Software Security
Lots to Learn

- As a software professional, you need to educate yourself on how to build secure software.
- We will cover the tip of the iceberg in CS 465.
- I suggest you read several books on the topic, and make it a habit to stay current with the latest thinking in this area.
Recommended Books

19 Deadly Sins of Software Security
Howard, LeBlanc, and Viega

Building Secure Software
Viega and McGraw

Writing Secure Code, 2nd Edition
Howard and LeBlanc
Genesis of the 19 Deadly Sins

- The remaining content of these slides is taken from *19 Deadly Sins of Software Security*
- In early 2004, Amit Yoran (Director of National Cyber Security Division, Department of Homeland Security) announced that 95% of software security bugs arise from 19 programming mistakes
- Learn to avoid these mistakes!
Sin 1 – Buffer Overflow

- Do use compiler-based defenses such as /GS and ProPolice
- Do not think that compiler and OS defenses are sufficient – they are not; they are simply extra defenses
- Do not create new code that uses unsafe functions
- Consider removing unsafe functions from old code over time
- Consider using C++ string and container classes rather than low-level C string functions
What is it? A recent issue discussed publicly since only 2000.

- In C/C++, a format string bug can allow an attacker to write to arbitrary memory locations
- Example `printf(user_input_without_validation)`
- When a user inputs “%x %x”, the output is data on the stack
- The %n designation writes the number of characters written so far to the address of the variable in the corresponding argument

Download Chapter 2 with permission at

http://www.devx.com/security/Article/29418/1763

http://doc.bughunter.net/format-string/exploit-fs.html

http://www.sans.org/resources/malwarefaq/LPRng.php
Sin 2 – Format String Problems

- Do use fixed format strings, or format strings from a trusted source
- Do not pass user input directly as the format string
- Consider using less vulnerable high-level languages
Sin 3 – Integer Overflows

- Integer overflow and underflow, and arithmetic overflows of all types can cause crashes, logic errors, escalation of privileges, and execution of arbitrary code.

- Required reading on the lecture page
Sin 3 – Integer Overflows

- Do check all calculations used to determine memory allocations and array indexes for overflow flaws in the arithmetic.
- Do use unsigned integers for array offsets and memory allocation sizes.
- Do not think this problem only applies to C/C++.
Sin 4 – SQL Injection

- Typically the result of an attacker providing malformed data to an application that uses it to construct an SQL command
  - `exec (@query)`
  - User inputs an ID and the system constructs a command
    `SELECT @query = 'select ccnum from cust where id = '' ' + @id + ' '''`
  - The attacker can add extra data and a comment character
    `1 or 2>1` -
  - The result is that the query returns the entire customer table
Sin 5 – Command Injection

- Command injection problems occur when untrusted data is passed to a compiler or interpreter that might execute the data if it is formatted in a certain way.

- Example
  - How to Remove Meta-characters From User-Supplied Data in CGI Scripts
    - [http://www.cert.org/tech_tips/cgi_metacharacters.html](http://www.cert.org/tech_tips/cgi_metacharacters.html)
Sin 5 – Command Injection

- Do validate user input before passing it to a command processor
- Do handle the failure securely if an input validation check fails
- Do not use the deny list approach, unless you are 100 percent sure you are accounting for all possibilities
- Consider avoiding regular expressions for user input validations; instead write a simple and clear validators by hand
Sin 6 – Failing to Handle Errors

- When a programmer fails to handle an error, the program could get into an insecure state or crash.
- Program termination opens up a denial of service vulnerability.
- Revealing too much information about an error can aid an attacker.
Sin 6 – Failing to Handle Errors

- Do check the return status of every security-related function
- Do make every attempt to recover from an error gracefully to avoid DOS attacks
- Do not catch all exceptions without a very good reason, as you may be masking errors in the code
- Do not leak error information to untrusted users
Sin 7 – Cross-Site Scripting

- Cause – a web application takes user input and echoes that input directly in a web page
- If the input is a script language (e.g., JavaScript), then the input could be interpreted in the browser
- An attacker can create a malformed query string for a vulnerable web site, and then gets a victim to click on the link in their own browser
  - Example – forward a cookie with sensitive information to the attacker
Sin 7 – Cross-Site Scripting

- Do check all web-based input for validity and trustworthiness
- Do HTML encode all output originating from user input
- Do not echo web-based input without checking for validity first
- Do not store sensitive data in cookies
Sin 8 – Failing to Protect Network Traffic

- Opens up users to the following attacks
  - Eavesdropping
  - Replay
  - Spoofing
  - Tempering
  - Hijacking
Sin 8 – Failing to Protect Network Traffic

- Do use a strong initial authentication scheme
- Do perform ongoing message authentication
- Do encrypt all data that is sensitive
- Do use TLS for your on-the-wire protocols
- Do not hardcode keys
- Do not ignore the security of your data on the wire
Sin 9 – Use of Magic URLs and Hidden Form Fields

- An application encodes authentication information in a URL and sends it in the clear.
- The server stores information in a hidden field and assumes the user cannot see it or tamper with it.
  - Some web sites have included the price in a hidden field and used that value to process a transaction.
Sin 9 – Use of Magic URLs and Hidden Form Fields

- Do test all web input with malicious input
- Do not embed confidential data in any HTTP or HTML construct if the channel is not encrypted
- Do not trust any data in a web form
- Do not think the application is safe just because you use cryptography
Sin 10 – Improper Use of SSL/TLS

- Do use the latest version
- Do use a certificate allow list
- Do verify the certificate – integrity, ownership, expiration, revocation
- Do not only check the name in a certificate – anyone can place any name in a certificate
Sin 11 – Use of Weak Password-Based Systems

Examples

- TENEX bug that leaked information via the virtual memory system
- Paris Hilton Hijacking
  - Attacker reset her password by answering her “secure” question – what is the name of your pet?
Sin 11 – Use of Weak Password-Based Systems

- Do ensure passwords are not sent in the clear
- Do give a single error message for failed login attempts
- Do log failed password attempts
- Do provide a secure mechanism for people to change passwords
- Do make it easy for customer support to reset a password over the phone
- Do not store plaintext passwords on the server
- Do not store passwords in code
- Do not log the failed password
- Do not allow short passwords
Sin 12 – Failing to Store and Protect Data Securely

- Do think about access control
- Do realize that some data is so sensitive it should never be stored on a general purpose, production server
- Do leverage the operating system capabilities to secure secret and sensitive data
- Do use appropriate permissions if you must store sensitive data
- Do remove the secret from memory once you have used it
- Do scrub the memory before you free it
Sin 12 – Failing to Store and Protect Data Securely

- Do not create world-writeable objects in Linux, MAC OS X, and UNIX
- Do not crate objects with Everyone (Full Control) or Everyone (Write) access control entries
- Do not store key material in a demilitarized zone
- Do not embed secret data of any kind in your application. This includes passwords, keys, and database connection strings
- Do not create your own “secret” encryption algorithms
- Consider using encryption to store information that cannot be properly protected by an ACL, and signing to protect information from tampering
- Consider never storing secrets in the first place – can you get the secret from the user at run time instead?
Sin 13 – Information Leakage

- An attacker obtains data that leads to a security breach
- Do define who should have access to what error and status information
- Do use operating system defenses such as ACLs and permissions
- Do use cryptography to protect sensitive data
- Do not disclose system status info to untrusted users
- Consider using other less commonly used operating system defenses such as encrypted file systems
- Consider using cryptography implementations specifically hardened against timing attacks
- Consider using the Bell-LePadula model
Sin 14 – Improper File Access

- Race conditions – vulnerability after making a security check on a file
  - TOCTOU – Time of check, time of use
- An attacker manipulates a path name to overwrite important files
Sin 14 – Improper File Access

- **Do be strict about what you will accept as a valid filename**
- **Do not blindly accept a filename thinking it represents a valid file – especially on server platforms**
- **Consider storing temporary files in the user’s temporary directory, not in a shared location. This may help run your application in least privilege mode.**
Sin 15 – Trusting Network Naming Resolution

- Do use cryptography to establish the identity of your clients and servers
- Do not trust DNS information – it isn’t reliable!
- Consider specifying IPSec for the systems your application will run on
Sin 16 – Race Conditions

- Do write code that doesn’t depend on side effects
- Do be very careful when writing signal handlers
- Do not modify global resources without locking
- Consider writing temporary files into a per-user store instead of a world-writable space
Sin 17 – Unauthenticated Key Exchange

- Do realize that key exchange alone is often not secure. You must authenticate the other party.
- Do use off-the-shelf solutions for session establishment, such as SSL/TLS
- Do ensure that you read all the fine print to make sure you have strongly authenticated every party
- Consider calling in a cryptographer if you insist on using custom solutions
Sin 18 – Cryptographically Strong Random Numbers

- Do use the system cryptographic pseudo-random number generator (CRNG) when at all possible.
- Do make sure that any other cryptographic generators are seeded with at least 64 bits of entropy, preferably 128 bits.
- Do not use a noncryptographic pseudo-random number (PRNG) generator.
- Consider using hardware random number generators (RNG) in high assurance situations.
Sin 19 – Poor Usability

- Do understand your users’ security needs, and provide the appropriate information to help them get their jobs done
- Do default to a secure configuration whenever possible
- Do provide a simple and easy to understand message, and allow for progressive disclosure if needed by more sophisticated users or admins
- Do make security prompts actionable
- Do not dump geek-speak in a big honking dialog box. No user will read it.
- Do not make it easy for users to shoot themselves in the foot – hide options that can be dangerous
- Consider providing ways to relax security policy selectively, but be explicit and clear about what the user is choosing to allow