A method to mitigate the Code-Reuse Attacks

Advisor – Prof. R C Hansdah

Niranjan Singh

Sr. No. 11170

Buffer Overflow

- Applications use call Stack.
- Structure of call Stack shown below

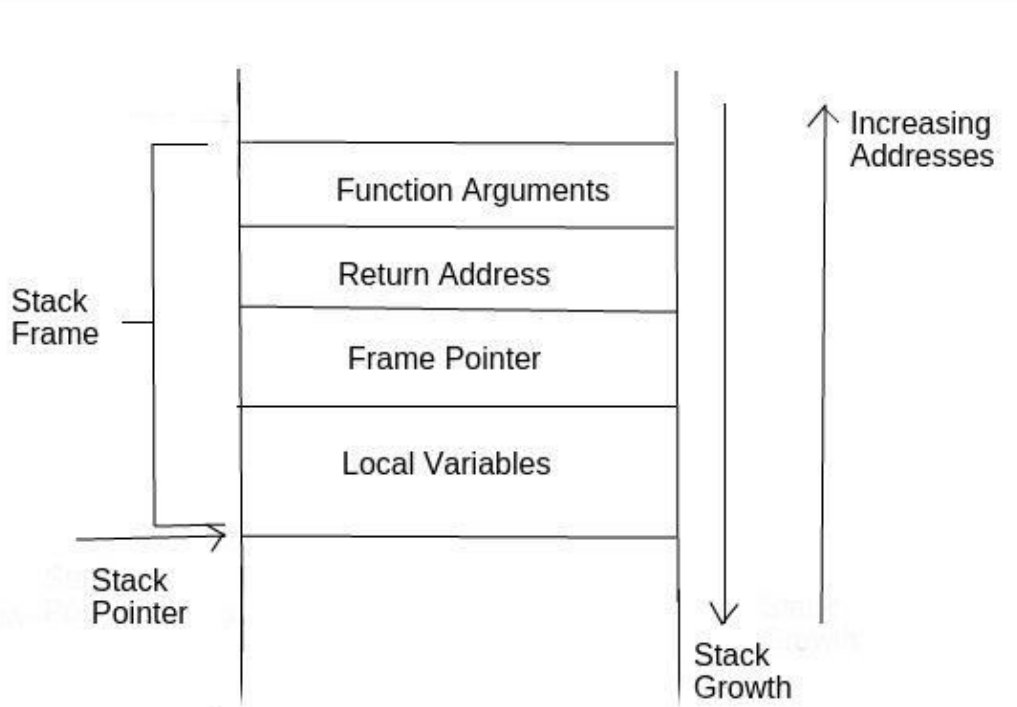

Figure 1: Structure of call stack

- Consider a small C language subroutine with one local variable buffer[12].
- strcpy() used to fill it.
  - Does not check bounds and overwrites other locations.

- Shellcode placed at location 0x80484cd.
- Input be "A"*12 +"A"*4 +“\xcd\x84\x04\x08”
- First 12 fills buffer[12]
- Next 4 ovewrites frame pointer
- Next 4 overwrites return address
- Now function would return to injected code at 0x80484cd.

Stack after injection looks like-

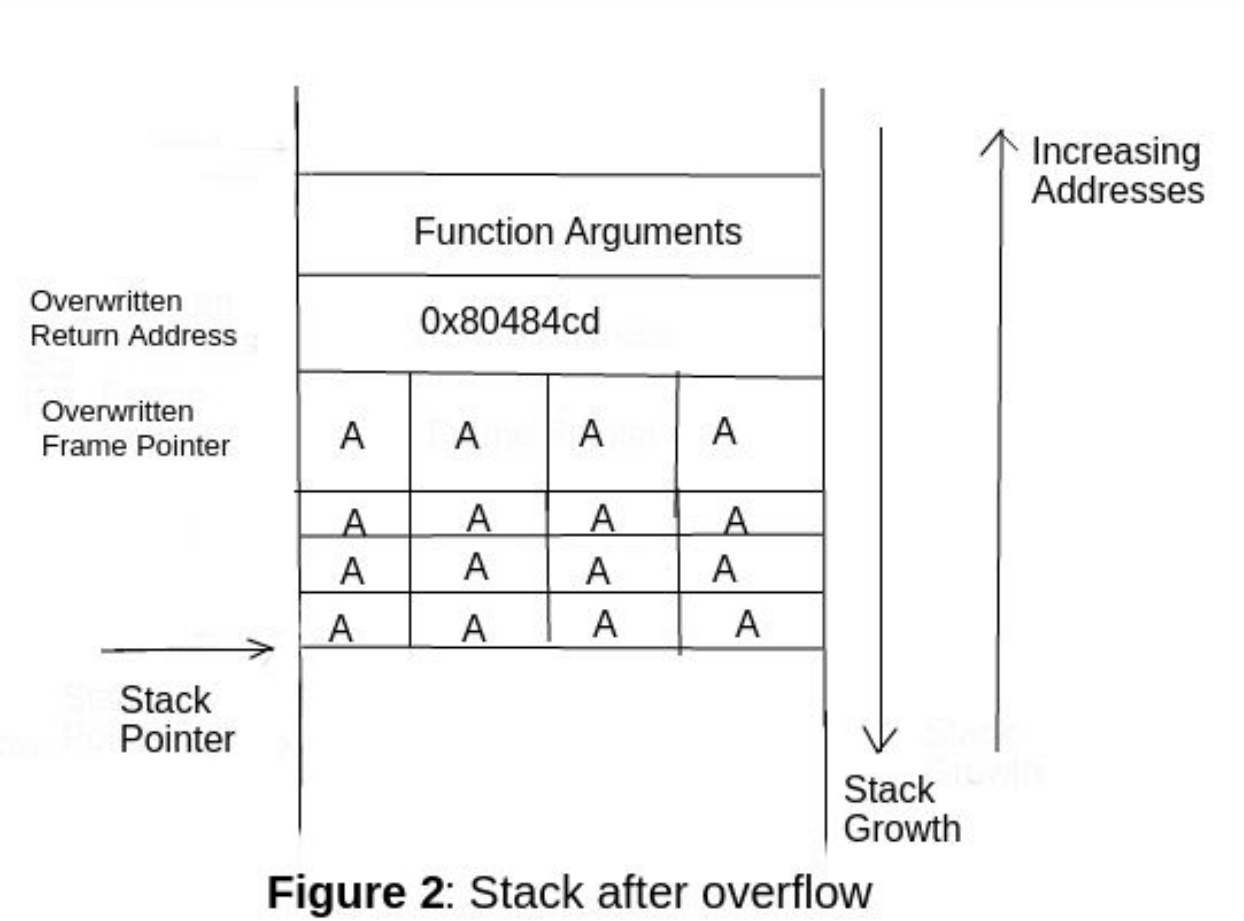

Figure 2: Stack after overflow

- prevented by Data execution prevention(DEP) technique.
- marks stack and heap segments as non-executable.
- used in almost every modern OS.

- Stack canary technique proposed, but shown to be bypassed.

Code-Reuse Attacks

- Does not inject new code
- Uses code present in applications address space.
- Shown to be Turing-complete.
- Overwrite code pointers to direct control flow to attacker payload.
- Payload consists of series of return address pointing to gadgets.
- Gadgets - atomic operations like ADD, LOAD, etc. ending at a "ret" instruction.
  - E.g. mov %edx, %eax; ret
- Mostly include-
  - Return-to-libc -> directs to code in libc
  - Return oriented programming -> uses gadgets ending with “ret”.
  - Jump oriented programming -> uses gadgets ending with “jmp”.

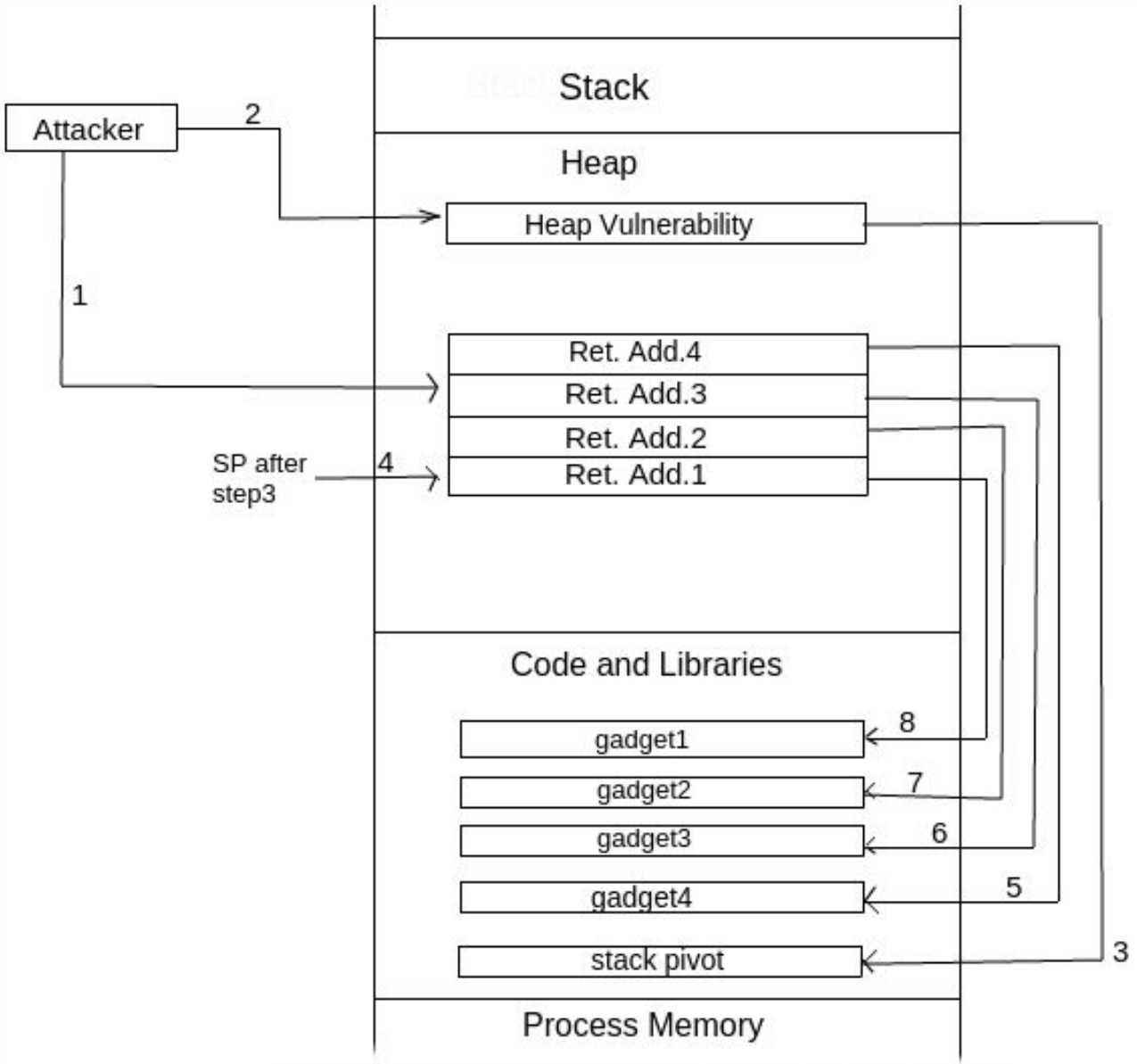

Figure 3: Basic flow of code reuse attack

Basic Mitigation Techniques

- Control-flow Integrity
- Address Space Layout Randomization

Control-flow Integrity

- creates a control flow graph of the application before execution.
- Each node represents a basic block
- runtime behavior of application is monitored to ensure that a valid path is followed by the control flow in CFG.
- Uses shadow stack, store return address and compares them with original stack when returns.
- Overhead of 21%.
- Coarse-grained solutions-
  - return address should point to an instruction directly after a call instruction.
  - monitor the number of instruction executed between consecutive branches.
  - Shown to be bypassed.

Address Space Layout Randomization

- randomize the base address of all the process areas like text, data, bss, libraries, etc.
- prevented the return-to-libc attacks.
- Used by almost all modern OS.
- Bypassed by brute-force on 32-bit systems.
- vulnerable to memory discloser attacks i.e., a single pointer leak may lead to the failure of whole ASLR.
- ASLR on 64-bit with fine-grained randomization proposed
  - Randomization done even at basic block level.
  - Bypassed on Linux using offset2lib vulnerability.
- Methods like ASLR-Guard proposed to prevent ASLR.

Advanced Offensive and Defensive Techniques

- Just-In-Time Code Reuse
  - Uses the single leaked code pointer to get address layout of application recursively at runtime.
- HAFIX: Hardware-Assisted Flow Integrity Extension
  - hardware based Backward-edge CFI approach.
  - a return is allowed to only target a call preceded instruction in a function that is currently active.

Proposed Method

- Idea is to prevent the discovery of stack pivot gadget.
- stack pivot gadget
  - modifies stack pointer to point it to attacker payload.
  - E.g. xchg %esp, %eax; ret
- After loading application, search for stack pivot gadgets.
- Store in table and encrypt them.
- Replace their location in code by a pointer to the table.
- Do decryption only for valid functions.

Conclusions and Future Work

- Code-reuse attacks are one of the most prominent security exploits present.
- No single approach is a silver bullet in defending these.

- In future we intend to implement our approach against code-reuse attacks.
- Will focus on security of encryption key.
- Consideration on performance of decryption.

References

- Aleph One. Smashing the stack for fun and profit. Phrack magazine, 7(49):14-16, 1996.
- PaX Team. Pax address space layout randomization (aslr), 2003.
- Kevin Z Snow et al. Just-in-time code reuse: On the eectiveness of ne-grained address space layout randomization. In Security and Privacy (SP), 2013 IEEE Symposium on, pages 574-588. IEEE, 2013.
- M L Davi et al. Haxfix: Hardware-assisted flow integrity extension. 2015.
- Ahmad-Reza S et al. Securing legacy software against real world code-reuse exploits: Utopia, alchemy, or possible future? In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, pages 55-61. ACM, 2015.

Thank You….