
Abstract

A new low-complexity approach to intrusion detection called "bottleneck verification" was developed which can find novel attacks with low false alarm rates. Bottleneck verification is a general approach to intrusion detection designed specifically for systems where there are only a few legal "bottleneck" methods to transition to a higher privilege level and where it is relatively easy to determine when a user is at a higher level. The key concept is to detect 1) When legal bottleneck methods are used and 2) When a user is at a high privilege level. This approach detects an attack whenever a user performs operations at a high privilege level without using legal bottleneck methods to transition to that level. It can theoretically detect any novel attack which illegally transitions a user to a high privilege level without prior knowledge of the attack mechanism.

Bottleneck verification was first applied to UNIX workstations to detect when an attacker illegally achieves root-level privilege and starts a root shell. Initial experiments used sniffing data, ASCII transcripts created by extracting text strings from the sniffed data, and more than 80,000 actual telnet sessions. Use of the single legal command that can transition to root (the UNIX "su" command) was detected using a telnet transcript parser written in Perl. The same parser was also used to determine when a user obtains root-level privilege primarily by examining the shell prompt and the types of commands that are successfully issued. Bottleneck verification was compared to a baseline intrusion detection system which uses keyword counts and an expert system to determine the likelihood that a telnet session contains an attack. Receiver operating characteristic (ROC) curves were generated for the baseline system and bottleneck verification using known instances of attacks in the real data. At a detection rate of 80%, bottleneck verification reduced the false alarm rate by almost two orders of magnitude compared to the baseline system. It also indicated exact locations in scripts where attacks occur and is efficient enough to consider real-time implementation. Bottleneck verification found many different types of attacks that lead to illegal root transitions including buffer overflows, suid root backdoor shells, and bugs in specific application programs without any prior knowledge of these attacks.

Following this initial work, the bottleneck verification approach was extended to work with Solaris Basic Security Module (BSM) audit log data. The BSM bottleneck verification algorithm issues a warning whenever a root shell is started from a process and the owner of the process did not either log in directly as root (and thus start with root permissions) or execute a valid "su" command anywhere in the process' ancestry. It detects a shell being launched by monitoring all fork() and exec() system calls. A table of all currently running processes is kept which includes the process id and privilege level. Entries are removed from the table by monitoring the exit() and kill() system calls to detect when processes are terminated. This system was implemented in real time using a perl program that examines BSM audit data after it has been converted to ASCII using the "praudit" application. Real-time implementation is possible because only a few
audit events are monitored and the detection logic is relatively simple. Testing used buffer
overflow and suid root shell script attacks. BSM bottleneck verification found all attacks and has
not produced any false alarms when monitoring normal computer activities.

Current work involves performing more extensive testing of BSM bottleneck verification,
developing a real-time version of the sniffer-based algorithm, and extending the BSM algorithm
to detect other hacker actions besides root shell creation including modifying the passwd and
other important system configuration files.