COMS E6998-9:

Software Security and Exploitation

Lecture 6: Input Validation

Hugh Thompson, Ph.D.
hthompson@cs.columbia.edu

Columbia University

Preventing Buffer Overflow Exploits

A Caveat…

- Most defenses in operating systems, compilers (/GS), etc. do not (or cannot) remove buffer overflow vulnerabilities, instead they focus on two things:
  - Make it very difficult to execute arbitrary code
  - Make it difficult to alter the execution flow of the application
- Even the best runtime defenses do not prevent a buffer overflow being exploited to *crash* the application.

Some unsafe C lib functions

strcpy (char *dest, const char *src)

strcat (char *dest, const char *src)

gets (char *s)

scanf (const char *format, …)

printf (conts char *format, …)

Preventing buf overflow attacks

- Use "safer" cousins of vulnerable C functions
  - Strncpy, strncat, etc.
- Type safe languages such as Java, C#, etc.
- Mark stack as non-execute (NX, DEP)
- Compile with canaries
- Make image relocatable (eg. -fpie in gcc)
- Run time checking (StackGuard, etc.)
- Test vigorously (static analysis, fuzzing, etc.)

Using NX

- Advantages:
  - When properly implemented, used *code* can not be executed from data segments
- Problems:
  - Some apps need executable stack
  - Does not block more general overflow exploits:
    - Overwriting a function pointer
    - Overwriting a variable value
    - Cannot make all the data segment non-executable
    - Can place your own parameters on the stack and then call some other function

Canary Types

- Random canary
  - Generate random string at each execution.
  - Insert canary string into every stack frame.
  - Verify canary before returning from function.
  - Makes reliable exploitation difficult…unless you can overwrite a function pointer or exception handler.

- Terminator canary
  - These are typically a set of termination values: Null, CR, LF, and -1 (0xFF)
  - In its most basic form, the canary may be 0x00000000

Address Space Layout Randomization

- Several operating systems - Windows Vista, Linux (via the kernel), etc. – now support address space layout randomization.
- Randomizes system libraries, heap, and stack.
- If compiled appropriately, the application image may be randomized as well
- ASLR is an important companion to NX – makes NX subversion tricks very difficult.

Other Vulnerabilities

Format String Vulnerabilities

Passing user-supplied data directly to a function in the**printf()*-family function is dangerous.

Dangerous calls can be identified by an argument deficiency.

Good	• printf(“%s“, inputdata);
Bad	• printf(inputdata);

The problems with:

```
printf(inputdata);
```

An attacker could set `inputdata= “...%x...”` and possibly view the contents of the stack (if the output of `printf` is user-visible

An attacker could set `inputdata=“…%s%s%s…”` and crash the application by forcing it read from an arbitrary address on the stack

An attacker could set `inputdata=“…%n…”` and write data to memory to gain control over the application (think back to the buffer overflow lab)

Solution strategies

Golden rule of format functions:

Explicitly set format specifiers or users will do it for you.

Integer Overflows

```
public static void main(String[] args) {
  short num = 32765;
  int big_num = 32765;
  for (int i=1;i<10;i++){
          System.out.println("num = " +
          num++ + " big_num = "    +
          big_num++);
  }
}
```

SQL Injection

- SQL Injection takes advantage of user data being concatenated with SQL commands
- Attackers can effectively insert code and modify the SQL statement
- These commands are then passed to the database

Solution strategies

- Use parameterized stored procedures (don't allow user data to "escape" out of the SQL command)
- User server-side input filtering (regular expressions are great for this)
- Combining both strategies helps to minimize 2nd order attacks (discussed later)