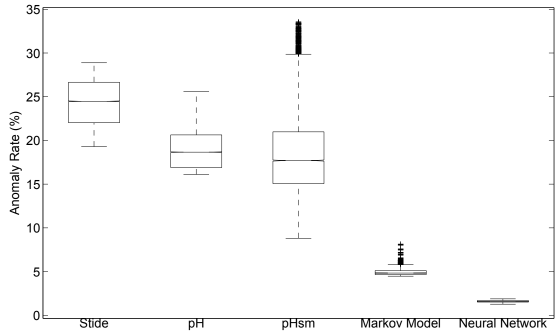
(a) traceroute



(b) restore



(c) samba



(d) ftpd

Fig. 9 Box plot analysis of 'black-box' attack anomaly rates *with preamble*, plotted separately per application.

6.2 Impact of detector delay counter measure

In this work, we observe that there are two parts to each attack, the preamble and the exploit. The preamble is composed of system calls that the application executes when attacker is trying to gain control. On the other hand, an exploit contains system calls after the attacker gains full control of the application. During the preamble phase, the attacker does not have full control over the application hence an attacker may not be able to prevent the vulnerable application from generating anomalous behaviour. The length and the anomaly rates of the preamble components for each vulnerable application – as summarized in Table 11 – indicates that preambles are typically long (with the exception of traceroute preamble, a few thousand system calls) and they are fairly anomalous. The previous work (specifically [35]) assumed that an attacker can gain control of the application silently. However, the above preamble analysis indicates that the preamble leaves anomalous system call traces, at least in the cases of traceroute, restore, samba and ftpd applications. The anomalous nature of the preambles may also impact on the (counter measure) delays that certain detectors enforce on the attacks.

Table 11 Details of the preamble components. Each application preamble has five anomaly rates, against each detector. The length of the preambles (no. of system calls) are provided in brackets under each application name.

	Stide	**pH**	**pHsm**	**Markov Model**	**Neural Network**
traceroute (53)	6.98%	36.49%	77.78%	8.54%	22.04%
restore (1,425)	77.82%	81.01%	93.67%	35.08%	13.29%
samba (3,868)	3.57%	9.97%	12.07%	6.78	6.34%
ftpd (2,601)	19.04%	21.94%	14.30%	6.11%	6.88%

Table 12 Length of the ‘white-box’ exploits generated against five anomaly detectors (no. of system calls).

	Stide	**pH**	**pHsm**	**Markov Model**	**Neural Network**
traceroute	77	38	158	97	736
restore	92	60	60	95	167,207
samba	213	91	90	124	65,648
ftpd	135	43	54	106	334,252

Table 13 Length of the ‘black-box’ exploits generated against five anomaly detectors (no. of system calls).

	Stide	**pH**	**pHsm**	**Markov Model**	**Neural Network**
traceroute	34	118	1,000	957	1,000
restore	1,000	1,000	999	1,000	1,000
samba	1,000	1,000	1,000	983	1,000
ftpd	11	1,000	994	1,000	1,000

Pareto ranking allows EEG to maintain solutions, which can optimize different objectives. Given that pH and pHsm provide delay feedback, EEG provides multiple best exploits against these detectors. Namely, two best exploits can be selected: (1) the exploit that minimizes the anomaly rate against the training detector – as reported in Section 6.1 – and (2) the exploit that minimizes the delay against the training detector.

Therefore, in addition to selecting the best ‘black-box’ exploits according to the anomaly rates, an additional ‘best’ exploit is selected according to delays, for pH and pHsm detectors. For all other detectors, best attacks are selected based only on the anomaly rates, as detailed in Section 6.1, since only pH and pHsm provide delay feedback. Thus, delays reported in Tables 14 and 15 include two best attacks for pH and pHsm. The delays provided in brackets are for exploits optimizing the delay and the remaining delays are for the attacks optimizing the anomaly rate.

The ‘white-box’ approaches focus on finding the first exploit with 0% anomaly rate; thus delays, which result from the anomalies in the exploits, are typically not considered as a part of the ‘white-box’ search. On the other hand, when an exploit with 0% anomaly rate is deployed against another target detector and particularly when the preamble is included; the anomaly rates and consequently delays become substantial.

Table 14 details the delays for both ‘black-box’ and ‘white-box’ exploits (no preamble), when the best performing individuals trained on a particular detector are tested against all five detectors. Similarly, Table 15 details the delays for the corresponding attacks (preamble plus exploit). Although Stide keeps track of the locality frame count, pH and pHsm employs the frame count to delay the processes. Hence, the delays are reported only for pH and pHsm. The locality frame count keeps track of the anomalies recorded over a period (by default, the previous 128 system calls). Therefore, a cluster of anomalies produce high locality frame counts causing long delays whereas the same number of anomalies distributed over a longer time frame (i.e. more system calls) produces smaller locality frame counts, causing shorter delays.

‘Black-box’ Exploits			**‘White-box’ Exploits**		
	pH	**pHsm**		**pH**	**pHsm**
Target: Training conducted on Stide			*Target: Exhaustive search conducted on Stide*		
traceroute	0.27	0.16	traceroute	0.69	0.58
restore	9.99	17.81	restore	0.84	0.73
samba	10.04	9.86	samba	246.86	5.00×10^{10}
ftpd	0.03	0	ftpd	1.27	1.16
Target: Training conducted on pH			*Target: Exhaustive search conducted on pH*		
traceroute	1.11 (0.01)	1.00 (0.00)	traceroute	0	0.19
restore	9.94 (9.94)	11.11 (11.11)	restore	0	0
samba	9.94 (9.94)	14.02 (14.02)	samba	0	0.72
ftpd	9.94 (9.94)	23.84 (23.84)	ftpd	0	0
Target: Training conducted on pHsm			*Target: Exhaustive search conducted on pHsm*		
traceroute	1.65×10^{35} (0.01)	1.65×10^{35} (0.00)	traceroute	0	0
restore	15.08 (10.03)	3.94×10^{7} (9.87)	restore	0	0
samba	3.88×10^{13} (0.10)	7.37×10^{12} (0.00)	samba	0	0
ftpd	6.44×10^{23} (0.01)	9.86×10^{17} (0.00)	ftpd	0	0
Target: Training conducted on Markov Model			*Target: Exhaustive search conducted on Markov Model*		
traceroute	3.17×10^{34}	1.10×10^{34}	traceroute	0.89	0.78
restore	6.29×10^{18}	6.52×10^{23}	restore	0.87	0.76
samba	2.02×10^{19}	2.00×10^{14}	samba	1.16	1.05
ftpd	9.06×10^{18}	7.52×10^{12}	ftpd	0.98	0.87
Target: Training conducted on Neural Network			*Target: Exhaustive search conducted on Neural Network*		
traceroute	2.22×10^{39}	1.86×10^{39}	traceroute	2.05×10^{39}	3.82×10^{38}
restore	5.04×10^{38}	5.66×10^{37}	restore	4.84×10^{22}	6.19×10^{24}
samba	9.26×10^{35}	9.19×10^{34}	samba	9.49×10^{40}	9.49×10^{40}
ftpd	1.84×10^{28}	3.46×10^{22}	ftpd	5.66×10^{41}	7.32×10^{38}

Table 14 Delays (in seconds) of the ‘black-box’ and ‘white-box’ exploits (*no preamble*) tested on the Stide, pH, pHsm, Markov Model and Neural Network detectors. The exploit delays are categorized according to the target detectors. For pH and pHsm, delays reported in brackets are for the best exploits optimizing the delay. All the other delay measurements are for the best exploits optimizing the anomaly rate.

pH and pHsm respond to attacks by delaying the process based on the observed locality frame count. Specifically, the delay is expressed in terms of $df \times 0.01 \times 2^{LFC}$, where the higher *df* (delay factor, which is equal to 1 by default) can cause longer delays and LFC signifies how many of the past 128 system calls were anomalous. Even a slight increase in locality frame count is sufficient to stop an attack since its effect on delay is exponential and it remains high until the locality frame moves beyond the anomalous segment of system calls.

When the delays associated with exploits are compared with the attack delays, it becomes apparent that the anomalous nature of the preambles cause most of the attacks to be delayed substantially. While the corresponding exploits are only delayed by a few seconds, Table 14, the attacks are delayed by centuries (any delay above 10^{15} is expressed in centuries) as shown in Table 15. This implies that although the exploits can achieve close to 0% anomaly rates, anomalies from the preamble can generate clusters and prevent the attack from deploying. Sustained high locality frame counts from preambles enforces high delays hence increasing the overall delay. Therefore, once the locality frame rises above a certain value (e.g. if the locality frame count rises to 120, the associated delays will be close to 10^{35} seconds), pH and pHsm effectively freezes the attack hence preventing the successful execution of the exploit.

Furthermore, the best attacks according to delay demonstrate that Pareto ranking allows EEG to discover exploits with shorter delays. For example, best samba ‘black-box’ exploits against pHsm in Table 14 show that the attacker has the option to choose the attack with 0.1 second delay over the attack with 3.88×10^{13} second delay, achieving a substantial gain in exploit delays. However, one should note that the resulting attacks (detailed in Table 15) indicate that the anomalous samba preamble causes delays in the magnitude of 7.95×10^{27} seconds regardless of the exploit delay.

The delays associated with the attacks demonstrate that the detector counter measure is clearly very effective. Even though the exploit achieves low anomaly rates, it can be frozen effectively if the anomalies are clustered together. In particular, ‘white-box’ attacks generally produce low exploit delays yet the delays associated with the corresponding attacks are expressed in

‘Black-box’ Attacks			‘White-box’ Attacks		
	pH	**pHsm**		**pH**	**pHsm**
Target: Training conducted on Stide			*Target: Exhaustive search conducted on Stide*		
traceroute	0.8	0.69	traceroute	1.22	1.11
restore	1.90×10^{38}	2.16×10^{39}	restore	1.90×10^{38}	4.33×10^{38}
samba	7.95×10^{27}	1.01×10^{21}	samba	7.95×10^{27}	1.01×10^{21}
ftpd	5.26×10^{30}	4.59×10^{31}	ftpd	5.26×10^{30}	7.84×10^{17}
Target: Training conducted on pH			*Target: Exhaustive search conducted on pH*		
traceroute	6.39×10^{6} (0.54)	3.52×10^{11} (0.43)	traceroute	0.83	0.72
restore	1.90×10^{38} (1.90×10^{38})	2.17×10^{39} (2.17×10^{39})	restore	1.90×10^{38}	4.33×10^{38}
samba	7.95×10^{27} (7.95×10^{27})	1.95×10^{28} (1.95×10^{28})	samba	7.95×10^{27}	1.01×10^{21}
ftpd	5.26×10^{30} (5.26×10^{30})	4.59×10^{31} (4.59×10^{31})	ftpd	5.26×10^{30}	7.84×10^{17}
Target: Training conducted on pHsm			*Target: Exhaustive search conducted on pHsm*		
traceroute	1.65×10^{35} (0.54)	1.65×10^{35} (0.43)	traceroute	9.97×10^{4}	4.85×10^{7}
restore	1.90×10^{38} (1.90×10^{38})	2.16×10^{39} (4.04×10^{38})	restore	1.90×10^{38}	4.33×10^{38}
samba	7.95×10^{27} (7.95×10^{27})	1.95×10^{28} (1.59×10^{20})	samba	7.95×10^{27}	1.01×10^{21}
ftpd	5.26×10^{30} (5.26×10^{30})	4.59×10^{31} (1.98×10^{27})	ftpd	5.26×10^{30}	7.84×10^{17}
Target: Training conducted on Markov Model			*Target: Exhaustive search conducted on Markov Model*		
traceroute	3.17×10^{34}	1.10×10^{34}	traceroute	8.11×10^{29}	3.89×10^{32}
restore	1.90×10^{38}	2.16×10^{39}	restore	1.90×10^{38}	4.35×10^{38}
samba	7.95×10^{27}	1.95×10^{28}	samba	1.32×10^{32}	1.30×10^{32}
ftpd	5.26×10^{30}	4.59×10^{31}	ftpd	5.24×10^{32}	2.60×10^{31}
Target: Training conducted on Neural Network			*Target: Exhaustive search conducted on Neural Network*		
traceroute	2.23×10^{39}	1.88×10^{39}	traceroute	2.07×10^{39}	4.40×10^{38}
restore	7.01×10^{38}	2.41×10^{39}	restore	1.90×10^{38}	4.33×10^{38}
samba	9.48×10^{35}	1.13×10^{35}	samba	9.49×10^{40}	9.49×10^{40}
ftpd	5.58×10^{30}	4.60×10^{31}	ftpd	5.66×10^{41}	7.32×10^{38}

Table 15 Delays (in seconds) of the ‘black-box’ and ‘white-box’ attacks (*preamble plus exploit*) tested on the Stide, pH, pHsm, Markov Model and Neural Network detectors. The attack delays are categorized according to the target detectors. For pH and pHsm, delays reported in brackets are for the best exploits optimizing the delay. All the other delay measurements are for the best exploits optimizing the anomaly rate.

centuries. On the other hand, if delays associated with locality frame counts will be employed in the real-world, reducing the false positive for the detector deserves further attention since legitimate behaviour, which is unknown to the detector can cause substantial delays.

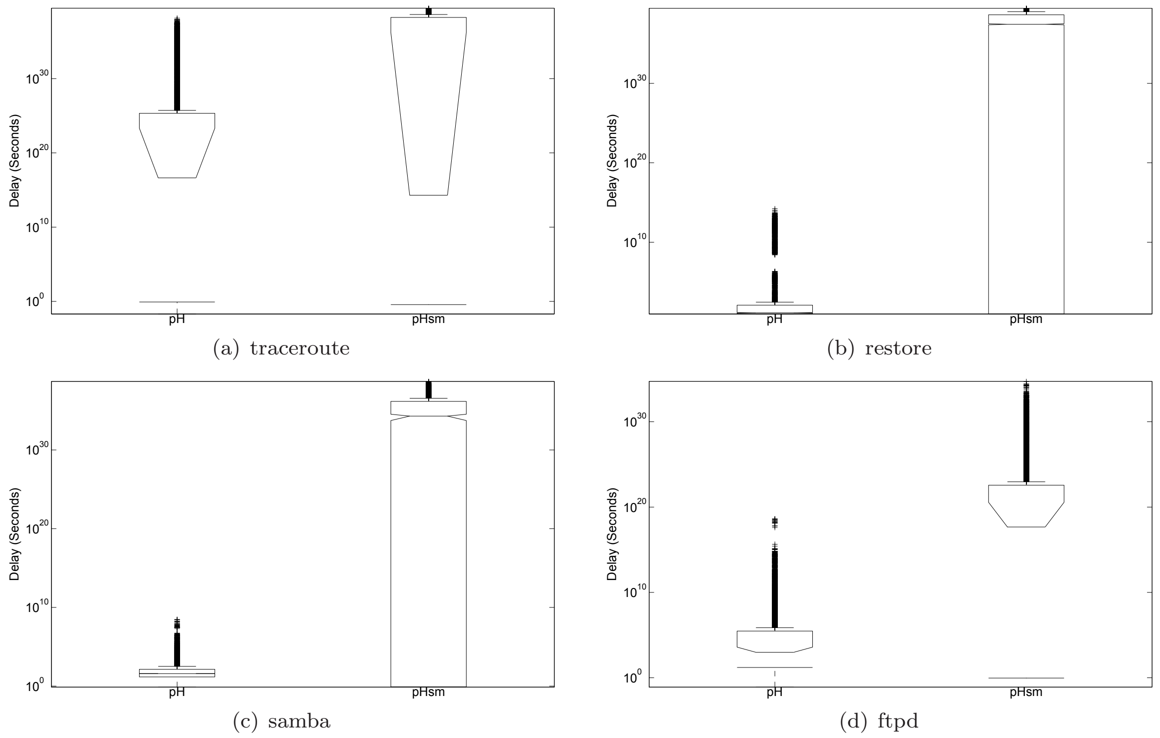
Figures 10 and 11 provide box plot analysis of exploit and attack delays, respectively. Since only pH and pHsm reports delays, the box plots are provided only for these detectors. In addition to showing the distribution of exploit and attack delays, the box plot underlines that pHsm is much more difficult to evade, a property most likely due to the unknown parameterization of the (pHsm) schema mask.

7 Conclusion

In this paper, we compare evasion attacks generated by two assumptions: ‘white-box’ assumption – adopted by the previous work – which assumes that the internal knowledge of the detector knowledge is present, and ‘black-box’ assumption – adopted by our work – which assumes that only the output of the detector is accessible, which presents a more practical scenario. The objective of the analysis is to investigate how much ‘internal’ knowledge the attacker needs to deploy successful attacks.

To do so, each attack (both ‘white-box’ and ‘black-box’) was deployed against different detector configurations and the results were provided in terms of anomaly rates, delays and exploit lengths. Evasion attacks generated by EEG provide anomaly rates comparable to the anomaly rates of ‘white-box’ attacks although the access to the detector is limited to the anomaly rate alone. Such a ‘black-box’ assumption is particularly suitable if the attacker does not posses internal knowledge of the detector.

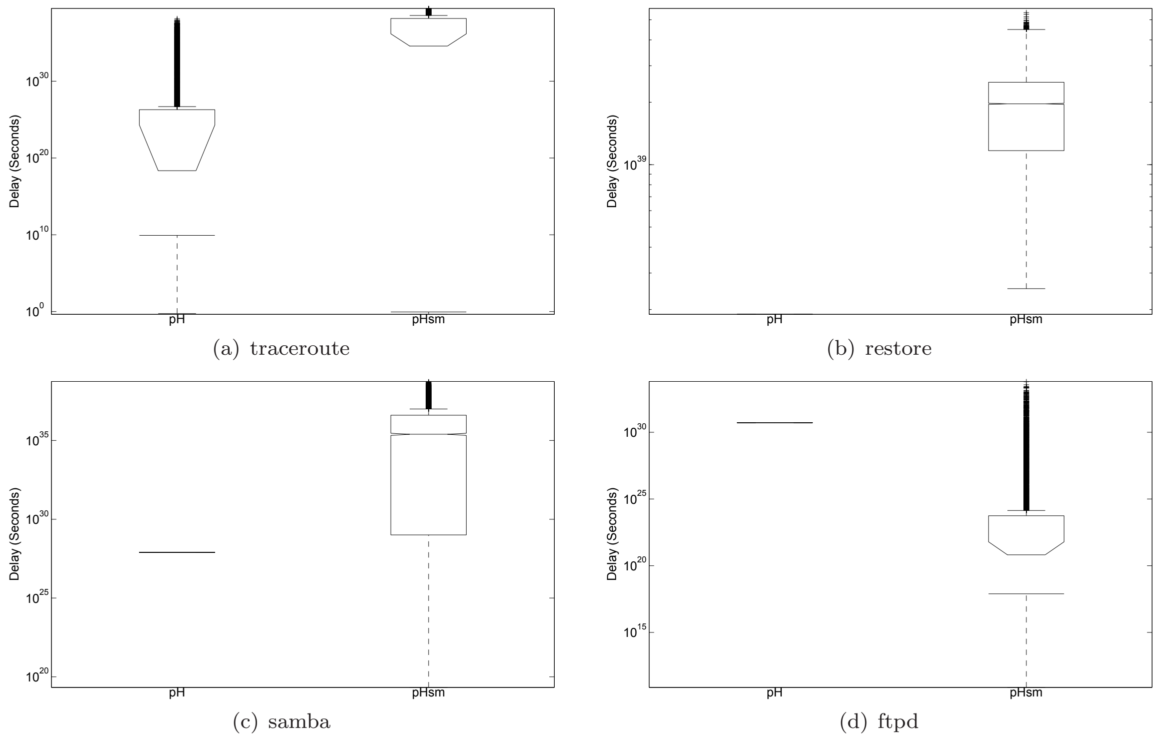
The results demonstrated that assuming a ‘black-box’ approach where the access to the detector is limited to the detector output, EEG succeeds in reducing the anomaly rate of the attacks. While ‘white-box’ techniques produce lower anomaly rates on the exploit alone, ‘black-box’ technique produces comparable attack anomaly rates by generating longer exploits, which reduce the anomalous effects of preambles. Although assuming a ‘black-box’ access presents a substantially more difficult problem, limiting the detector knowledge to the ‘black-box’ level does not prevent the identification of attacks, which are equally effective as a ‘white-



(a) traceroute (b) restore

(c) samba (d) ftpd

Fig. 10 Box plot analysis of 'black-box' exploit anomaly delays, plotted separately per application.



(a) traceroute (b) restore

(c) samba (d) ftpd

Fig. 11 Box plot analysis of 'black-box' attack anomaly delays *with preamble*, plotted separately per application.

box' model for the same detector. Moreover, when a 'white-box' access is assumed, exhaustive search methods are customized per detector to be able to facilitate an effective search, which implies the 'white-box' techniques are very detector dependent. Thus, assuming a 'black-box' access extends the application of vulnerability testing beyond the cases where internal knowledge of the detector is required to custom-design a 'white-box' search per detector.

Our analysis indicated that evading anomaly detectors can be more difficult than previously believed due to attacker's lack of control over the preamble component; specifically, system calls that execute before the exploit is deployed. While the previous work [35] [33] [32] [12] [24] [14] reported anomaly rates on the exploit alone without considering the anomaly rate of the preamble, our results demonstrated that even if the attacker can create an exploit with 0% anomaly rate, the corresponding attacks, which included the anomalies from the preamble, would be more than 0%. The effects of preamble is magnified especially if the preamble is long and anomalous, which also affects the delays.

We investigated whether the evasion attacks could generalize to the detectors for which they are not targeted. In other words, how likely is an attack trained against one detector to remain undetected when deployed against another detector? Thus, to answer this question, as opposed to deploying the attacks only against the detector on which they are generated, we deployed the attacks against all five anomaly detectors. The results indicated that, to a certain extent, evasion attacks can produce low anomaly rates when they are deployed against other detectors, especially if the detection mechanism is similar and if the detector on which they are trained employed a longer sliding window length than the detector on which they are tested (e.g. attacks trained for pH deployed against the Markov Model detector).

Additionally, as opposed to the previous work, which reported anomaly rates alone, our experiments reported the delays associated with the evasion attacks and demonstrated that a delay associated with locality frame counts can be an effective way to stop an attack. Even if the attack achieves low anomaly rates, it can be delayed by years, if the anomalies are sufficiently clustered together. In particular, although both 'black-box' and 'white-box' techniques succeeded in producing attacks with low anomaly rates, the delays are commonly expressed in years and centuries. Therefore, by incorporating additional metrics such as the dispersion of anomalies (using the locality frame count) in pH, the detectors can be more robust against evasion attacks. Nevertheless, should the delays be employed in real-world, potential delays enforced on legitimate actions can cause false alarms and deserve further investigation.

Acknowledgements The authors gratefully acknowledge the support of SwissCom Innovations SA., Telecom and Research Alliance (TARA) Inc., Killam, Precarn, MITACS and Natural Sciences and Engineering Research Council (NSERC) funding programs.

References

1. (Last accessed August 2010) Securityfocus vulnerability archives – lbnl traceroute heap corruption vulnerability. `http://www.securityfocus.com/bid/1739`
2. (Last accessed August 2010) Securityfocus vulnerability archives – redhat linux restore insecure environment variables vulnerability. `http://www.securityfocus.com/bid/1914`
3. (Last accessed August 2010) Securityfocus vulnerability archives – samba 'call_trans2open' remote buffer overflow vulnerability. `http://www.securityfocus.com/bid/7294`
4. (Last accessed August 2010) Securityfocus vulnerability archives – wu-ftpd remote format string stack overwrite vulnerability. `http://www.securityfocus.com/bid/1387`
5. (Last accessed August 2010) Stide website – source code of stide and system call data sets. `http://www.cs.unm.edu/~immsec/systemcalls.htm`
6. Banzhaf W, Francone FD, Keller RE, Nordin P (1998) Genetic programming: an introduction: on the automatic evolution of computer programs and its applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA
7. Baum LE, Petrie T, Soules G, Weiss N (1970) A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. The Annals of Mathematical Statistics 41(1):164–171, DOI 10.2307/2239727
8. Bishop CM (1995) Neural Networks for Pattern Recognition. Oxford University Press, Inc., New York, NY, USA
9. Chambers JM, Cleveland WS, Tukey PA (1983) Graphical methods for data analysis. Wadsworth
10. Deb K (2001) Multi-Objective Optimization using Evolutionary Algorithms. John Wiley and Sons
11. Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA (1996) A sense of self for unix processes. In: SP '96: Proceedings of the 1996 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, p 120

12. Gao D, Reiter MK, Song D (2004) Gray-box extraction of execution graphs for anomaly detection. In: CCS '04: Proceedings of the 11th ACM conference on Computer and communications security, ACM, New York, NY, USA, pp 318–329, DOI http://doi.acm.org/10.1145/1030083.1030126
13. Gao D, Reiter MK, Song D (2006) Behavioral distance measurement using hidden markov models. In: Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection - RAID, Lecture Notes in Computer Science, LNCS 4219, pp 19–40
14. Giffin JT, Jha S, Miller BP (2006) Automated discovery of mimicry attacks. In: Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006, Springer, Lecture Notes in Computer Science, vol 4219, pp 41–60
15. Inoue H, Somayaji A (June 2007) 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), pp 9–19
16. Japkowicz N, Myers C, Gluck M (1995) A novelty detection approach to classification. In: Proceedings of the Fourteenth Joint Conference on Artificial Intelligence, pp 518–523
17. Kang DK, Fuller D, Honavar V (2005) Learning classifiers for misuse and anomaly detection using a bag of system calls representation. Information Assurance Workshop, 2005 IAW '05 Proceedings from the Sixth Annual IEEE SMC pp 118–125, DOI 10.1109/IAW.2005.1495942
18. Kayacık HG (2009) Can the best defense be a good offense? evolving (mimicry) attacks for detector vulnerability testing under a black-box assumption. PhD thesis, Dalhousie University
19. Kayacık HG, Zincir-Heywood AN (2008) 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, IEEE Computer Society, Washington, DC, USA, pp 213–223, DOI http://dx.doi.org/10.1109/PST.2008.25
20. Kayacık HG, Heywood M, Zincir-Heywood N (2006) On evolving buffer overflow attacks using genetic programming. In: Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO), SIGEVO, ACM, pp 1667–1674
21. Kayacık HG, Heywood M, Zincir-Heywood N (2007) Evolving buffer overflow attacks with detector feedback. In: Proceedings of the EvoWorkshops (EvoCOMNET), Springer, LNCS, vol 4448, pp 11–20
22. Kayacık HG, Zincir-Heywood AN, Heywood M, Burschka S (2009) Optimizing anomaly detector deployment under evolutionary black-box vulnerability testing. In: Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on, pp 1 –8, DOI 10.1109/CISDA.2009.5356546
23. Kramer MA (1991) Nonlinear principal component analysis using autoassociative neural networks. AIChE Journal pp 233–243
24. Kruegel C, Kirda E, Mutz D, Robertson W, Vigna G (2005) Automating mimicry attacks using static binary analysis. In: SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium, USENIX Association, Berkeley, CA, USA, pp 161–176
25. Kumar R, Rockett P (2002) Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution: a pareto converging genetic algorithm. Evolutionary Computation 10(3):283–314, DOI http://dx.doi.org/10.1162/106365602760234117
26. Lee J, Cho S, Baek J (2003) Trend detection using auto-associative neural networks: Intraday kospi 200 futures. Computational Intelligence for Financial Engineering, 2003 Proceedings 2003 IEEE International Conference on pp 417–420, DOI 10.1109/CIFER.2003.1196290
27. Manevitz L, Yousef M (2007) One-class document classification via neural networks. Neurocomput 70(7-9):1466–1481, DOI http://dx.doi.org/10.1016/j.neucom.2006.05.013
28. Sekar R, Bendre M, Dhurjati D, Bollineni P (2001) A fast automaton-based method for detecting anomalous program behaviors. In: SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, p 144
29. Somayaji AB (2002) Operating system stability and security through process homeostasis. PhD thesis, The University of New Mexico, chairperson: Stephanie Forrest
30. Tan KMC, Maxion RA (2002) "why 6?" defining the operational limits of stide, an anomaly-based intrusion detector. In: SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, p 188
31. Tan KMC, Maxion RA (2003) Determining the operational limits of an anomaly-based intrusion detector. Selected Areas in Communications, IEEE Journal on 21(1):96–110, DOI 10.1109/JSAC.2002.

806130

32. Tan KMC, Killourhy KS, Maxion RA (2002) Undermining an anomaly-based intrusion detection system using common exploits. In: Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection - RAID, Lecture Notes in Computer Science, LNCS 2516, pp 54–73
33. Tan KMC, McHugh J, Killourhy KS (2003) Hiding intrusions: From the abnormal to the normal and beyond. In: IH '02: Revised Papers from the 5th International Workshop on Information Hiding, Springer-Verlag, London, UK, pp 1–17
34. Vigna G, Robertson W, Balzarotti D (2004) Testing network-based intrusion detection signatures using mutant exploits. In: CCS '04: Proceedings of the 11th ACM conference on Computer and communications security, ACM, New York, NY, USA, pp 21–30, DOI http://doi.acm.org/10.1145/1030083.1030088
35. Wagner D, Soto P (2002) Mimicry attacks on host-based intrusion detection systems. In: CCS '02: Proceedings of the 9th ACM conference on Computer and communications security, ACM, New York, NY, USA, pp 255–264, DOI http://doi.acm.org/10.1145/586110.586145