1. What is the paper about? What is/are the vulnerability? What causes 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 a fixed-sized buffer has too much data copied into it. 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 and can give the attacker control over the host/victim.

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

Hacking circles are the mostly likely origin for buffer overflow attacks, but the attacks in general are behind many of the modern day cyber attacks. While there are many existing product to defend against buffer overflows, the authors of the paper rebuke that they are limited in meeting four requirements many customers want. Their stance is that customers like maintenance to be simple, existing application software/hardware/server OS is in not impacted, a solution that is easy to understand resisting obfuscation, and the ability to deploy the solution Internet wide.

Existing defenses sometimes require changes to compiler extensions, modifications to the OS, and hardware modifications all require changes making the solution not transparent to the client and/or server. Other defenses that capture code running symptoms of an attack have a high runtime overhead or need special not commercially available software. Even other defenses require source code to defend against coding errors/bugs, but many times it is not possible to distribute the source code or source code is not available for other reasons. Worms are another defense and work by blocking attack packets; however, they require signatures adding greatly to the maintenance and rely on the attack to have an already known signature.

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, which allows it to provide transparent protection, meaning no updates to the OS, hardware, or software on either the client or the server.

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

The first weakness I could see what they software relies on the assumption normal requests between the client/server 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. While they insist their thresholds blocked all the attack packets they generated, it is possible the attack would not trigger their threshold, but still could contain enough instructions to carry out the attack.

One strength of the paper is that they have a lot of research and data to back up their claims all outlined throughout the paper. They also not only looked at how well it was able to defend against the attacks, but also how use of their software impacted performance of the system. In this case, they could increase their transparency by decreasing their system impact.

Another strength is the outline their limitations. While this may seem counterintuitive because it gives attackers a way to get around their defenses, it also gives those wishing to use their product a good general idea of where else their system may be vulnerable.

1. What can you do better?

Improvements to the SigFree software could be to identify if the source code detected is malicious or not before rejecting it. While this would open up the possibility for obfuscation by the attacker, it would allow for use of the product in client/server situation where code is included in the messages. In addition, I would include support for branch functions as they give the attacker a way to mask their program flow, which can allow malicious code to go undetected.