a

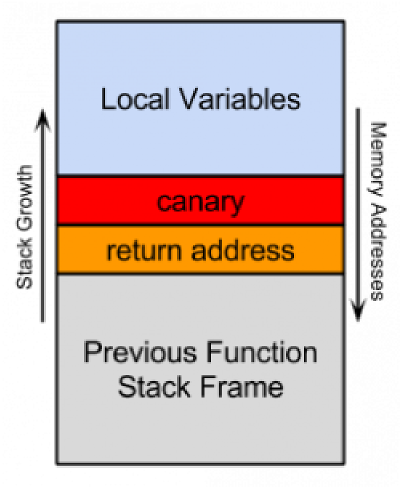
(𝔖$𝔖)

Session 11
Stack Canaries & Format Strings

Security Summer School
28th of July 2014
ACS/Ixia/Hexcellents

Contents

Stack Canary - Overview



Stack Canary - Types

- random
- terminator
- random XOR

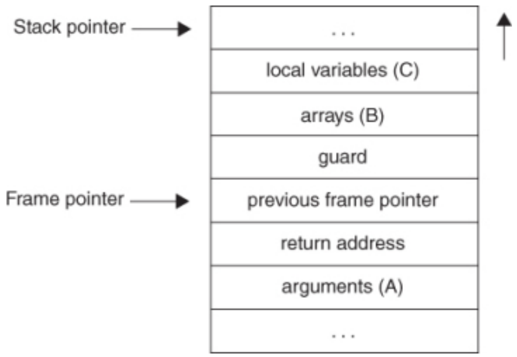
Stack Canary - Implementations

- StackGuard
- StackShield
- ProPolice SSP

Stack Canary - GCC

- GCC uses SSP (ProPolice)
- rearranges variables based on type
- `--fstack-protector`
- 3 levels of protections (#number of functions)

Stack Canary - GCC



Stack Canary - Attack Vectors

- target
    - parameter function pointers
    - return address
    - old base pointer
    - plain function pointer
- buffers on stack or on heap/.bss
- indirect attacks (reach target via other pointer)
- information leak

Format String Exploits

```c
void print_something(char* user_input)
{
    printf(user_input);
}
```

vs.

```c
void print_something(char* user_input)
{
    printf("%s", user_input);
}
```

Functions

- family of functions: printf, fprintf, sprintf, etc.
- also: setproctitle, syslog, err*, etc.
- indirect attacks (reach target via other pointer)
- information leak

Functionality

- convert simple C datatypes to a string representation
- specify representation format
- process the resulting string (output to stderr, stdout, syslog)

Mechanism

- the format string controls the behaviour of the function
- specifies the type of parameters that should be printed
- parameters are saved on the stack (pushed)
- saved either directly (by value), or indirectly (by reference)
- caller does stack correction after format function finishes

Stack

stack top
. . .
`<&a>`
`<a>`
`<i>`
`A`
. . .
stack bottom

where:

A	address of the format string
i	value of the variable i
a	value of the variable a
&a	address of the variable i

Format string

- `printf ("The magic number is:  %d", 1911);`
- some parameters: %d, %u, %x, %s, %n

Attack Basics

- `%s`: display memory from address supplied on stack
- `%x`: display value in hex
- `%n`: write number of bytes wirtten to variable
- `printf ("%s%s%s%s%s%s%s%s%s%s%s%s");`
- `printf ("%08x.%08x.%08x.%08x.%08x");`

Write

```
int a;
printf ("%10u%n", 7350, &a);
/* a == 10 */
int a;
printf ("%150u%n", 7350, &a);
/* a == 150 */
```

Write multi-byte values

```
strcpy (canary, "AAAA");
printf ("%16u%n%16u%n%32u%n%64u%n",
        1, (int *) &foo[0], 1, (int *) &foo[1],
        1, (int *) &foo[2], 1, (int *) &foo[3]);
printf ("%02x%02x%02x%02x\n", foo[0], foo[1],
        foo[2], foo[3]);
printf ("canary: %02x%02x%02x%02x\n", canary[0],
        canary[1], canary[2], canary[3]);
```

Generalization

- user input affects execution
- SQL injection
- XSS injection

Resources

- https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/cowan/cowan_html/cowan.html
- http://courses.cs.washington.edu/courses/cse504/10sp/Slides/lecture3.pdf
- http://lwn.net/Articles/584225/
- http://crypto.stanford.edu/cs155/papers/formatstring-1.2.pdf