Finding Local Resource Exhaustion Vulnerabilities*

João Antunes Nuno Neves Paulo Veríssimo
LaSIGE, Fac. de Ciências da Univ. de Lisboa
{jantunes,nuno,pjv}@di.fc.ul.pt

Abstract

Computer systems connected to the Internet are highly susceptible to hackers that can compromise the service availability through denial of service attacks, causing damage to customers and service providers. Our work focuses on using the attack injection methodology with some advanced monitoring capabilities to detect and identify local resource exhaustion vulnerabilities. It goes even further by providing valuable insight about the effort necessary for an attacker to exploit them and for the target system to be able to sustain the attacks.

1. Introduction

Every service provided through the Internet can suffer denial of service (DoS) attacks aiming at its disruption. Attackers can perpetrate a DoS either by overwhelming the target system or its network connection with an excessive load, or by making use of some specific and well-crafted packets that exploit some known DoS vulnerability. Our work focuses on the identification and detection of a particular kind of DoS vulnerabilities that allows an attacker to deplete some local resource on the target system. In this context we define a *local resource exhaustion* vulnerability as a specific type of fault that by causing the consumption or allocation of some unnecessary resource, or the failure to release it when no longer needed, makes the resource susceptible of being eventually depleted. Consuming unnecessary resources happens when some component allocates more resources than what is required to perform the task. For instance, a badly designed algorithm executes too many instructions or reserves huge chunks of memory that will not be used. If the resource was indeed necessary but it was not made available after its use, such as by neglecting to close a temporary file descriptor, to free some allocated chunk of memory, or to delete a no longer necessary file log, the component is leaking the resource.

The attack injection methodology presented here differs greatly from other fault injection methodologies. Classic fault injectors [1, 2, 5] inject simple software or hardware faults in a target system, usually for hardware validation or for the verification of fault handling mechanisms. However, the low-level mimicked faults (e.g., pin-level faults or single bit-flips in memory) are too simple to be applied in the detection of security vulnerabilities, where the universe of the possible faults escalates to an intractable problem. Fuzzers [3, 6] deal with this intractability by injecting random samples as input to the target system. Though these tools have evolved into more intelligent vulnerability detectors, they are only capable of detecting crash-related faults. AJECT [4], our first attack injection tool, is able to *fuzz* protocol specifications with some known malicious attack patterns (e.g., very long strings, strange characters, known usernames and filenames) to detect security vulnerabilities. A vulnerability is detected upon the observation of a anomalous server behavior, such as the reception of SIGSEGV signal. It must be noted that AJECT is not required to actually exploit the vulnerability but only to cause some detectable disturbance in the target system. In other words, for instance, to detect a buffer overflow the attack does not need to actually fill the overflown buffer with some root shell command. A simple out-of-bounds of arbitrary content is sufficient for the detection of the illegal memory access. AJECT, however, lacks a finer monitoring system capable of detecting more subtle kinds of vulnerabilities. Local resource exhaustion vulnerabilities can be particularly difficult to locate because normally they are only perceived when some resource has been depleted (e.g., disk or CPU). The paper presents a new attack injection tool, PREDATOR, that aims at detecting DoS and local resource exhaustion vulnerabilities.

2. The PREDATOR Tool

PREDATOR (PREDict ATtacks On Resources) is an attack injection tool with some very interesting characteristics: it provides a thorough resource and process monitoring, capable of automatically detecting small resource usage variations in servers, such as in CPU work, wall time, number of processes, memory, disk, or open files; it performs on-line analysis of the monitoring data to refine its attack injection and detection mechanism to the most promising attacks; and it is

*This work was partially supported by the EC through project IST-2004-27513 (CRUTIAL) and NoE IST-4-026764-NOE (RESIST), and by the FCT through projects POSC/EIA/61643/2004 (AJECT) and the Large-Scale Informatic Systems Laboratory (LASIGE).

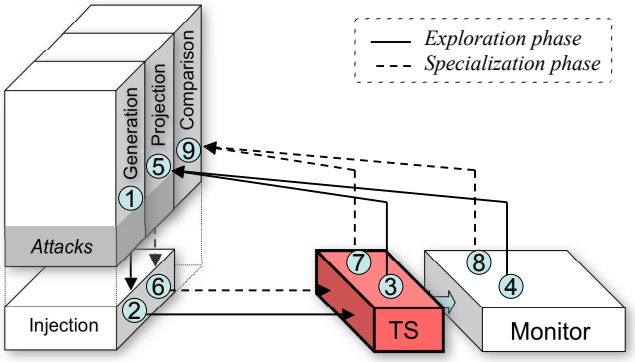

Figure 1. PREDATOR's architecture.

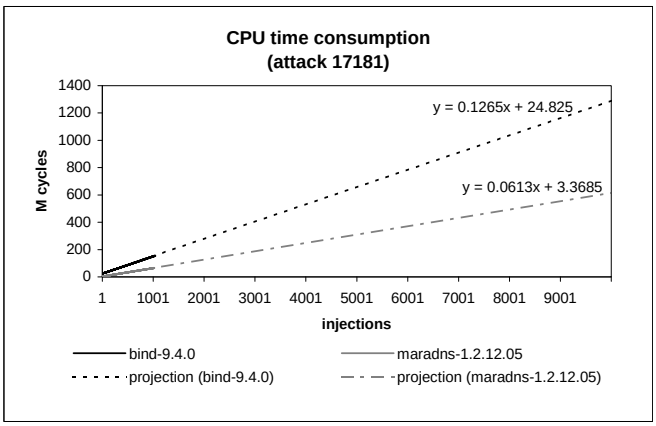

Figure 2. DNS servers CPU consumption.

able to provide a prediction on the resource usage which can be utilized to compare the behavior of distinct target servers in a given system. This prediction is an estimate for the resource consumption for the long-term, giving the ability to pinpoint the most dangerous protocol interactions and predict when the resource utilization may reach a fatal threshold.

The general architecture of the tool is represented in Figure 1. PREDATOR can perform the injections of the attacks under the same conditions of the attackers, outside the target machine (e.g., from the Internet). However, the specificity of the OS monitoring facilities demands that the monitor component be placed in the operating environment of the target system (TS), and consequently in the same machine.

The injection and monitoring is attained in several steps, as depicted by the numbers in the figure. The attacks are generated from a specification of the server's communication protocol and according to an attack generation algorithm (step 1). Then, each attack is injected several times in order to obtain a relevant amount of monitoring data needed to generate a resource usage trend (step 2). The monitoring data for each attack injection is used to obtain a resource usage projection or trend (steps 3 and 4). The injector performs linear regression on the utilization of each resource in order to select the potentially more dangerous attacks (step 5). Therefore, a *resource usage profile* for each pair of attack/resource is modeled after the resulting linear function. This trend can be used to predict the resource usage for any number of client requests. The growth rate for a particular resource is given by the line's slope. The higher the slope, the worse the server's performance and resource usage efficiency is. Figure 2 shows the PREDATOR's prediction for the CPU consumption under the same attack (protocol request) of two widely used DNS servers. The prediction is a projection of the real resource usage data for a larger number attacks. The attacks with higher projection values are selected for a more exhaustive and specialized injection campaign (step 6). Finally, the resulting monitoring data from the selected attacks allows the different servers to be compared on an equal basis (steps 7, 8, and 9).

Naturally, the final prediction for these selected attacks get an even higher estimate accuracy. Figure 2 is actually the final prediction for 10,000 repeated attacks, based on the monitoring data obtained from 1024 injections. With the collected results one can, for instance, determine how vulnerable a given server is for each attack, or compare the expected behavior of distinct servers on the same platform (i.e., determine which server would collapse first under a specific attack).

In our preliminary experimental results with a few of the most used DNS servers, PREDATOR indicated the most relevant resource usage trends and predicted the potential resource depletion. Figure 2, for instance, shows that the BIND server performs worse than MaraDNS under the same attack, which means that the later is able to sustain a larger number of attacks than the first. Similarly, other local resources, such as wall time, memory, disk, children/threads and file descriptors, are also analyzed and their exhaustion estimated. This predictive analysis can be of invaluable help in preventing DoS by either providing useful information on the hardware limitations, or by identifying resource bottlenecks.

References

[1] J. Arlat, Y. Crouzet, and J.-C. Laprie. Fault injection for dependability validation of fault-tolerant computing systems. In *Proc. of the Int. Symp. on Fault-Tolerant Computing*, 1989.

[2] J. Carreira, H. Madeira, and J. G. Silva. Xception: Software fault injection and monitoring in processor functional units. In *Proc. of the Int. Working Conf. on Dependable Computing for Critical Applications*, 1995.

[3] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Comm. of the ACM*, 33(12), 1990.

[4] N. Neves, J. Antunes, M. Correia, P. Veríssimo, and R. Neves. Using attack injection to discover new vulnerabilities. In *Proc. of the Int. Conf. on Dependable Systems and Networks*, 2006.

[5] T. K. Tsai and R. K. Iyer. Measuring fault tolerance with the FTAPE fault injection tool. In *Int. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation*, volume 977 of *LNCS*. 1995.

[6] University of Oulu. PROTOS, 1999–2003. http://www.ee.oulu.fi/research/ouspg/protos/.