CSCI 455 Principles of Computer Security

Secure Coding: Best Practices List from Professor Howles

1. Always compile with full compiler warnings enabled
2. Integer errors: Using gcc, compile with –Wconversion and –Wsign to identify possible type conversion or sign errors; enable runtime error checking –ftrapv to trap overflow
3. Be sure all input from untrusted sources is scrubbed; consider every source untrusted
4. Reject bad data; do not try to correct
5. Watch the data content and the lengths
6. Look for metacharacters or escape sequences
7. Buffer management: Be sure all operations are safe from overflow
8. Check for the use of unsafe libraries/functions
9. If using safe libraries, check operations; programmers may not be very familiar using them
10. Always check C-style strings for the null terminator; since strings continue to be a source of exploits, explicitly set the null each time you copy the string
11. Always use static format strings for print statements
12. No hardcoded or sensitive data
13. No references to server names, user names or other internal resources
14. Temporary file names are not easily guessed
15. Temporary files and all other resources are cleaned up upon exit – best if files are closed and resources deallocated as soon as no longer needed
16. Error or exception messages to not disclose sensitive data
17. File names to not disclose device names or full paths
18. Check for every error condition
19. Value being secure over being robust. If an unexpected or misunderstood event occurs, it may be safest to stop/exit than to try to continue
20. Look for off-by-one errors (loops, arrays, C-style strings)
21. Test for overflow or sign errors
22. Look for unsafe pointer operations
23. Do not store flat-text sensitive data
24. Avoid calling functions with an arbitrary number of arguments
25. When transmitting over a network, be sure a valid connection is maintained the entire time
26. Always run with the least privilege
27. Be sure that calls to release memory where sensitive data was stored are not optimized out
28. Do not develop your own crypto code
29. Use strong crypto routines
30. Require that user passwords/authentication are long, strong and random
31. Be sure your random number generator is *really* a random number generator
32. Keep security routines or security critical sections short, simple; and easy to read and understand
33. Check all arguments for type, length and context
34. Check for instances where you may not have allowed for the NULL string terminator
35. Check all function return values, even those you don't expect the function to fail
36. When creating child processes be aware of open files when the child is created
37. Always use fully qualified path names for any open files – relative path names are easy to subvert
38. Avoid execlp and execvp; do not use inherited environment variables from the parent process
39. C and C++ are the hacker's favorite languages – keep that in mind
40. Complexity == defects
41. When finding defects, check for the same defect in different modules or applications
42. When static checking, use monitoring tools that examine processes as they interact with the OS
43. Using free software? Are you sure it's safe? How will you certify the code?