1. What is the paper about? What is/are the vulnerability? What cause the vulnerability?

The authors of the paper propose an application layer blocker to prevent code-injection buffer overflow attacks targeting web services. Buffer overflows occur when too much data is copied into a fixed-sized buffer. Since the buffer cannot hold all the data, the overflow data is written into adjacent memory locations without determining if those memory locations are free before doing so. Buffer overflow attacks exploit the vulnerability, but can be prevented/detected in six different ways or classes.

1. What is/are the contributions of the paper? How was the vulnerability or insecurity discovered?

Buffer overflows were originally discovered in hacking circles.

The first type of prevention technique is to clean up or eliminate bugs in source code. Since most often buffer overflows exploits work because there are logical errors in the software, eliminating these points of weakness eliminate the ability for the exploits to be successful. The second is through compiler extensions.

The third class of buffer overflow attacks use OS modifications.

The fourth type of attack is hardware modification.

Defense-side obfuscation is the fifth type of attack.

The sixth and final type of buffer overflow attacks are capturing code running symptoms of buffer overflow attacks.

1. The detailed techniques to solve the problem.

The SigFree software detects the presence of code in client request messages and removes instructions it believes to be useless. This means the software does not need access to the source code, as it analyzes messages with embedded machine code. Since SigFree software is signature free, it does not rely upon already discovered sequences to detect and block attacks. This also eliminates the need for updates based on newly discovered attack signatures. SigFree software runs as a proxy between the servers/clients it is protecting will allow it to provide transparent protection.

1. What are the strength/weaknesses of the paper?

The first weakness I could see what they software relies on the assumption normal requests do not contain executable machine code. The problem with that assumption is if a person is trying to exploit a system, they will usually use techniques outside of the norm, which could include putting executable machine code in requests.

The second weakness is that the software uses a threshold to compares the number of useful instructions when determining if the sequence contains code. It is possible the attack would not trigger their threshold, but still could contain enough instructions to carry out the attack.

1. What can you do better?