Chapter 11
Software Security

Software Security Issues

- Many vulnerabilities result from poor programming practices.
- Consequence from insufficient checking and validation of data and error codes.

Software error categories:
- Insecure interaction between components
- Flawly resource management
- Poisons defenses

Software Security, Quality and Reliability

- Software quality and reliability:
  - Concerned with the accidental damage of an program on a result, wrong, unanticipated, or use of incorrect code.
  - Improving using structured design and testing to identify and eliminate as many bugs as possible from a program.
  - Concern is not how many bugs, but how often they are triggered.

- Software security:
  - Attackers choose probability distribution specifically targeting bugs that result in failure that can be exploited by the attacker.
  - Triggered by inputs that differ dramatically from what is usually expected.
  - Likely to be identified by common testing approaches.

Defensive Programming

- Designing and implementing software so that it continues to function even when under attack.
- Requires attention to all aspects of program execution, environment, and type of data it processes.
- Software is able to detect erroneous conditions resulting from some attack.
- Also referred to as secure programming.
- Key rule is to never assume anything, check all assumptions and handle any possible error states.
Defensive Programming

Programmers often make assumptions about the type of inputs a program will receive and the environment it executes in. Assumptions need to be validated by the program and all potential failures handled gracefully and safely. Requires a changed mindset to traditional programming practices. Programmers have to understand how failures can occur and the steps needed to reduce the chance of them occurring in their programs.

Security by Design

- Security and reliability are common design goals in most engineering disciplines.
- Software development not as mature.
- Recent years have seen increasing efforts to improve secure software development processes.
- Software Assurance Forum for Excellence in Code (SAFECode):
  - Developers show outlining industry best practices for software assurance and providing practical advice for implementing proven methods for secure software development.

Handling Program Input

- Incorrect handling of insecurely validated input:
  - Users may input sensitive data from external sources, which may not be validated as expected by the program.
  - Malicious input can lead to data corruption or breaches.
  - Explicitly validate assumptions on data and type of input before use.

Input Size & Buffer Overflow

- Programmers often make assumptions about the maximum expected size of input.
  - Incorrect buffer size is not confirmed.
  - Resulting in buffer overflow.
- Testing may not identify vulnerability.
  - Testing is unlikely to include large enough inputs to trigger the overflow.
- Safe coding treats all input as dangerous.

Interpretation of Program Input

- Program input may be binary or text.
  - Binary interpretation depends on encoding and is usually application specific.
- There is an increasing variety of character sets being used.
  - Care is needed to identify which set is being used and what characters are being read.
- Failure to validate may result in an exploitable vulnerability.
- 2014 Heartbleed OpenSSL bug is a recent example of a failure to check the validity of a binary input value.
Injection Attacks

- Flows relating to invalid handling of input data, specifically when program input data can accidentally or deliberately influence the flow of execution of the program
  
  Most often occur in scripting languages

  - Encourage reuse of other programs and system utilities where possible to save coding effort
  - Often used as Web CGI scripts

- Vulnerable PHP code
- Safe PHP code

Figure 11.3 SQL Injection Example

Cross Site Scripting (XSS) Attacks

- Attack where special data fed into one script is displayed to another user

  Commands sent in accordance with application
  - Introduce malicious content into web application
  - Interpret content as part of script
  - User's browser interacts with content as if they were a part of the site

  XSS reflection vulnerability

  Attackers include the following:
  - Introduce malicious content into web application
  - User interacts with content as if it was part of the site

Figure 11.5 XSS Example
Validating Input Syntax

- It is necessary to ensure that input data is stored and processed correctly.
- Input data should be converted to a format that is used.
- The input data should be validated.
- If only simple input data is used, the program is more likely to remain secure.

Alternate Encodings

- May have multiple means of encoding text.
- Unicode is used for Internet standardization.
- UTF-8 encodes 1-4 byte sequences, making it suitable for any encoding standard.
- Canonicalization:
  - Transforms input data into a single, standard, minimal representation.
  - Even if this is done for input data, it can be converted with a single representation of all acceptable input values.

Validating Numeric Input

- Additional concern when input data represents numeric values.
- Internally stored in fixed size values 8, 16, 32, 64-bit integers.
- Floating point numbers depend on the processor used.
- Must correctly interpret text input and process consistently.
  - Some issues comparing signed vs. unsigned.
  - Could be used for: threshold buffer overflow check.

Input Fuzzing

- Developed by Professor Barton Miller at the University of Wisconsin Madison in 1989.
- Software testing technique that uses randomly generated data to inputs to a program.
  - Range of inputs is very large.
  - Input is to determine program or function correctly handles abnormal inputs.
  - Ensures security as well as reliability.
  - Can also use templates to generate classes of known problem inputs.

Correct Algorithm Implementation

- If the correct program development technique is used, the correctness of the program can be improved.
- Consequence of inaccuracy is a bug in the resulting program.

Security issues:

- Correct algorithm implementation.
- Correct machine instructions for algorithm.
- Valid manipulation of data.
Ensuring Machine Language Corresponds to Algorithm

- Issue is ignored by most programmers
  - Assumption is that the compiler or interpreter generates or executes code that validly implements the language statements
  - Requires comparing machine code with original source
    - Slow and difficult
- Development of computer systems with very high assurance level is the one area where this level of checking is required
  - Specifically Common Criteria assurance level of EAL 7

Correct Use of Memory

- Issue of dynamic memory allocation
  - Used to manipulate unknown amounts of data
  - Allocated when needed, released when done
- Memory leak
  - Steady reduction in memory available on the heap to the point where it is completely exhausted
  - Crash
- Many older languages have no explicit support for dynamic memory allocation
  - Use standard library routines to allocate and release memory
- Modern languages handle automatically

Operating System Interaction

- Programs execute on systems under the control of an operating system
  - Mediates and shares access to resources
    - Constructs execution environment
    - Includes environment variables and arguments
- Systems have a concept of multiple users
  - Resources are owned by a user and have permissions granting access with various rights to different categories of users
  - Programmers need access to various resources, however excessive levels of access are dangerous
  - Concerns when multiple programs access shared resources such as a common file

Correct Data Interpretation

- Data stored as bits/bytes in computer
  - Grouped as words or longwords
  - Accessed and manipulated in memory or copied into processor registers before being used
  - Interpretation depends on machine instruction executed
- Different languages provide different capabilities for restricting and validating interpretation of data in variables
  - Strongly typed languages are more limited, safer
  - Other languages allow more liberal interpretation of data and permit program code to explicitly change their interpretation

Race Conditions

- Without synchronization of accesses it is possible that values may be corrupted or changes lost due to overlapping access, use, and replacement of shared values
- Arise when writing concurrent code whose solution requires the correct selection and use of appropriate synchronization primitives
  - Deadlock
    - Processes or threads wait on a resource held by the other
    - One or more programs has to be terminated

Environment Variables

- Collection of string values inherited by each process from its parent
  - Can affect the way a running process behaves
  - Included in memory when its created
- Can be modified by the program process at any time
  - Modifications will be passed to its children
- Another source of untrusted program input
- Most common use is by a local user attempting to gain increased privileges
  - Goal is to subvert a program that grants supervisory or administrative privileges

Command injection

- Function calling/indirectly understanding exactly what they do.

Environment Variables

- Collection of string values inherited by each process from its parent
  - Can affect the way a running process behaves
    - Included in memory when its created
- Can be modified by the program process at any time
  - Modifications will be passed to its children
- Another source of untrusted program input
  - Most common use is by a local user attempting to gain increased privileges
    - Goal is to subvert a program that grants supervisory or administrative privileges

*path* : where are the executables?

Dynamic Variables
Vulnerable Compiled Programs

Programs can be vulnerable to PATH variable manipulation
- Must reset to "safe" values

If dynamically linked may be vulnerable to manipulation of LD_LIBRARY_PATH
- Used to locate suitable dynamic library
- Must either statically link privileged programs or prevent use of this variable

Use of Least Privilege

Privilege escalation
- Exploit flaws may give attacker greater privileges

Least privilege
- Run programs with least privilege needed to complete their function

Determine appropriate user and group privileges required
- Decide whether to grant extra user or group privileges

Ensure that privileged program can modify only those files and directories necessary

Root/Administrator Privileges

Programs with root/administrator privileges are a major target of attackers
- They provide elevated level of system control and access to protected system resources

Often privilege is only needed at start
- Can then run as normal user

Good design partitions complex programs into smaller modules with needed privileges
- Provides a greater degree of protection by limiting the impact of a security breach in one component
- Easier to test and verify

System Calls and Standard Library Functions

Programmers make assumptions about their operation
- If incorrect behavior is not what is expected
- May be arrest of system, e.g., reducing access to shared resources
- Results in requests for service, e.g., being sent to inappropriate places or otherwise unsuitable places
- May optimize system use
- Optimizations may conflict with program goals

Figure 11.6 Vulnerable Shell Scripts

Figure 11.7 Example Global Data Overflow Attack
Preventing Race Conditions

- Programs may need to access a common system resource
- Need suitable synchronization mechanisms
  - Most common technique is to acquire a lock on the shared file
- Lockfile
  - Process must create and own the lockfile in order to gain access to the shared resource
  - Concerns
    - If a program chooses to ignore the existence of the lockfile and access the shared resource, the system will not prevent this
    - All programs using this form of synchronization must cooperate
    - Implementation

Safe Temporary Files

- Many programs use temporary files
- Often in common, shared system area
- Must be unique, not accessed by others
- Commonly create name using process ID
  - Unique, but predictable
  - Attacker might guess and attempt to create own file
- Secure temporary file creation and use requires the use of random names

Other Program Interaction

- Programs may use functionality and services of other programs
  - Security vulnerabilities can result unless care is taken with this interaction
  - Such issues are of particular concern when the program being used may not adequately identify all the security concerns that might arise
- Burden falls on the newer programs to identify and minimize any security issues
  - Each program needs to ensure that it is protected
- Issues of data confidentiality and integrity
- Detection and handling of exceptions and errors generated by interaction is also important from a security perspective

Handling Program Output

- Final component is program output
  - May be stored for future use, sent over net, displayed
  - May be binary or text
- Important from a program security perspective that the output conform to the expected form and interpretation
- Programs must identify what is permissible output content and filter any possibly untrusted data to ensure that only valid output is displayed
- Character set should be specified
PA3: due 12/4

1) Buffer overflows, pathnames, SQL injection

2) Pathname attack
   (CGI script) Fingerprint read memos
   etc/shadow
   How did they get root access via CGI script?

3) SQL injection

Details:
- Only 1 machine
- Buffer overflow
  1) Valid request to webserver
  2) Crafted request: buffers to overflow using "Get" request

- Interacting with the operating system and other programs
  a) Environment variables
  b) Implementing shell scripts
  c) Using standard library functions
  d) Preventing race conditions with input/output
  e) Safe temporary file use
  f) Interacting with other programs

- Handling program input
  a) Input size and buffer overflow
  b) Interpretation of programs
  c) Handling input/output
  d) Input processing

- Writing safe program code
  a) Correct algorithm implementation
  b) Ensuring that machine language corresponds to program
  c) Validating input data values
  d) Correct use of memory
  e) Preventing race conditions with temporary memory

- Software security issues
  a) Introducing software security and defensive programming

Summary

11/12/2015