

UCLA Computer Science Department
CS239 (Lecture 3) – Winter 2003

active-SPAM

Extending SPAM to defend against “Busy DoS” attacks

Petros Efstathopoulos
pefstath@cs.ucla.edu

1. Introduction

Denial of service Attacks have become one of the most common ways to attack a system. Numerous recent incidents have proved that even carefully designed systems (e.g. large scale web servers) can be brought down by a DoS attack. This is mainly because today, there are no wide spread and well established mechanisms to detect DoS attacks and give the servers appropriate notice. Many ongoing research projects aim to deal with the DoS attack detection. The goal of this project was to develop a mechanism for detecting a certain category of DoS attacks.

One way to categorize DoS attacks is based on the kind of resources the attackers try to abuse. We can say that there are two kinds of attacks:

- ***Busy attacks***, which are targeted to renewable resources (like CPU cycles) and try to bring down the system by not allowing legitimate usage of these resources
- ***Claim-and-hold attacks***, which are targeted to non-renewable resources (like memory and disk space) and try to allocate and hold large portions of limited resources, thus bringing down the system due to resource (e.g. memory) starvation.

SPAM (*State Profiling and Analyzing Module*) addresses the problem of Busy attacks. The method we use to attack the problem is based on real-time in-kernel process

monitoring, used to gather data about a process’ behavior, which are later on inspected by a user space application that is responsible for analyzing data and detecting possible DoS attacks.

2. Basic SPAM concepts

SPAM is based on the assumption that server processes have a specific structure: they can be seen as a repeated pattern of CPU intensive, computational work surrounded by two or more system calls. This assumption is based on observation of most of the popular internet applications. Furthermore, most of the examples of server processes vulnerable to “Busy attacks” support this assumption. A server process can thus be treated as an repetition of the following pattern:

- Execute initial system call (e.g. read data)
- Execute other possibly needed system calls (or none)
- Process request (CPU intensive)
- Execute other possibly needed system calls (or none)
- Execute final system call (e.g. write/send results)

This model for server processes implies that the code vulnerable to “Busy attacks” is the CPU intensive part that is surrounded by system calls. During a busy attack, the attacker is trying to provide malicious input to the server, so as to consume almost all the CPU cycles by exploiting weaknesses of the request processing code.

Based on this model, SPAM tries to identify all system calls of a server process and use them to build a state machine that describes the process’ execution. Each system call that appears in the process code is a node (state) of this state machine. The arcs that connect the states (transitions from one state to another) have to different kinds of “weights”: time elapsed between the two occurrences of the states and frequency of occurrence of each transition. After this state machine has been built for a particular process, one can identify possible “Busy attacks” by monitoring the process’ execution and comparing the measured properties with the ones on the arcs of the state machine. The reference state machine is built under normal conditions of execution (no attacks take place). In other words, SPAM has a *training phase*.

The implementation of SPAM is a combination of a Linux kernel module and a user space daemon. The SPAM kernel module intercepts a subset of the system calls and replaces them with a custom version that has the exact same functionality as the original system call, but also records the system call as an “event” of the process’ execution. The recording of events is done by a lightweight tracing mechanism that goes through the stack and uses register’s values (mainly the values of EIP) to compute a checksum that uniquely positions and identifies the system call in the process’ execution tree.

The user space daemon is responsible for gathering the information from the kernel module in order to construct the state machine. Events are read from a special character device supported by the kernel module (`/dev/spam`). During the training phase the events are used to build the state machine and calculate the standard deviation for each arc value (time elapsed, frequency of occurrence). After the training phase is over, the state machine is stored in files so that it can be used later on in order to monitor the process’ execution and detect possible “Busy attacks”.

3. Extending SPAM – *active-SPAM*

The initial goal of SPAM was to detect “Busy” DoS attacks. After SPAM was successfully implemented there was an obvious question that needed an answer: what should the system do *after* a “Busy attack” has been detected? The purpose of the project was to improve SPAM and extend it so that it can react appropriately after an attack has been detected. This led to the next version of SPAM : *active-SPAM*.

Two alternative methods of defending against a “Busy attack” are introduced in this project: delaying system call execution and/or applying traffic shapers to malicious incoming traffic.

Delaying a system call execution assumes that *active-SPAM* can identify the process that’s being attacked. This is always the case though, since one of the things SPAM reports is which process is being attacked. Thus, once *active-SPAM* has detected that a particular process is under attack, it instructs the kernel module to delay the execution of all system calls issued by that particular process by a certain amount of seconds (jiffies actually). Since we don’t want to permanently penalize the process, this measurement is effective for a specific period of time (e.g. 10 minutes). This means that once the kernel module has been informed that a particular process is being attacked, it will delay the execution of all the system calls issued by the process until the penalty expires (e.g. until after 10 minutes). This ensures that penalized processes can still operate normally after the attack is over.

Traffic shaping is a standard way of defending against DoS attacks which is used mainly on routers. Applying a traffic shaper to all incoming packets with a particular source address can reduce the effects of a DoS attack, since less malicious requests reach the server. In order to apply a traffic shaper one must know the source IP address of the packets. This information has to become part of the information included in a SPAM “event”. In the implementation of *active-SPAM* the required information is gathered in the IP level of the network stack. The functions that handle IP events (incoming connections, incoming packets etc) were replaced by SPAM-aware versions that monitor incoming traffic and provide the same network functionality. By keeping a record of IP addresses that recently sent data to a specific server process, *active-SPAM* can shape

traffic originating from the “suspect” host. The traffic shaping approach is based on the Linux kernel’s QoS services.

4. Implementation - Details

The implementation was based on the SPAM structure, which needed to be heavily modified and enhanced in order to support the new defense mechanisms. SPAM was improved so as to be more precise as far as timing and time measurement are concerned. Most of the data structures had to be modified to support the new features and provide information to the defense mechanisms.

Implementing the system call delays was based on kernel timers. The *active-SPAM* kernel module associates a kernel timer with each penalized process. The timer expires after a certain (compile-time user defined) period of time, after which the process is no longer considered “under attack”. If the process’ system calls are delayed and process receives a new attack, then the kernel timer is reset. After the attack is over gradually all penalized processes will return to normal execution status.

One major change was the interception of IP related functions. All incoming connections and packets are now intercepted, so that *active-SPAM* can keep track of the source IP addresses. Knowing the IP addresses related to each request allows the system to defend against attacks using traffic shapers. The user space daemon that recognizes the attacks is responsible for instructing the kernel to apply the appropriate traffic shaper. After a certain (user defined) period of time the *active-SPAM* daemon removes the shaper, in order to ensure that the remote host is not penalized permanently. If the attack has not finished, the shaper is applied again.

One interesting issue with *active-SPAM*’s traffic shaping defense is implemented in the IP level. That makes it effective for both TCP and UDP attacks. This is particularly interesting since defending against UDP based attacks has proven to be particularly difficult.

A set of tools needed to be implemented in order to test *active-SPAM*’s functionality. The most important tools are *spam-ctrl* and *httpd*. The kernel module is monitored by *spam-ctrl*, an application that uses `ioctl` calls to interact with *active-SPAM* kernel mechanisms. A simple http server was implemented in order to test the system’s ability to detect “Busy DoS attacks”. This http server is vulnerable to a specific attack: it uses a very inefficient loop to parse URLs which makes it vulnerable to incoming requests with many “/” appended after the URL.

5. Comments - Future work

One of the most common issues with security mechanisms is the possibility of “false positives”. The state machine monitoring implemented by *active-SPAM* reports some false positives. Although a totally legitimate request might cause a few false positives the system is still able to recognize attacks since the number of “positives” reported after malicious requests is one or two order magnitude bigger. A typical number of “false positives” during our tests was 24 per group of legitimate requests. A group of malicious requests produced 120 to 1200 “positives”. This issue has introduced the need for a “tolerance” factor: when the daemon starts it prompts the user to enter the maximum number of “positives” that will be ignored. For most of the tests the tolerance factor was set to 30.

An interesting extension of *active-SPAM* would involve cooperation with routers. Traffic shaping is much more effective when done by routers. Since *active-SPAM* is able to detect “Busy DoS attacks”, it could notify the corresponding router(s) and provide them with the IP addresses of the attackers and the attacked ports. The combination of *active-SPAM*’s defense mechanisms with router-based defense would lead to a much more effective solution to most of the DoS attacks detected by *active-SPAM*.

This project has shown that it is possible to build a system that is able to detect and defend against anomalies in the execution of a process that are due to a certain type of DoS attacks. The effectiveness of the defense mechanisms used by *active-SPAM* is still under study. Building a more stable (and bug-free) version of the system is the main goal for future work.

6. References

- [1] Petros Efstathopoulos and Vassilis Pappas “*SPAM: State Profiling and Analyzing Module. An approach to DoS attack detection*” UCLA, Fall 2002.
- [2] Xiaohu Qie, Ruoming Pang and Larry Peterson “*Defensive Programming: Using an Annotation Toolkit to Build Dos-Resistant Software*” OSDI, 2002.
- [3] Daniel P. Bovet and Marco Cesati “*Understanding the Linux Kernel*” (2nd Edition).
- [4] Anil Somayaji and Stephanie Forrest “Automated response using system call delays” 9th USENIX Security Symposium, 2000.
- [5] Stephanie Forrest, Steven Hofmeyr, Anil Somayaji and Tomas Longstaff “*A sense self for Unix processes*” IEEE Symposium on Security and Privacy, 1996.