Generating Mimicry Attacks using Genetic Programming: A Benchmarking Study

H. Güneş Kayacık[1], A. Nur Zincir-Heywood[1], Malcolm I. Heywood[1], Stefan Burschka[2]
[1] Dalhousie University, Faculty of Computer Science,
6050 University Avenue, Halifax, Nova Scotia. B3H 1W5 Canada
[2] Software & Security Technologies, Swisscom Innovations, Switzerland

***Abstract*—Mimicry attacks have been the focus of detector research where the objective of the attacker is to generate multiple attacks satisfying the same generic exploit goals for a given vulnerability. In this work, multi-objective Genetic programming is used to establish a "black-box" approach to mimicry attack generation. No knowledge is made of internal data structures of the target anomaly detector, only the anomaly rate reported by the detector. Such a "black box" methodology enables a vulnerability testing approach where both open-source and commodity anomaly detection systems can be tested. The approach successfully identifies exploits when benchmarked over four detectors and four applications.**

I. Introduction

Penetration testing for Intrusion Detection Systems (IDS) is a relatively new area compared with similar tests on cryptographic protocols. The main idea behind penetration testing is to locate vulnerabilities or holes in a system before attackers exploit them. Thus, administrators can test their IDS to establish the robustness of such systems in real world situations by using the same tools and methodologies as used by attackers.

Several researchers have previously identified a number of evasion attacks on network based IDSs with misuse detection [1-4], and host based IDSs with anomaly detection [5-9]. In these works, the general approach employed is based on the generation of mimicry attacks to perform evasion. To generate mimicry attacks, the original exploit is reestablished by producing a legitimate sequence of system calls while performing malicious actions (e.g., add the attacker to the system password file). Consequently, the resulting mimicry attack remains within the bounds of normal behavior (e.g., statistical characterization) and deploys undetected by the detector. Moreover, mimicry attack research typically assumes "white box" access to detector operation. This has resulted in very efficient algorithms for designing mimicry exploits. Under such an assumption, the design of exploits concentrates on locating sequences of system calls that both match the contents of a detector's database of normal behaviors whilst successfully reaching the behavioral objectives of the exploit. On the other hand, this work takes a "black box" approach to mimicry attack generation. As such the feedback from the detector is limited to the anomaly rate alone, where this is readily provided to the user as part of normal operation. Hence no use is made of the internal data structures or algorithms specific to a particular detector. Design of mimicry exploits now takes the form of a search process in which the anomaly rate from the detector is used as the only guide to the effectiveness of the exploit.

We believe that assuming a "black-box" approach is more suitable for testing both commodity and the open source detectors, particularly if the detector developer is unable or unwilling to make public the internal data structures with the tester. We aim to demonstrate that although representing a more difficult problem, limiting knowledge of the detector to the level of "black box" feedback does not prevent the identification of equally effective attacks as a "white box" model of the same detector. We automate the process of generating mimicry attacks using Genetic Programming (GP). Throughout the search process, GP maintains a population of candidate solutions. Each candidate solution or individual takes the form of a program (i.e. the buffer overflow attack). Programs were represented as a sequence of system calls where the set of permitted system calls was derived from the application (again restricting access to public information alone). The search process progresses through the iterative application of GP search operators. Performance (fitness) of an attack is evaluated using a fitness function, which ranks the population according to attack success and anomaly rate and other attack characteristics.

In addition, we recognize that there are two parts to an attack: the exploit itself, and the activities necessary to gain control of the vulnerable application. In a buffer overflow exploit, the first step is to corrupt the data types and local variables, which gives the attacker the control of the application. We call the actions taken by the attackers before they gain full control of the application as the preamble. After the attacker gains control of the application, the second step (i.e. the exploit) involves executing arbitrary code or command to carry out a malicious action such as spawning a root shell or creating a super-user account. Commonly, this is achieved by injecting a shellcode. Previous work assumed that the attack was optimal if the exploit component raised no alarms [5-9]. However, even when the exploit raises no alarms, introducing the preamble can introduce alarms for both the preamble itself and the transition between the preamble and exploit. Thus, in the following work, the preamble is explicitly incorporated, where this is shown to have a significant impact on the ability to identify alternative exploits that evade detection.

In the following, Section II details the proposed genetic programming framework while also detailing anomaly

detectors and vulnerable applications employed in our evaluation, thus placing the proposed approach in the wider context of mimicry attack research. The results of our experiments are reported in Section III and the conclusions are drawn in Section IV.

II. Methodology

Our objective is to develop an automated process for building "white-hat" attackers within a mimicry attack context. By 'mimicry' we assume the availability of the exploit goals, where this establishes a series of behavioral objectives associated with the exploit. The objective of the automated "white-hat" attacker will therefore be to establish as many specific attacks corresponding to the exploit associated with the attack goals. Candidate mimicry attacks will take the form of system call sequences that minimize anomaly rate as measured by the detector.

Previous research has established the suitability of the genetic programming (GP) machine learning paradigm as an appropriate process for automating specific processes associated with the design of buffer overflow attacks [4], [10]. In this work, we extend the approach to a general framework for mimicry attack generation, based on the evolution of system call sequences rather than generic parameters of an attack. The general motivation for using GP within this context are as follows [11],

Goal Based Objectives: In the case of mimicry attack generation the goal is to discover programs (of system calls) that mimic the behavior of a predefined 'core' (buffer overflow) attack. Learning is therefore directed by information supplied following an interaction with an environment. The environment in this case takes two forms, the anomaly rate returned by detector and the number of generic exploit goals satisfied. Such a mode of operation precludes the vast majority of machine learning paradigms as they make explicit assumptions regarding the relationship between representation (the components from which a solution is built) and how the objective is specified [12].

Representation: Given that the objective is to design mimicry attacks, it follows that the machine learning representation must have the capacity to represent system call sequences that explicitly correspond to the attack.

Intron Code: Solutions from GP take the form of a program in a predefined language. However, not all instructions comprising a solution necessarily contribute to the underlying operation of the individual, where this artifact is synonymous with 'introns' of biological genomes. Such introns result from the stochastic action of the search operators (crossover and mutation) and might account for as much as 80% of the instructions in an individual [12]. Typically, instructions corresponding to intron behavior are removed post learning. In the context of this work, however, intron code aids obfuscation of the exploit and explicitly contribute to the alarm rate associated with the attack.

A. Anomaly Detectors

Anomaly detection systems attempt to build models of normal user behavior and use this as the basis for detecting suspicious activities. This way, known and unknown (i.e. new) attacks can be detected as long as the attack behavior deviates sufficiently from the normal behavior. Thus, if the (buffer overflow) attack is sufficiently similar to normal behavior, it may not be detected.

1) Stide

Forrest et al. [13] employed a methodology motivated by immune systems. This characterizes the problem as distinguishing 'self' from 'non-self' (normal and abnormal behaviors 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 fixed-length sequences formed by the sliding window are stored in a table that establishes the normal behavior model. During the deployment (detection) phase, if the pattern from the sliding window is not in the normal behavior database it is considered a mismatch. In our experiments, the default training parameters for Stide are employed i.e., a sliding window length of 6.

2) Process Homeostasis (pH)

Process Homeostasis (pH) [14] represents the second generation of the original Stide methodology. In particular, pH is implemented as an extension to Linux 2.2 Kernel where it monitors behavior more efficiently by capturing system calls directly at the kernel level (Stide utilized Strace to capture system calls). pH monitors the changes in short sequences of system calls by employing 'look ahead pairs.' The sliding window approach is retained, however, pH does not store the sliding window patterns but records tuples, which consist of the current and the past system calls and the sliding window location. Somayaji [14] established that such a look-ahead method is more efficient to store and could potentially converge to a normal profile quicker than the sequence method of Stide. Additionally, tolerization and sensitization concepts were introduced to allow pH to improve false alarm rates and reduce the likelihood of abnormal behavior from leaking into the normal behavior database [14]. In our experiments, we employed the default pH training parameters listed in Table I.

TABLE I.
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) Process Homeostasis (pH) with Schema Masks

Inoue and Somayaji [15] introduced the concept of random schema masks to improve pH. Specifically, a longer sliding window (usually 20 as opposed to 9) is maintained and a number of taps are taken from sliding window. The locations of the taps (or schema mask) are determined

randomly before training. The authors claim that introducing a schema mask creates another source of generalization. In addition to the configuration parameters detailed in Table I, additional configuration parameters of the pH with schema mask are detailed in Table II.

TABLE II.
PH WITH SCHEMA MASK CONFIGURATION PARAMETERS

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

4) Markov Model

Markov Model is a statistical modeling technique, which is frequently employed for building probabilistic models of event sequences. Markov Models have been utilized within the context of speech recognition and intrusion detection systems [16, 9, 17], as has the related case of a Finite State Automata [18]. In a first order Markov Model, the next state is only dependent on the current state; where such an assumption is widely employed in such systems to reduce the number of 'free parameters' that require estimation.

In order to establish values for the above model parameters, the Baum-Welsh model will be assumed [19]. Let I be a countable set of states, where $i, j \in I$ represent states. If there are N states in I, a first order Markov model can be represented as a two-dimensional $N \times N$ matrix $P = (p_{ij} \mid i, j \in I)$. Let X_t define the state at step t. From the training data, probability of transition from state i to j p_{ij} can be calculated as follows;

$$p(i,j) = \frac{C(X_t = j, X_{t-1} = i)}{\sum_i C(X_{t-1} = i)}$$

where $C(X_t = j, X_{t-1} = i)$ is the number of times state j follows state i in the training data. The i^{th} row of the P is the probability distribution of moving to all states from state i. This probability distribution is also called λ_i. Such a scheme was previously used to act as a generic model for TCP packets, with the objective of summarizing normal data [20].

For each test sequence, the Markov Model estimates the log likelihood as follows:

$$LL_{seq} = \sum_t \log(p(x_t, x_{t-1}))$$

Such a log likelihood estimate is dependent on the attack size and therefore not suitable for distinguishing between attack and normal behavior, we base our detection decision on a characterization of state transition behavior. This concept was employed in previous approaches to Markov Model detectors (e.g., [9] and Stide [21]). If a transition occurs in the test sequence, which was not encountered during training (hence $p(i,j) = 0$), a flag is set. We maintain a count of the number of times the flag is set, and define the anomaly rate as:

$$\text{Anomaly Rate} = \frac{\text{number of flags}}{\text{total number of transitions}}$$

Such an anomaly rate implies that, if the test sequence follows the training model (i.e. normal behavior), it will encounter a low number of flag conditions. Thus a low anomaly rate is assigned. Configuration parameters of the Markov Model detector are provided in Table III.

TABLE III.
MARKOV MODEL PARAMETERS

Parameter	Setting
Order	First order
Number of states	223 (Number of system calls)
Training Algorithm	Baum-Welsh

B. Vulnerable Applications

In the following experiments, four Linux applications are tested: namely Traceroute, Restore, FtpD, and Samba, all of which have known and documented vulnerabilities. These are also the vulnerable applications most frequently used in the mimicry attack literature [5, 6, 7, 8]. Traceroute, and Restore vulnerabilities can be exploited locally whereas FtpD and Samba vulnerabilities can be exploited remotely. For each application, we developed normal use cases, which represent the scenarios of legitimate use.

1) Traceroute

Traceroute is a network diagnosis tool, which is used to determine the routing path between a source and a destination by sending a set of control packets to the destination with increasing time-to-live values. A typical use of traceroute involves providing the destination IP, whereas the application returns information on the route taken between source and destination.

Redhat 6.2 is shipped with Traceroute version 1.4a5, where this is susceptible to a local buffer overflow exploit that provides a local user with super-user access [22]. The attack takes advantage of vulnerability in malloc chunk, and then uses a debugger to determine the correct return address to take control of the program. In order to analyze the traceroute behavior under normal conditions, we developed five use cases, Table IV; whereas in the previous research [6] only one normal use case was used for training, namely use case 1.

TABLE IV.
TRACEROUTE NORMAL USE CASES

Use Case	System Calls
1. Target a remote server	736
2. Target a local server	260
3. Target a non existent host	153
4. Target localhost	142
5. Help screen	24

2) Restore

Restore is a component of UNIX backup functionality, which restores the file system image taken by the dump command. Files or directories can be restored from full or incremental backups.

Restore version 0.4b15 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 [23], the attacker spawns a root shell. Table V summarizes five

normal use cases that we developed for Restore. As in the previous work [8], we monitored a regular user, retrieving backup data from a file system dump.

TABLE V.
RESTORE NORMAL USE CASES FROM A FILE SYSTEM DUMP (FSD)

Use Case	Sys Calls
1. Restore a small FSD from a full backup.	2256
2. Restore a small FSD from an incremental backup.	1027
3. Restore a large FSD from a full backup.	167207
4. Restore a large FSD from an incremental backup.	68185
5. Help screen	53

3) Samba

The Samba suite provides printer and file sharing for Windows clients and can run on most UNIX variants. Samba sets up printer and network shares that appear as Windows disks and printers under a Windows operating system.

Redhat 9.0 is shipped with Samba suite version 2.2.7a, which has a vulnerability [24] that can be exploited over the network to gain super-user privileges. The buffer overflow occurs when Samba service tries to copy user supplied data into a static buffer without checking. The published attack binds a root shell to a network port. Table VI summarizes the six normal use cases that we developed for Samba, To best of our knowledge, Samba has not been employed in the previous work for mimicry attack generation.

TABLE VI.
SAMBA NORMAL USE CASES

Use Case	Sys Calls
1. Mount a samba share successfully	1156
2. Invalid password while mounting samba share	680
3. Unmount a samba share successfully	186
4. Find and edit a remote file. (Using commands: ls - cd - ls - pico)	254
5. Find and copy a 38MB remote file to a local directory (Using commands: ls - cd - cp)	65648
6. Change samba password remotely	1527

TABLE VII.
FTPD NOORMAL USE CASES

Use Case	System Calls
1. Upload 10K data	2249
2. Upload 60M data	32912
3. Upload 650M data	334252
4. Download 10K data	2252
5. Download 60M data	32908
6. Download 650M data	334244
7. Three failed login attempts	2236
8. Help screen	2017
9. Attempt to access non-existent files and directories	2213
10. Type non-existent commands.	2017

4) FtpD

Redhat 6.2 is shipped with Washington University Ftp Server version 2.6.0(1), which provides FtpD 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 his/her shellcode. Although the attack is an input validation attack [25], the deployment is similar to a buffer overflow attack. Table VII summarizes the ten normal use cases that we developed for FtpD. Use cases 7 through 10 represents the legitimate errors that a user can make during a normal FtpD session. On the other hand, in the previous research [5] wuftpd was run on only large file downloads over a period of two days.

C. Linear Genetic Programming

The process for designing mimicry attacks will be automated using Genetic Programming (GP). The GP paradigm differs from most machine learning methodologies in that a 'population' of candidate solutions are maintained concurrently throughout the search process [12]. Each candidate solution, or individual, takes the form of a program. Programs are represented, in the case of this application, as a sequence of system calls (as opposed to program characteristics [4] or assembly instructions [10]). Although parameters for the system calls are specified, there is no need to support the specification of internal state i.e., register values.

The search process progresses through the iterative application of a selection operator, evaluation of performance associated with the subset of individuals targeted by the selection operator, and application of search operators. Specifically, the selection operator in this work takes the form of a steady state tournament. This means that an even number of individuals (4 in this work) is selected from the population of (many more) individuals with uniform probability. Performance (fitness) of this subset of individuals is evaluated, ranking the individuals participating in the tournament relative to each other. Search operators are then applied to the better performing half of the tournament, resulting in (2) children. The children overwrite the individuals of the worst half of the tournament, taking their place in the original population. Such a scheme is inherently elitist with the best individuals always surviving.

Individuals are defined using a variable length format, thus population initialization creates individuals with program varying program lengths. Search operators take three forms: crossover, instruction-wise mutation, and instruction swap. The specific details of each operator are as follows, however, all search operators are applied stochastically relative to a predefined probability of application, Table VIII.

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

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

Instruction-wise mutation: Mutation operator provides a way to introduce new sequences to the individual. Mutation

is applied instruction-wise, that is to say, each instruction is tested independently for modification. If the test returns true then the instruction is replaced with an alternative instruction, from a predefined list of instructions. Moreover, we also let the probability of applying the mutation operator decay linearly with tournament count [11]; thus lowering the likelihood of introducing instructions that are not currently in the population as the population evolves. This effectively places more emphasis on the crossover operator as evolution progresses, thus reinforcing the reuse of system call sequences that where earlier demonstrated to minimize detection.

TABLE VIII.
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 the normal use cases
Program Length	Initialized over 240 system calls, maximum 1000 system calls

The principle GP design decisions are now limited to defining the instruction set (representation) and appropriate goal (fitness function) and a method to achieve multiple objectives.

Tables IX, X, XI and XII detail the instruction sets for Traceroute, FtpD, Restore and Samba applications, respectively. Furthermore, Tables IX, X, XI and XII also contain the occurrences of the most frequently used 20 system calls as executed by the applications. Such an analysis merely entails logging the system calls made by the application. No 'internal' knowledge of the application is necessary. In case of traceroute (Table IX) it is apparent that the top 20 system calls can cover over 90% of the executed instructions. On the other hand in case of FtpD and Restore (Table X and XI) most frequent 2 to 4 instructions cover 99% percent of the executed instructions.

TABLE IX.
GP INSTRUCTION SET FOR THE TRACEROUTE APPLICATION

System Call	Percentage	System Call	Percentage
gettimeofday	16.73%	mprotect	2.59%
write	10.80%	socket	2.21%
mmap	8.59%	recvfrom	2.13%
select	7.53%	brk	2.05%
sendto	7.53%	fcntl	1.98%
close	7.07%	connect	1.52%
open	6.54%	ioctl	1.14%
read	5.70%	uname	1.06%
fstat	5.55%	getpid	0.91%
munmap	3.73%	time	0.76%

TABLE X.
GP INSTRUCTION SET FOR THE FTPD APPLICATION

System Call	Percentage	System Call	Percentage
read	24.45%	chdir	0.05%
t_sigaction	24.39%	fcntl	0.04%
alarm	24.36%	getcwd	0.04%
write	24.35%	socket	0.03%
close	1.53%	connect	0.03%
open	0.14%	fchdir	0.03%
time	0.14%	mprotect	0.03%
mmap	0.13%	lstat	0.03%
fstat	0.10%	send	0.03%
munmap	0.07%	brk	0.03%

TABLE XI.
GP INSTRUCTION SET FOR THE RESTORE APPLICATION

System Call	Percentage	System Call	Percentage
write	88.698%	fcntl	0.010%
read	11.052%	rt_sigprocmask	0.010%
lseek	0.055%	utime	0.006%
mmap	0.038%	unlink	0.003%
open	0.028%	chmod	0.003%
close	0.023%	chown	0.003%
fstat	0.021%	ioctl	0.003%
mprotect	0.013%	stat	0.003%
munmap	0.012%	rt_sigaction	0.003%
brk	0.010%	_llseek	0.003%

TABLE XII.
GP INSTRUCTION SET FOR THE SAMBA APPLICATION

System Call	Percentage	System Call	Percentage
read	41.03%	mmap	0.18%
_llseek	13.92%	mmap2	0.18%
select	13.61%	getegid32	0.14%
gettimeofday	13.61%	geteuid32	0.14%
send	13.60%	time	0.13%
fcntl64	1.64%	getsockopt	0.10%
stat	0.64%	mprotect	0.07%
open	0.32%	umask	0.06%
close	0.30%	setresgid32	0.06%
munmap	0.22%	write	0.05%

D. Fitness Calculation and Pareto Ranking

Pareto Ranking is a method for combining multiple, possibly conflicting objectives under the concept of dominance [26]. Specifically in the case of a problem where objectives are being minimized, solution A dominates solution B ($A \prec B$), if and only if:

$$A \prec B \Leftrightarrow (\forall_O)(A_O \le B_O) \wedge (\exists_O)(A_O < B_O)$$

An individual that is not dominated by any other individual is called a non-dominated individual. Pareto 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. We employed a ranking with ties algorithm where the rank of the individual equals to the number of individuals that it dominates [26]. In such a scheme, the performance of the individuals are calculated based on the following attack characteristics:

Attack Success: The original attack contains a standard shellcode, which uses the execve system call to spawn a UNIX 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 traceroute, restore, samba and ftp, we would expect the original attack to be easily detected. To this end, we employ a different strategy for defining the exploit such that the need to spawn a UNIX shell is eliminated [11]. Most programs typically perform I/O operations, in particular open, write to / read from and close files. Tables IX, X, XI and XII demonstrate that UNIX applications frequently uses open / write / close system calls. We therefore recognize that performing the following three steps mimics the goals of the original shell code attack: (1) Open the UNIX password file (/etc/passwd); (2) Write a line, which provides the attacker a super-user account that can login without a password; and (3) Close the file.

The objective of the search process conducted by the GP is to discover a sequence of system calls (and appropriate arguments) that perform the above three steps in the correct order (i.e. the attack cannot write to a file that it has not opened) while minimizing the anomaly rate from the detector. A behavioral success function rewarding the above behavior awards a total of 5 'points' for establishing the behavioral steps of the 'core' attack, Algorithm 1.

Algorithm 1. Behavioral success function

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

Anomaly Rates: Anomaly rate represents the principle metric for qualifying the likely intent of a system call sequence, a would be attacker naturally wishes to minimize the anomaly rate of the detector. Again, no 'inside knowledge' is necessary as detectors provide alarm rates as part of their normal mode of operation. Moreover, as indicated in the introduction, the attacker also needs to minimize the anomaly rate of the exploit. Provided that the preamble is not highly anomalous, relatively low attack anomaly rates can be accomplished by utilizing very concise exploits. However this is not always the case, thus we evolve exploits with preamble appended to facilitate identification of the most appropriate content.

Attack Length: Attack length is not an immediate concern for the attacker; longer attacks potentially provide more obfuscation. However, attack length appears as an objective to encourage GP to perform a wider search for solutions i.e., short solutions will be included as well as longer under the Pareto methodology.

III. Results

In order to establish the effectiveness of the mimicry attack generation methodology, we first need to determine the anomaly rate of the original attacks. To do so, we downloaded the original attacks [22, 23, 24, 25] from SecurityFocus website and deploy it against the detector configurations on the four vulnerable applications detailed in Sections II.A and II.B. Table XIII details the anomaly rates reported for the original attacks.

TABLE XIII.
ANOMALY RATE (%) OF THE ORIGINAL ATTACKS: PREAMBLE + EXPLOIT

	Stide	pH	pHsm	Markov Model
traceroute	61.26	66.27	95.06	38.78
Restore	84.69	87.49	99.55	44.26
Samba	10.16	16.02	16.67	9.03
ftp	22.78	25.54	44.31	7.15

As discussed in Section II.A, pH with schema mask has two distinct properties relative to pH. First, it has a longer sliding window and second, the attacker needs to "guess" the schema mask parameter. With this in mind, two deployment scenarios were considered. In the first scenario, the attacker does not know the schema mask employed, whereas in the second scenario he does. This way, we aim to identify the characteristic that makes the pH with schema mask more resistant against mimicry attacks. Therefore, the anomaly rates on the GP-formed mimicry attacks against pH with schema masks (Tables XIV and XVII) incorporate the two scenarios.

In a Pareto framework, multiple solutions are maintained; hence multiple mimicry attacks are generated. Table XIV details the anomaly rates of the mimicry attacks that produced the least anomaly rate during training in our framework. It is apparent that mimicry attacks produce smaller anomaly rates than the original attacks. For example, pH with schema mask detects the original traceroute attack (i.e. preamble + exploit) with 95.06% anomaly rate whereas the GP-formed attack produces only 2.70% anomaly rate, if the attacker possesses the schema mask knowledge. Even without the schema mask knowledge GP-formed attack achieves a 29.28% anomaly rate, still lower than the original attack and albeit higher than the scenario where the schema mask is known. Furthermore, we also emphasize that the framework is employed on all detector configurations with the same settings. Our previous results show that it is possible to enhance the results by using different training settings for different configurations. For example, although the anomaly rate Stide reports for the traceroute GP-formed mimicry attack is 10.96%, it can be further improved by using greedy search operators [11].

TABLE XIV.
ANOMALY RATE (%) OF THE BEST GP MIMICRY ATTACKS: PREAMBLE + EXPLOIT

	Stide	pH	pHsm (mask known)	pHsm (mask unknown)	Markov Model
traceroute	10.96	18.29	2.70	29.28	0.20
restore	46.25	48.57	54.52	57.92	21.05
samba	3.00	8.11	7.36	15.84	5.45
ftp	19.30	16.11	10.62	20.19	4.47

As discussed in Section I, attacks consist of two components, exploit and the preamble. Table XV shows that the exploits employed by the original attacks are detectable with fairly high anomaly rates whereas exploits identified by

the mimicry attacks have lower anomaly rates, Table XVI.

TABLE XV.
ANOMALY RATE (%) OF THE ORIGINAL EXPLOITS

	Stide	pH	pHsm	Markov Model
traceroute	71.48	73.91	94.21	47.89
Restore	88.13	90.70	99.63	48.84
Samba	60.04	60.51	44.78	25.53
ftp	47.52	47.85	67.71	13.65

TABLE XVI.
ANOMALY RATE (%) OF THE BEST GP BASED MIMICRY EXPLOITS

	Stide	pH	pHsm (mask known)	pHsm (mask unknown)	Markov Model
traceroute	16.67	11.71	0.00	27.60	0.10
Restore	0.40	0.10	0.20	0.31	0.10
Samba	0.50	0.10	0.00	29.23	0.10
ftp	57.14	0.10	0.00	35.55	0.10

The anomaly rates of the preamble (i.e. break-in) components are detailed in Table XVII. We note that none of the attacks deployed completely undetected. In previous mimicry attack research, the attack is considered to completely evade detection if the exploit component raises no alarms. However, even the exploit raises no alarms (e.g. FtpD attack against pH with schema mask, when the schema mask is known by the attacker, in Table XVI), preamble component, in other words, the actions taken by the attacker to take control of the application raises alarms (Table XVII) hence producing non-zero anomaly rates (Table XIV).

TABLE XVII.
ANOMALY RATE (%) OF THE PREAMBLE COMPONENTS ONLY

	Stide	pH	pHsm	Markov Model
traceroute	22.22	36.49	96.83	8.54
Restore	77.82	81.01	99.39	35.08
Samba	3.57	9.97	8.68	6.78
ftp	19.04	21.94	40.48	6.11

In addition to comparing anomaly rates of the best mimicry attacks, we also analyze the spread of anomaly rates generated for traceroute (Figure 1), restore (Figure 2), samba (Figure 3), and FtpD (Figure 4) using box plots. The box plot defines the 3rd quartile, median and 1st quartile. Whiskers extend from each end of the box to the adjacent values in the data; that 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.

According to the box plots, compared with the other detectors, the Markov Model produces the least anomaly rate under each application. This implies that the attacker has a better chance of hiding the attack within normal behavior. Given that the Markov Model only monitors the current and the past system call (as opposed to past 6, 9 and 20 for Stide, pH and pH with schema mask, respectively), this is not surprising. Furthermore, although the best mimicry attacks against pH with schema mask produced anomaly rates comparable to pH, the box plot analysis shows that mimicry attacks against pH with schema mask produce higher anomaly rates. Therefore, schema mask seems to be effective in making it more difficult to generate mimicry attacks as long as the schema mask remains unknown to the attacker.

IV. CONCLUSION

In this work, we employ an evolutionary mimicry attack framework against four anomaly detectors that monitor various vulnerable UNIX applications. The mimicry attacks are evolved not manually but automatically, in this case using a linear genetic programming based approach. It is assumed that the attacker can obtain a copy of the detector and run it locally. Realistically, the detector will not be open source and / or the attacker will not know the exact configuration parameters of the detector at the target site. Therefore, the task of the attacker is to craft a mimicry attack that will evade the detector on the victim host by minimizing the anomaly rate obtained from the local copy of the detector. Such a perspective denotes a black box model of the detector, whereas previous research has relied on a white box assumption [5-8]. In our framework, emphasis is placed on the: (i) Identification of an appropriate set of system calls from which exploits are built, in this case informed by the most frequently executed instructions from the vulnerable application. (ii) Identification of appropriate goals such as minimization of detector anomaly rate and delay, whilst matching key steps in establishing the exploit goals. (iii) Support for obfuscation, where in this case this is a direct side effect of the stochastic search operators inherent in EC. (iv) Measuring anomaly rate over the entire attack, i.e. preamble + exploit.

Relative to the first instance of this framework [11], we extend the approach to include Pareto multi-objective optimization, thus identifying a spread of solutions per run. Secondly, rather than concentrate on a single detector-application, the work reported here benchmarks against multiple detector-applications under the same configuration of evolutionary mimicry attacker. Thus, we are able to explicitly establish that the model is scalable and generic.

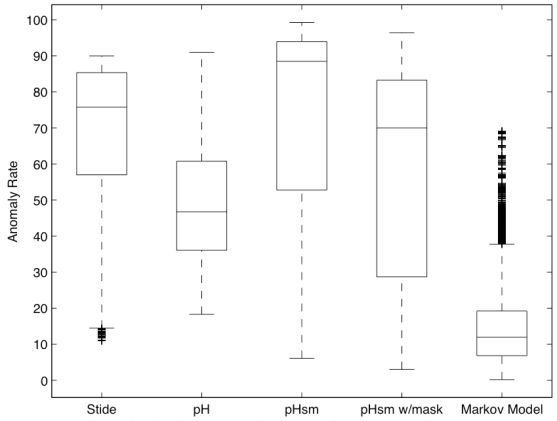

Figure 1. Box plot of mimicry attack anomaly rates against Traceroute

Results indicate that the framework is successful in reducing the anomaly rate of the attacks only utilizing the anomaly rate that the detector produces without using privileged detector information. For example, the traceroute mimicry attack against pH with schema mask produces 2.70% anomaly rate whereas the original attack produces over 95% anomaly rate. Needless to say, the degree of success depends on various characteristics of the attack such as the preamble length, preamble anomaly rate and the

exploit length. In other words, finding a zero anomaly rate exploit does not necessarily imply the attack can completely evade detection. Our results indicate that for an FtpD exploit that raises no alarms, the corresponding attack still produced 10.62% anomaly rate.

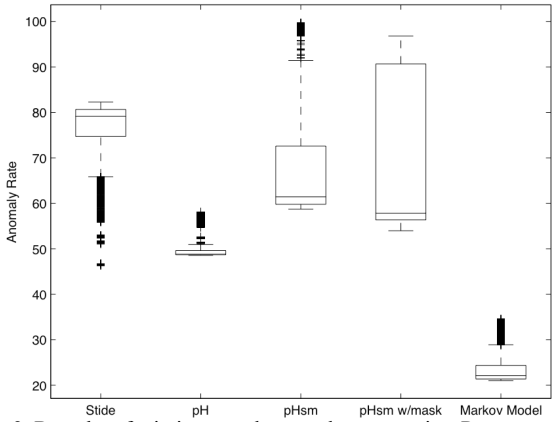

Figure 2. Box plot of mimicry attack anomaly rates against Restore

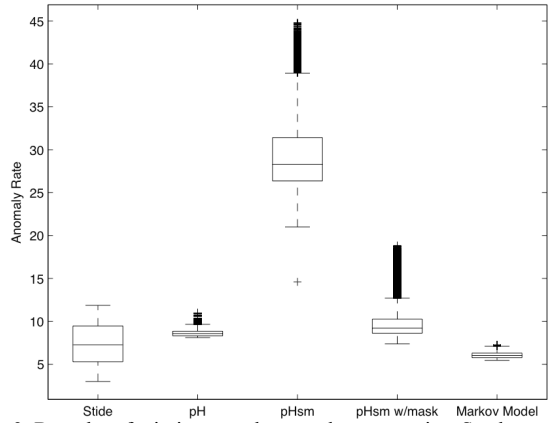

Figure 3. Box plot of mimicry attack anomaly rates against Samba

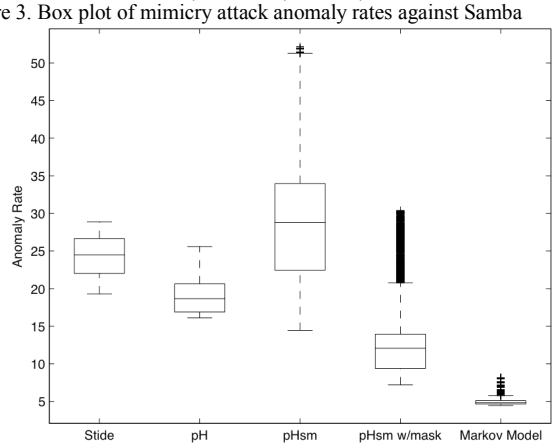

Figure 4. Box plot of mimicry attack anomaly rates against Ftp

REFERENCES

[1] Ptacek T. H., Newsham T. N., Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection, Technical Report, Secure Networks, January 1998.

[2] Mutz D., Vigna G., Kemmerer R., An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems, 19th Annual Computer Security Applications Conference, pp. 374-383, 2003.

[3] Vigna, G., Robertson, W., Balzarotti D., Testing Network Based Intrusion Detection Signatures Using Mutant Exploits, ACM Conference on Computer Security, pp. 21-30, 2004.

[4] Kayacik H. G., Zincir-Heywood A. N., Heywood M. I., Evolving Successful Stack Overflow Attacks for Vulnerability Testing, 21st Annual Computer Security Applications Conference, pp. 225-234, 2005.

[5] Wagner D., Soto P., Mimicry attacks on host based intrusion detection systems, ACM Conference on Computer and Communications Security, pp. 255-264, 2002.

[6] Tan, K. M. C., Killourhy, K. S., Maxion, R. A., Undermining an Anomaly-based Intrusion Detection System using Common Exploits, RAID'2002, LNCS 2516, pp 54-73, 2002.

[7] Kruegel C., Kirda E., Mutz D., Robertson W., Vigna G., Automating mimicry attacks using static binary analysis, USENIX Security Symposium, pp. 717-738, 2005.

[8] Tan, K. M. C., McHugh J., Killourhy K. S., Hiding Intrusions: From the Abnormal to the Normal and Beyond, Symposium on Information Hiding, pp. 1-17, 2002.

[9] Tan K. M. C., Maxion R. A., Determining the Operational Limits of an Anomaly-Based Intrusion Detector, IEEE Journal on Selected Areas in Communications, 21(1), pp. 96-110, 2003.

[10] H. G. Kayacik, M. I. Heywood, A. N. Zincir-Heywood, On Evolving Buffer Overflow Attacks using Genetic Programming. Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO'06). SIG EVO, ACM Press. pp. 1667-1673, 2006.

[11] Kayacik H. G., Heywood M. I., Zincir-Heywood A. N., "Evolving Buffer Overflow Attacks with Detector Feedback", Proceedings of the 4th European EvoCOMNET Workshop, LNCS 4448:11-20, 2007.

[12] Banzhaf W., Nordin P., Keller R. E., Francone F. D., Genetic Programming: An Introduction. Morgan Kaufmann, 1998.

[13] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A sense of self for unix processes. In Proceedings of the IEEE Symposium on Security and Privacy, pp. 120--128, 1996,

[14] Somayaji, A. B.: Operating System Stability and Security Through Process Homeostasis. Doctoral Thesis. UMI Order Number: AAI3058952., The University of New Mexico. 2002.

[15] H. Inoue and A. Somayaji, "Lookahead Pairs and Full Sequences: A Tale of Two Anomaly Detection Methods." 2nd Annual Symposium on Information Assurance (academic track of the 10th NYS Cyber Security Conference), June 2007, Albany, NY

[16] Yeung D.- Y., Ding Y., Host-based Intrusion Detection using Dynamic and Static Behavioral Models. Pattern Recognition. 36. pp 229-243, 2003.

[17] Debin Gao, Michael K. Reiter and Dawn Song "Behavioral Distance Measurement Using Hidden Markov Models" In Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID 2006), Hamburg, Germany, September 2006.

[18] R. Sekar, M. Bendre, D. Dhurjati & P. Bollineni, A Fast Automaton-based Method for Detecting Anomalous Program Behavior, IEEE Symposium on Security and Privacy pp. 144-155, 2001.

[19] Baum L.E., Petrie T., Soules G., Weiss N., A maximization technique occurring in the statistical analysis of probabilistic functions of Markov Chains. Annuals of Mathematical Statistics. 41(1) pp 164-171, 1970.

[20] Kayacik H.G., Zincir-Heywood A.N., Generating representative traffic for intrusion detection system benchmarking. Proceedings of the 3rd Annual Communication Networks and Services Research Conference. pp 112-117, 2005.

[21] Stide, http://www.cs.unm.edu/~immsec/data-sets.htm, Last accessed May 2006.

[22] SecurityFocus Vulnerability archives: LBNL Traceroute Heap Corruption Vulnerability, http://www.securityfocus.com/bid/1739

[23] SecurityFocus Vulnerability archives: RedHat Linux restore Insecure Environment Variables Vulnerability, http://www.securityfocus.com/bid/1914/

[24] SecurityFocus Vulnerability archives: Samba 'call_trans2open' Remote Buffer Overflow Vulnerability, http://www.securityfocus.com/bid/7294

[25] SecurityFocus Vulnerability archives: Wu-Ftpd Remote Format String Stack Overwrite Vulnerability, http://www.securityfocus.com/bid/1387/

[26] Kumar, R. and Rockett, P. 2002. Improved sampling of the Pareto-front in multiobjective genetic optimizations by steady-state evolution. Evolutionary Compututation 10(3), (2002), 283-314.