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

The authors of the paper propose an automatic detection and defense against Internet worm attacks on buffer overrun, format string vulnerabilities, and other overwrite attacks. These vulnerabilities target important information such as return addresses, function pointers, or format strings on the target machine and can be used to do many malicious things from crash a program to execute harmful code.

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

Existing defenses can require source code or special complication of the binaries it uses. These constraints make it impossible to use when source code is not available and inconvenient to use when a wide range of applications are in use (recompiling each application is time consuming). Fine-grained detectors can be costly and are not always deployable on every vulnerable host. Since fast filters are key to securing a host/program before the vulnerability reaches it, many existing systems use content-based signatures to pattern-match when determining if the packet payload contains a specific attack. Unfortunately, these signatures are all part of a database that is manually generated. This lends to a slow reaction time, which can be devastating in the case of a worm set to infect thousands of machines in just a few hours.

TaintCheck does not require source code or the special compilation of binaries it uses. It operates on the existing binary program, making it simple to use even on proprietary programs or programs in which no source code is available. This makes TaintCheck easy to deploy across a wide range of applications. TaintCheck also boasts that it detects format string attack and overwrite attacks targeting return address pointers, function pointers, or offsets for function pointers. There were no false positives in the experiments performed with the default policy. They claim any false positives that do appear reduced or eliminated through TaintCheck configuration.

1. The detailed techniques to solve the problem.

The TaintCheck software identifies which parts of the payload could be useful in an attack signature and performs automatic semantic analysis. At the processor-instruction level, TaintCheck monitors every byte of every attack payload is used by the vulnerable program. This analysis can be used to generate an attack signature directly instead of relying upon signatures generated from content pattern extraction. Since the semantic analysis provides information about not only the possible vulnerability, but also how the vulnerability may be exploited, less payloads would be necessary to generate an accurate signature then with content pattern extraction. This could help minimize the damage caused by a newly introduced work by catching the exploit/vulnerability much earlier and allowing defenses to be put in place to contain it.

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.