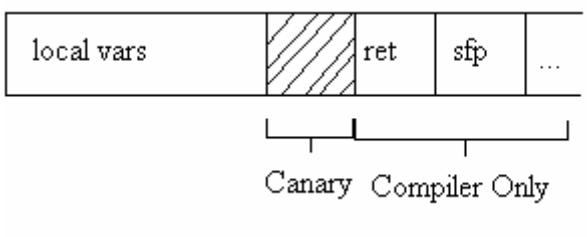
Runtime Defenses

Scribe Notes 9-9-08

By Mark Winterrowd

1. Stackguard – Make gcc modification

Stack frame layout:



Risk: Buffer in local vars overflows, alters compiler-only data

Solution: extra word, "canary", push canary on stack, exit will check canary to make sure is same. If changed, has overflowed

Ways to Choose Canary:

- Random: hard for attacker to know
- Have a zero byte: Will act as end of string at string overflow; won't stop memcpy

Weaknesses:

- Only protects stack
- Overwriting ptr allows attacker to bypass canary to change data
- Does not protect against buffer underruns
- Does not protect against format strings; can both bypass and discover canary
- Does not protect against attacks on local vars

2. Non-Executable Stack

- Also designed to protect against stack smashing
- Prevents code on the stack from being run

Weaknesses:

- No protection against arc injection
- No heap protection
- Does not protect ret address, can still jump to malicious code in heap

3. W^X (Write XOR Execute)

- Implemented by default in Vista
- Central Idea: No page in memory should be both writable and executable
- Impedes modification of code and running of data

Weaknesses:

- Does not protect against arc injection
- Can still map to arbitrary preexisting code through func pointer
- Can still overwrite local vars

4. Separate Stacks (to defend against smashing)

- One stack for Compiler and Fixed Length Data
- One stack for variable length data
- Still doesn't defend against arc injection, func pointers, heap overflows
- Turns stack smashing bugs into heap overflows

5. Electric Fence (Modification to Malloc and Free)

- When malloc-ing buffer on heap, put a guard page at the end of the buffer
- Any attempt to access guard page traps. Idea is to stop buffer overruns in guard

Weaknesses:

- Uses up virtual address space
- Can still modify past the guard page if you do not access memory sequentially
- No defense against format string bugs, buffer underruns, function pointer
- Could put guard before page to protect against buffer underruns
- Performance overhead is not good

6. Valgrind Memcheck

- Like a software simulator for CPU, memcheck checks for bad memory addresses
- Reserve part of address space for shadow data structures to track data
- For every byte in mem, there exists a bit in special region, says whether or not is okay to access region
- Change malloc to update bits to say memory is valid, free to say not valid
- Any portion of memory not intended for use by program can be marked

Weaknesses:

- slow (x20 slowdown)
- Architecture specific
- Doesn't prevent against arc injection
- Valgrind can't do much to protect stack, as data structure cannot tell difference between different data on stack

Some speed concerns slightly less relevant because is intended as debug tool, not protection modification.

7. Hypothetical Approach: Valgrind + Separate Stacks

- Any attacks against this?

8. ASLR (Address Space Layout Randomization)

- Rather than start stack @ fixed location, pick random spot. Do same for DLLs, code, etc.
- Attacker cannot predict location as easily

Weaknesses:

- Attacker can still use trampolines to jump to a location on stack. (Modify system to prevent use of trampolines)
- Attacker can randomly try spaces in memory
  - Success depends on entropy in choosing offsets
  - Current schemes try not to disrupt page alignment, and so uses addresses at r * 2^16. Lower bytes not randomized
  - If attacker tries 2^16-1 times, will cause 2^16-1 crashes which could be easy to detect, but not always. Sometimes errors do not appear. Example: Windows program which caught all exceptions, ignored exceptions caused by exploits
  - In Vista, ASLR uses 2^8 locations for most things, but 2^5 for stacks; low entropy means fewer guesses for attacker
- ASLR breaks some apps, Vista offers option to opt out. Two most popular opt outs are Firefox 2 and IE 7, apps that are likely to be attacked
  - Some programs also opt out of W^X (Java, Flash)

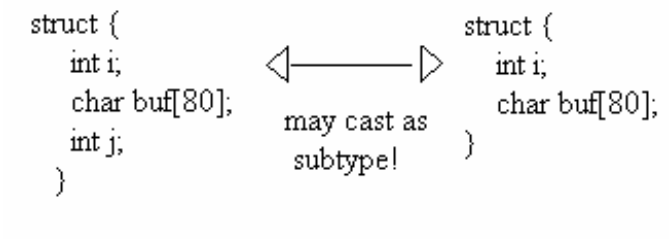
- Can strengthen ASLR by running two instances on the same input at the same time on different machines with different mappings. If under attack, one machine's path will diverge from the other, which can be detected.

- ASLR could potentially be more effective on 64 bit systems
- Can worry about mem mapping table; should rewrite addresses at load time

9. CRED

Weaknesses:

- Casts can open new holes
- only does protection for string bufs
- does it deal w/ stack bufs?



- Does it track bufs within a struct? Probably not.
- Uninstrumented libraries are not detected.
    - Major challenge in many tools
    - Perhaps changing malloc and free can help dealing with libraries.

How does it compare to Valgrind?

-Cred works faster

-Cred has to be compiled with

Valgrind:	**Cred**:
Validity bits	Per Object Checking
Will not check bounds of Obj	Checks bounds of Obj

Lessons for today:

- Dealing with instrumented and uninstrumented code is a headache
- Tools keep some shadow data structure, extend semantics to perform usual operations + updating shadow data structure.

Sidenote about arc injection:

- Recent research shows attacks more serious than anticipated
- Originally used entirely for return into libc
- But, if we jump to ret. Address and modify stack, may be expecting argument stored on stack
- Process runs based on corrupted stack

- Chaining these attacks can result in very flexible attacks
- Removing tempting lib functions doesn’t stop this, can search address space for “hidden treasures” that “do something interesting”
- Gathering hidden treasures can make a Turing complete op set
- Can create exploit compiler for this purpose!