Review “Taint”

Name: Wenhui Zhang
Email: wuz49@ist.psu.edu

Paper Name :
Newsome, J., & Song, D. X. (2005, February). Dynamic Taint Analysis for Automatic Detection, Analysis, and SignatureGeneration of Exploits on Commodity Software. In *NDSS* (Vol. 5, pp. 3-4).

Contribution:
Newsome and Song propose a method for auto-detection and analysis of exploits, the dynamic taint analysis approach automatically generates signatures for attack filtering on commodity softwares. They also propose a tool, TaintCheck, which demonstrates the dynamic taint analysis approach.
This approach could (1) Detect new attacks. It could generate attack signatures and propagate the signature to other systems for new attacks. (2) TaintCheck as a sandbox. TaintCheck lets the exploit run, but prevents any real output from the machine. It log data coming out of the system and uses it to generate a more accurate signature. (3) Classification of vulnerabilities. TaintCheck detects details of exploit, it helps with classifying an exploit based on this information. (4) Signature verification. A new signature could be given to TaintCheck and perform detection for new vulnerabilities. Depending on how effective TaintCheck is at detecting the exploit with the signature, the signature is verified for its effectiveness.

Motivation:
Software vulnerabilities are exploiting the softwares and have huge negative impacts. It is easy to perform buffer overflow attacks and overwrite attacks , thus worms like CodeRed and Slammer are able to exploit hundreds of thousands of computers in just minutes. With this fast propagation speed, manual response strategies can’t react fast enough to stop the exploits. Thus we need an automatic detection and defense mechanism for exploit tracking and detection.

Methodology:
Dynamic taint analysis approach targets attacks that overwrite data in memory, such as overwrite attacks. TaintCheck runs on Valgrind as an extension, it marks input or data that comes from untrusted sources or data that is arithmetically derived from an untrusted source as tainted. It uses shadow memory pointing to taint structures. Dynamic taint analysis then keeps track of the data throughout

the life of the program and detects if the data is used in an exploiting manner. When tainted data is used in a malicious way, TaintCheck analyzes the exploit and creates a defense. Once the attack is detected, signatures or filters are generated from the tainted data and its markers, and further attacks will be defended by these signature filters.

Results:

TaintCheck works with already compiled and proprietary softwares,thus the testing consisted of various server and client programs, such as ATPhttpd, bftpd, cfingerd, gcc, ls, bzip2, make, vim, emacs, and bash.

The evaluation of TaintCheck is executed in two aspects: detection of attacks, and performance of programs. (1) detection of attacks, they used the 3 server programs along with 3 synthesized programs. (2) performance, 2 worst case scenarios and 1 average scenario are tested, each one being run at native speed, then in Nullgrind (an extension of valgrind that does nothing), Memcheck (the default extension for valgrind), and TaintCheck.

It successfully detected most overwrite attacks, with no known false positives. New signature creation approach is also performed, which conducts semantic analysis based signature generation.

Discussion and Take away:

(1) The performance is low on CPU bound workloads, for example, on CPU-Bound workload bzip2, there is 37.2 times longer on TaintCheck. Because TaintCheck runs on valgrind as an extension, which emulates the code by rewriting and retranslating the code into x86 instructions. This process makes a lot of overhead. A faster emulator DynamoRio might help with the case, however performance is still an issue. Thus this approach is good for attack detection not prevention.

(2) Fine grained tainted memory tracking of inputs and data is slow, and could only be performed in run-time, and it causes overheads. Fine grained context sensitive static analysis could perform tainted variable tracking at static time. By generating signatures in static time, run-time overheads might be saved.

(3) Over the years, people have been inventing static CFI on hardwares, however there is no data flow analysis on hardwares. Tainted memory tracking might be implemented at hardware level, to lower the run-time cost.