Testing Detector Parameterization using Evolutionary Exploit Generation

H.G. Kayacık, A.N. Zincir-Heywood, M.I. Heywood, S. Burschka

Faculty of Computer Science, Dalhousie University.
6050 University Avenue, Halifax NS, B3H 1W5, Canada
Software & Security Technologies, Swisscom Innovations, Switzerland
{kayacik, zincir, mheywood}@cs.dal.ca
Stefan.Burschka@swisscom.com

Abstract. The testing of anomaly detectors is considered from the perspective of a Multi-objective Evolutionary Exploit Generator (EEG). Such a framework provides users of anomaly detection systems two capabilities. Firstly, no knowledge of protected data structures need be assumed. Secondly, the evolved exploits are then able to demonstrate weaknesses in the ensuing detector parameterization. In this work we focus on the parameterization of the second generation anomaly detector 'pH' and demonstrate how use of an EEG may identify weak parameterization of the detector.

1 Introduction

Buffer overflow attacks represent one of the most effective and widespread examples of a host based attack. Detector techniques for recognizing such attacks generally fall into two basic forms: misuse or anomaly. The misuse paradigm focuses on recognizing the attack, whereas the anomaly paradigm concentrates on modeling 'normal' behaviour and flags everything else as 'anomalous.' Given the ease with which current examples of misuse detection can be evaded [1], we concentrate on the case of anomaly detection. Previous work on evolving attacks has indicated that it is possible to evolve exploits against the original Stide anomaly detector [2] when feedback from the detector is limited to the (publicly available) alarm rate [3]. Moreover, such works only consider crafting an exploit, without the preamble which set up the vulnerability such that the exploit can be launched.

Unlike the original Stide anomaly detector, the second generation process Homeostasis (pH) model of anomaly detection both detects and instigates counter measures against suspicious processes [4]. Detection is performed relative to sequences of system calls made by an application and processes with above normal anomaly rates receive a corresponding exponentially weighted delay. The goal of this work is to perform a vulnerability assessment of the pH anomaly detector under a vulnerable UNIX application using a black box approach. Specifically, the only detector information used to 'guide' the search for better exploits is the alarm rate, where this is public information available to a user. Conversely,

in order to establish better criteria for EEG we make use of a Pareto multi-objective fitness function. We demonstrate that such a framework presents users with a wider characterization of the scope of potential vulnerabilities for a given anomaly detector, as well as quantifying the impact of the detector parameterization. Moreover, we explicitly include the impact of the preamble within the evaluation, so that exploits are identified within context.

2 Methodology

We are interested in evolving buffer overflow attacks against pH where the detector has the capacity to delay the process associated with suspicious system call sequences. Moreover, given that an attack is composed from preamble and exploit it would also be useful to reward the evolutionary exploit generator for reducing anomaly rates associated with the attack as a whole as opposed to the exploit alone. Given the multiple criteria associated with this task it is reasonable to assume a Pareto framework for designing our exploit generator. Before we do so, however, we first need to establish the basic components of the evolutionary model; in particular the representation and model of deployment. To this end we assume the framework established by Kayacık *et al.* [3] and summarize this in Section 2.1. The design of the multi-criteria fitness function is considered relative to the basic framework in Section 2.2.

2.1 Basic Evolutionary Exploit Generator

A basic framework for EEG under a black box detector assumption was considered to comprise of two basic steps [3]: identification of a representative set of system calls, and design of the fitness function, as follows:

System Calls: It is necessary to *a priori* specify a set of instructions (system calls) from which candidate exploits are built. The approach adopted by Kayacık *et al.* was to first run the vulnerable application (*not* the anomaly detector) under normal operation and then review the most frequently executed system calls during execution. That is to say, a diagnostic tool – in this case Strace[1] – is deployed to establish the most frequent 20 system calls of the application (accounts for upwards of 90 percent of the system call utilization). It is these system calls that are used as the instruction set of the evolutionary exploit generator.

Fitness Function: Two basic criteria were previously identified for guiding evolution towards an effective attack: criteria for a valid attack, and quality of the attack. Criteria for a valid attack assumed that the basic goal of a buffer overflow exploit were to: open a file, write to the file, and close the file. The file in question was the password file, where the arguments to the open-write-close sequence established the specifics of the target file. Naturally, the sequence of these three operations is significant. Thus, individuals were

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

rewarded for both the instructions executed and the order of execution. In the case of exploit quality, use was made of the anomaly rate returned by the detector. Thus if the detector considered the instruction sequence anomalous the ensuing penalty would be greater.

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

2.2 Evolutionary Multi-criteria Exploit Generator

In assuming an EMO basis for evolving exploits we naturally make use of the method of Pareto ranking to combine multiple objectives into a single 'scalar' fitness function [5]. Specifically, a process of pairwise comparison is used to establish to what degree each individual is dominated by other individuals in the population. Thus, one individual (A) is said to dominate another (B) iff A is as good as B on all objectives and better than B on at least one. However, a wide range of algorithms have been proposed to support such a rank based fitness function [6]. In this work we assume the PCGA framework of Kumar and Rockett [7]. Such a scheme avoids the need to support explicit measures of similarity, where such schemes are generally employed to encourage diversity and/ or measure similarity in the population. Instead diversity is established care of a steady-state style of tournament. Specifically, after the initial population is ranked, parents are chosen using proportional selection and the ensuing children ranked. The rank of the children is compared to that of the worst ranked individuals in the population. If the children have a better rank, then they over write the (worst ranked) population members. Use of such a EMO model is necessary in this work as it is not possible to establish distance information relative to the original problem domain. That is to say, popular algorithms such as NSGA-II and SPEA all make use of Euclidean based distance functions when building the diversity metric (see for example [6]). In this work the representation takes the form of system call sequences as opposed to points in multi-dimensional space, thus precluding the application of such distance metrics.

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

1. Attack Success: As established in Section 2.1 the basic functionality of the attack is described in terms of an 'open-write-close' sequence of system calls, with reward established for each of the three instructions and the relevant order[2]. A behavioural success function gives a maximum of 5 points the correct 'open-write-close' behaviour, Figure 1.

[2] Relevant arguments are also considered, although pH does not consider them.

2. Anomaly Rate: Again as established in Section 2.1, the detector anomaly rate represents the principle metric for qualifying the likely intent of a system call sequence; a would be attacker naturally wishes to minimize the anomaly rate of the detector. Again, no inside knowledge is necessary as detectors provide alarm rates as part of their normal mode of operation. Moreover, as indicated in the introduction, the attacker also needs to minimize the anomaly rate of the exploit. We evolve exploit with preamble appended to facilitate identification of the most appropriate content.
3. Attack Length: Attack length appears as an objective to encourage evolution to perform a wider search for solutions.

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

Fig. 1. Fitness objective quantifying the exploit functionality.

3 Results

3.1 pH detector Configuration

pH is a second generation anomaly detector in which specific instances of the detector are associated with each application. During training, pH employs a sliding window to establish the normal behaviour of the application in terms of a matrix with dimensions: (1) current system call; (2) previous system call; (3) location of the previous system call in the sliding window. During testing, the same sliding window is employed on the candidate exploits. If a given sliding window sequence produced a look ahead pair that is not in the normal database, a mismatch is recorded. Given a pre-specified window size and system call trace length, the anomaly rate for the trace is calculated by dividing the number of mismatches by the total number of look ahead pairs. Moreover, the delay property used by pH to penalize suspicious processes is calculated as an exponential function of locality frame count; where locality frame (LF) count aims to identify clusters of anomalies. In this work, the default pH training parameters were employed, as defined by Somayaji [4].

3.2 Traceroute Application Configuration

In the following experiments, traceroute is tested. Traceroute is a vulnerable application frequently used in the attack automation literature [8]. Traceroute is

a network diagnosis tool, which is used to determine the routing path between a source and destination by sending a set of control packets to the destination with increasing time-to-live values. 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 [9]. In order to establish traceroute behavior under normal conditions we developed five use cases: (1) targeting a remote server, (2) targeting a local server, (3) targeting a non-existent server, (4) targeting the localhost, (5) help screen.

3.3 Performance Evaluation

Each run of the EEG may provide up to 500 exploits. In the following analysis, we concentrate on the non-dominated performance characteristics as captured post training. Specifically, this will take the form of three properties: anomaly, sequence length, and delay. By way of a baseline we also include the performance of the original attack. Table 1 details the characteristics of the non-dominated attacks under the Traceroute application. The last row details the characteristics of the original attack. It is apparent that the anomaly rate of the overall attack can be significantly reduced. In terms of attack and exploit delays, the attack that achieves the least delay achieves this by deploying a short exploit, hence reducing the delay from 6.39E+06 to 0.55 seconds. Naturally, this is only achieved while doubling the anomaly rate of the attack (from 16.29% to 30.91%).

Table 1. Non-dominated characteristics of Attacks against Traceroute application.

Optimization Criteria	Attack Anomaly (%)	Exploit Anomaly (%)	Length	Attack Delay (sec)	Exploit Delay (sec)
Anomaly rate	18.29	11.71	118	6.39 e+6	1.11
Delay	30.91	100	9	0.55	0.02
Length	65.11	66.77	1,000	3.83 e+28	3.75 e+28
Original	66.27	73.91	261	4.39 e+35	4.39 e+35

4 Conclusion

A second generation anomaly detection system – pH – is assessed for effectiveness under a black box information criterion. To do so, a multi-objective model of EEG is employed. Such an approach enables us to determine the cross-section of (non-dominated) behaviours evolved post training. It is clear that the preamble plays a very significant role in establishing the effectiveness of the overall attack. The parameterization of pH is a tradeoff between collecting statistics over longer window sizes and establishing a good model for normal behaviour

versus detection of very short attacks. Under Traceroute the low system call count of the preamble enables the configuration of attacks that on the one hand register a high anomaly rate, yet escape the delay property of pH. Specifically, the total system call count for the evolved attack is 91, thus smaller than the locality frame window size of 128. Moreover, the exploit alone required 9 system calls (100 percent anomaly); thus, the use of a Pareto multi-objective model of evolution enabled us to discover that such an attack could still be very effective. Thus, the would-be attacker would have compromised the system before an administrator could have reacted to the alarm. From the perspective of the overall vulnerability analysis the EEG enables us to optimize the detector parameterization.

Acknowledgements

The authors gratefully acknowledge the support of SwissCom Innovations, MITACS, CFI and NSERC grant programs.

References

1. Kayacik, H.G., Heywood, M., Zincir-Heywood, N.: On evolving buffer overflow attacks using genetic programming. In: Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO), SIGEVO, ACM (2006) 1667–1674
2. Forrest, S., Hofmeyr, S.A., Somayaji, A.B., Longstaff, T.A.: A sense of self for unix processes. In: Proceedings of the IEEE Symposium on Security and Privacy. (1996) 120–128
3. Kayacik, H.G., Heywood, M., Zincir-Heywood, N.: Evolving buffer overflow attacks with detector feedback. In: Proceedings of the EvoWorkshops (EvoCOMNET). Volume 4448 of LNCS., Springer (2007) 11–20
4. Somayaji, A.B.: Operating system stability and security through process homeostasis. PhD thesis, The University of New Mexico (2002)
5. Goldberg, D.E.: Genetic Algorithms in Search Optimization and Machine Learning. Addison Wesley (1989)
6. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley and Sons (2001)
7. Kumar, R., Rockett, P.: Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution. Evolutionary Computation **10**(3) (2002) 283–314
8. Tan, K., Killourhy, K., Maxion, R.: Undermining an anomaly-based Intrusion Detection System using common exploits. In: Recent Advances in Intrusion Detection Systems (RAID). Volume 2516 of LNCS. (2002) 54–73
9. SecurityFocus: Lbnl traceroute heap corruption vulnerability. http://www.securityfocus.com/bid/1739 (Last accessed June 2008)