Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns

Jonathan Pincus and Brandon Baker

Slides by: Noah Watkins
Buffer Overflows

• General term for a variety of attacks
• Largest share of CERT advisories
  – Computer Emergency Response Team
• Morris worm
• Code Red worm
• SQL Slammer
• Xbox, PS2, and Wii
Terminology

• Payload
  – Goal: Deliver the payload
  – Combination of 3\textsuperscript{rd} party code or data
  – Loaded at the time of attack (or prior)

• Program flow
  – Goal: Alter regular program flow
Buffer Overflow Example

• Write beyond array/buffer boundaries
• Buffer is located near security critical variables
• Stack Smashing Example
  – Write executable code in buffer
  – Modify function return value with data in overflow
  – On function return, code in buffer is executed
  – Example code...
Stack Smashing Example

(a)
```c
void f1a(void * arg, size_t len) {
    char buff[100];
    memcpy(buff, arg, len); /* buffer overrun if len > 100 */
    /* ... */
    return;
}
```

(b)
```c
void f1b(void * arg, size_t len) {
    char * ptr = malloc(100);
    if (ptr == NULL) return;
    memcpy(ptr, arg, len); /* buffer overrun if len > 100 */
    /* ... */
    return;
}
```
Fixing The Problem

• Check bounds
  – No, not gonna happen
• Language with run-time systems
  – Good solution
• C/C++ are vulnerable
• Speed
• Legacy code
• Must educate
Buffer Overflow Awareness Week

• Incompetent programmers
  – Get rid of them?
• Instead, taught generalities
• Stack smashing is resembles general buffer overflow definition
• Many classes of buffer overflows...
Buffer Overflow Variety

- Stack Smashing
  - Traditional exploit
- Arc Injection
  - Return-into-libc
- Pointer Subterfuge
  - Function and data pointers
- Heap Smashing
  - The above, but in dynamically allocated memory
Arc Injection

• Execution of code already present
• Supply data that causes normal behavior to give desired results (based on bad input data).
  – Spawn a new process
• More subtle than stack smashing
• Possible to do things such as running arbitrary process
• Example ->
void system(char *arg) {
    check_validity(arg);
    R = arg;
    ...
    target:
    execl(R, ...)
}
Pointer Subterfuge

- Function pointer subterfuge
  - Similar to modifying return address
- Data pointer subterfuge
- Exception-handler hijacking
- Virtual pointer smashing
Function Pointer Subterfuge

- Function pointer points to attacker code
- Buffer overflows may change function pointers
- Useful when return address is protected
  - StackGuard
- Combined with heap oriented overflows
- More flexibility in general
- Example...
**Function Pointer Overflow Example**

1. **Normal overflow**

```c
void f2a(void * arg, size_t len) {
    char buff[100];
    void (*f)() = ...;
    memcpy(buff, arg, len); /* buffer overrun! */
    f();
    /* ... */
    return;
}
```

2. **Mayhem**

**Combine with arc injection:** have function pointer be the address of code in Libc (e.g. `system()`)

---

**Note:** The code snippet provided demonstrates a function pointer attack where `f2a` takes a function pointer and an argument, and then calls the function with the argument. This can lead to buffer overflows if not handled carefully, especially when using dangerous libraries like `system()`. Combining this with an arc injection technique can further exploit vulnerabilities in the system.
Data Pointer Modification

• Arbitrary memory writes
• Controlling an address to which a write occurs
• Building block in complex exploits
• Combine with function pointer clobbering:
  – If the function pointer is not on the stack
  – Use data pointer modification to write it instead
• Example...
Data Pointer Modification Example

(b)

```c
void f2b(void * arg, size_t len) {
    char buff[100];
    long val = ...;
    long *ptr = ...;
    extern void (*f)();

    memcpy(buff, arg, len); /* buffer overrun! */
    *ptr = val;
    f();
    /* ... */
    return;
}
```

Control with buffer overflow

Arbitrary write
Exception Handler Hijacking

- Exceptions execute linked list of handlers
- Can modify the pointer to the list of handlers
  - Standard stack smashing
- Can create a apparently valid set of handlers
  - And change the first element of the real set
  - Use arbitrary memory writes
- Unexpected exceptions may bypass other protection mechanisms
Virtual Pointer Smashing

- Virtual pointers (think C++) are implemented with a table of function pointers
- The table is stored in object header
- Modify table via buffer overrun

```c
void f4(void * arg, size_t len) {
    char *buff = new char[100];
    C *ptr = new C;

    memcpy(buff, arg, len); /* buffer overrun! */
    ptr->vf(); // call to a virtual function
    return;
}
```
Heap Smashing

• Exploit dynamic memory allocator
• Corrupt block headers
  – May affect freeing mechanism behavior
• Difficult
  – Memory addresses are dynamic, hard to predict
  – Multi-threaded apps are less/non-deterministic
• Payload stored well in advance of attack
The Future

- C/C++ will continue to be used
  - In legacy applications
  - In new applications and languages
- Finding new protection mechanisms will lead to clever attacks
How To Get Started

• Read lots of code
• Study existing exploits
  – Subscribe to advisory/other lists
    • CERT
    • VulnWatch
    • BlackHat
    • Phrack
    • 2600