Review “StackGuard”

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

Paper Name：

Cowan, C., Pu, C., Maier, D., Walpole, J., Bakke, P., Beattie, S., ... & Hinton, H. (1998, January). Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks. In *USENIX Security Symposium* (Vol. 98, pp. 63-78).

Contribution:

StackGuard is a compiler technique which lower possibility of buffer overflow attack by inserting canary words between return addresses and local variables in stack.

Motivation:

C language is not type safe, which does not check array boundaries. Von Neumann architectures is used in our PC at that time, which store code and data in the same memory, where strings grow up and stacks grow down. x86 gives same access privileges to page, and does not distinguish “readable” and “executable” memories. Thus programs are vulnerable to buffer overflow attacks: (1)legacy code runs with SUID root with software errors; (2)new programs are written using no-boundary-typed C languages.

Related works:

Other mitigation methods for buffer overflow attack include: (1) mark stack as non-executable, separate page with readable and executable tags; (2) conduct static source code analysis, such as do stack integrity check on FreeBSD; (3) conduct runtime array bounds check; (4) conduct “object code insertion” to instrument all memory accessed, and perform integrity check for each memory access; (5) using type safe languages.

Methodology:

StackGuard embeds canary words in stack frames and verify their integrity prior to function return. The algorithm works as following: (1)add canary word in stack between local variables/buffer and return addresses; (2) check If the canary word is damaged, if it is, which means stack is corrupted, then jump to step 4, if not, jump to step 3; (3) normal execution; (4) instead of jumping to attacker code, abort the program and log intrusion attempt records.

Results:

Canary version adds only hundreds of lines of code. It consumes 6% more CPU time, which is proved to be an efficient solution against buffer overflow attacks.

Take away:

(1) If we are not aiming at performance, maybe we should adopt type safe languages like Java.

(2) Some unsafe C lib functions includes:
- strcpy (char *dest, const char *src);
- strcat (char *dest, const char *src)
- gets (char *s)
- scanf (const char *format, …)
- printf (conts char *format, …)

Thus use type safer versions like strncpy(), strncat().