Applied Soft Computing 11 (2011) 4366–4383

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Can a good offense be a good defense? Vulnerability testing of anomaly detectors through an artificial arms race

Hilmi Güneş Kayacık[a,*], A. Nur Zincir-Heywood[b], Malcolm I. Heywood[b]

[a] Carleton University, School of Computer Science, 1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada
[b] Dalhousie University, Faculty of Computer Science, 6050 University Avenue, Halifax, NS B3H 1W5, Canada

ARTICLE INFO

Article history:
Received 28 April 2010
Received in revised form 3 August 2010
Accepted 28 September 2010
Available online 14 October 2010

Keywords:
Computer security
Intrusion detection
Evasion attacks
Genetic Programming
Arms race

ABSTRACT

Intrusion detection systems, which aim to protect our IT infrastructure are not infallible. Attackers take advantage of detector vulnerabilities and weaknesses to evade detection, hence hindering the effectiveness of the detectors. To do so, attackers generate evasion attacks which can eliminate or minimize the detection while successfully achieving the attacker's goals. This work proposes an artificial arms race between an automated 'white-hat' attacker and various anomaly detectors for the purpose of identifying detector weaknesses. The proposed arms race aims to automate the vulnerability testing of the anomaly detectors so that the security experts can be more proactive in eliminating detector vulnerabilities.

1. Introduction

An intrusion detection system (IDS) is a combination of software and hardware that collects and analyses data from networks and hosts to determine if there is an attack [1] and possibly react to it as in the case of intrusion prevention systems [2]. Different detection techniques can be employed to search for evidence of intrusions. To this end, two major categories exist for detection techniques: misuse and anomaly detection. Misuse detection systems use *a priori* knowledge on attacks to look for traces of attacks. In other words, they detect intrusions by employing a description of the misuse [3]. On the other hand, anomaly detectors adopt the opposite approach, which is, to know what is normal, and then find the deviations from normal behavior. These deviations are considered as anomalies or possible intrusions. Anomaly detection systems rely on knowledge of normal behavior to detect attacks.

Naturally, intrusion detection systems are by no means infallible. Software vulnerabilities and hardware faults can cause them to misclassify or malfunction. In addition to traditional software errors, detectors are also susceptible to detector-specific vulnerabilities such as misconfigurations, blind-spots and deficiencies in detection methodology. Sophisticated attackers try to deploy attacks without getting detected. To this end, they may make use of detector vulnerabilities to alter their actions thus evading detection, rendering the detector ineffective. Although the evasion methodologies vary, the main objective of the attacker is to alter the attack so that it does not trigger signatures or generate anomalous behavior, while carrying out the attacker's goals.

In this work, we propose an arms race between artificial 'white-hat' attackers and candidate detectors. By 'white-hat' we imply an automated process for vulnerability testing (attack generation) without access to private information regarding the detector architecture. Feedback from the detector is limited to public information that a legitimate user might expect to receive; such as alarms. Such a scenario implies that we could deploy the same generic white-hat approach to any number of different detector architectures. Relative to the previous approaches discussed in Section 2, this means that we make extensive use of heuristic search – in this case Genetic Programming – to conduct the search for vulnerabilities under the guise of general objectives of an attack. Conversely, previous researchers made strong assumptions regarding the operation of specific detector architectures and face limitations on the subset of detectors they may evaluate (see [4] for a performance comparison between the two general approaches). We note that this work provides an empirical evaluation of an automated technique for vulnerability testing. Developing a theoretical model for testing detectors is beyond the scope of this work.

The proposed Evolutionary Exploit Generator (EEG) makes two assumptions (Section 3). There is a common goal of an attack – in this case adding the user to the root login file – and the system calls of the target application may be profiled. Given that the target

* Corresponding author.
E-mail addresses: kayacik@ccsl.carleton.ca (H.G. Kayacık), zincir@cs.dal.ca (A.N. Zincir-Heywood), mheywood@cs.dal.ca (M.I. Heywood).

1568-4946/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2010.09.005

applications take the form of generic Linux applications, such information can be collected without accessing privileged information from the application itself. We use the set of most common application system calls to define the instruction set for GP. The fitness function is expressed as a multi-criteria objective rewarding alarm rate minimization, correct formulation of an exploit and – in the case of the more sophisticated detectors – delay minimization.

The basic EEG framework represents a generic framework for designing buffer overflow style attacks, where this represents a major class of intrusions [5]. However, in order to be effective, users need to be clear regarding what the concept of an attack represents, particularly with respect to the relative contributions of preamble and exploit to the construction of an attack (Section 3). With the nature of buffer overflow attacks established, we then introduce the vulnerable applications and the corresponding set of candidate detectors on which benchmarking will later be performed (Section 4).

The following benchmarking study emphasizes three themes (Section 5). Firstly it is important to evolve attacks with the preamble included. Earlier works have tended to ignore the contribution of this entirely and make the mistaken claim that detectors can be avoided with zero anomaly rate. Secondly, the delay mechanisms embedded in some detectors (i.e., the IDS is able to react to potential intrusions by delaying the running process) has the capacity to render all the original attack variants, and all but one EEG attack, ineffective; again as a consequence of the contribution of the preamble. Finally, we are able to characterize the strategies used by EEG to obfuscate the true intent of an attack, with different strategies clearly being adopted depending on the detector against which EEG is deployed.

Section 6 brings these findings together and in doing so recognizes that a major factor in the most successful attack is the corresponding succinctness of the preamble. That is to say, the finite size of the vulnerable buffers may enforce system call quotas in which the attack must be expressed. Consequently, if the preamble is short, then the corresponding relative contribution of the exploit component of an attack (the part the attacker has most control over) has much more impact on the overall alarm rate. Thus the relative count of instructions that do not conform to the normal application behavior profile is much lower. Naturally attacks with lengthy and anomalous preambles will be next to impossible to obfuscate.

2. Related 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 and Soto [6] 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. [7] aimed to undermine the anomaly based IDS Stide [8] 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 data characterizing normal behavior into the malicious system call sequence. Vigna et al. [9] 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. [10] developed a static analysis tool for Intel x86 binaries in order to automatically identify instructions that can be used to redirect control flow. They use symbolic execution to achieve this. Giffin et al. generated mimicry attacks against Stide by applying automatic model checking to prove that no reachable operating system configuration corresponds to the effect of an attack [11]. However, in their approach, the operating system model, application (program) model and system call specifications as well as the attack configuration are 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 artificial 'white-hat' attackers i.e., the Evolutionary Exploit Generator (EEG) framework. The arms race rewards the attacker as it builds successful attacks, which can defeat the target detector. In such an arms race, the detector responds to attacks by providing feedback from the detector in the form of anomaly rates or other detection information such as the nature of dynamic measures deployed against an attack (delays). Consequently, the attacker utilizes the detection feedback to build evasion attacks, which achieve the objectives of the attacker while minimizing the detection from the target detector. The main result of the arms race is a set of evasion attacks, which can evade the target detector. The resulting attacks provide the defenders with crucial information that can be utilized to eliminate the weaknesses of the target detector. Needless to say, the exploits produced are entirely a result of the Evolutionary Exploit Generator with no hand crafting of the exploits.

Second, the previous work [6,7] assumed that the attacker can take control of the vulnerable application silently i.e., no consideration was given to the contribution of the preamble (Section 3.1) to attack detection. By contrast, in this work, we acknowledge 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 is it possible to evade the more sophisticated detectors (Section 5). We are also able to demonstrate that as the target detector changes, the composition of the attack controlled by EEG undergoes a significant change, implying that different detectors do have rather different implicit weaknesses.

3. Evolving buffer overflow attacks

In this section, the EEG framework is introduced to evolve attacks for analyzing vulnerabilities of detectors along with a brief discussion of buffer overflow attacks. Furthermore, we discuss the relevant work employing similar arms race methodologies for other attack types, although this work focuses on buffer overflow attacks.

3.1. Generic design decisions of a buffer overflow attack

In a typical buffer overflow exploit, the first step is to corrupt the data types and local variables, which gives the attacker control of the application. For example, in case of the original ftpd attack against wu-ftpd server [12], the attacker achieves this by logging onto the ftpd server anonymously and issuing malformed commands such as *CWD* ~ {. The actions taken by the attackers before they gain full control of the application are called the preamble. During the preamble phase, the application is still operational and the attacker does not have full control yet, hence the attacker may not be able to prevent the vulnerable application from generating anomalous behavior.