Foundations of Cybersecurity
C and C++ Secure Coding

Gianluca Dini
Dept. of Information Engineering
University of Pisa
Email: gianluca.dini@unipi.it
Version: 2021-03-03

1

Credits

- These slides come from a version originally produced by Dr. Pericle Perazzo

2

Secure coding

A BUNCH OF DEFINITIONS

3

Secure Coding

- Programming errors which caused the most common/dangerous vulnerabilities
- Remediation best practices
- Risk assessment
  - Exploitation probability
  - Impact
  - Remediation cost
- Objectives:
  - Protect customers
  - Limit patches

4

Secure coding studies those programming errors which have caused the most common, dangerous, and disruptive software vulnerabilities in the past; the remediation best practices; and the *risk assessment*, i.e., the probability that these errors are exploitable, the possible impact, and the remediation cost. Secure coding aims at protecting customers from money loss due to security incidents and limiting the patch releases of software.

Secure Coding

- Strongly language-dependent
- C/C++ are particularly error-prone
  - Intended to be lightweight
  - Power-to-the-programmer philosophy
- C/C++ still broadly used
  - Embedded devices
  - High-load servers
  - Legacy code

5

Secure coding is strongly language-dependent. As a general rule, the lower the programming language level is, the more error prone, and thus the more susceptible to vulnerabilities. C and C++ languages are particularly error-prone, because they are intended to be lightweight and to produce a small codeprint. The programmer, especially if coming from higher-level languages like Java, may assume that the C language implicitly perform checks when it does not. Moreover, C and C++ follow the *power-to-the-programmer* philosophy, that they do not prevent the programmer from doing what needs to be done. So, they always consider the programmer to be fully aware of the consequences of the code he is writing. Despite their poor security characteristics, C and C++ are widely used still today, in particular when performances are a requirement (e.g., constrained/embedded devices or high-load servers) and when legacy code must be maintained.

Undefined Behavior

- Undefined behavior: C/C++ gives no requirement
  - Out-of-bound buffer access
  - Null pointer dereferencing
  - Signed integer overflow
- Unspecified behavior: C/C++ gives multiple possibilities
  - Argument evaluation order in function calls
- Unexpected behavior: well-defined behavior unanticipated by the programmer

6

An *undefined behavior* is a behavior on which the C/C++ standard poses no requirements, for example in case of out-of-bound buffer access, null pointer dereferencing, or signed integer overflow. An *unspecified behavior* is a behavior on which the C/C++ standard gives two or more possibilities, for example the order of argument evaluation in a function call. An *unexpected behavior* is a well-defined behavior, yet unexpected by the programmer, due to incorrect programming assumptions. Most software vulnerabilities is based on undefined behaviors. The attacker tries to induce the program to perform undefined behaviors by means of special crafted inputs. Then, the attacker tries to damage somehow the system. Undefined behaviors must be avoided. Unspecified behavior must be known. Unexpected behaviors must be expected.

Taint Analysis Terminology

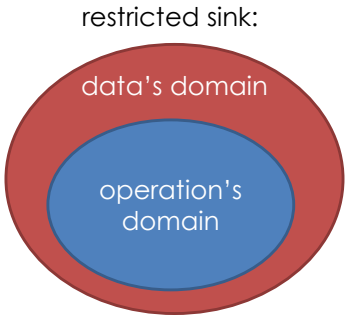
UNIVERSITÀ DI PISA

- Tainted data:

Not sanitized data from an external source

- Operations on tainted data gives tainted data
- Restricted sink:

Operand/argument with domain smaller than its type domain



7

Data that comes from a source external to the program (network, user input, file, command line arguments, environment variables, other software, etc.) and that has not been sanitized yet is called *tainted data*. The result of an operation over tainted data is tainted data too. An operand or a function argument whose domain (i.e., the set of valid values) is a subset of the domain of its type (i.e., the set of all the possible values) is called *restricted sink*. Undefined/unspecified/unexpected behaviors happen when tainted data is given as input to a restricted sink.

Sanitization

- Sanitization removes taint from data
  - By replacement: replace out-of-domain values with in-domain values
  - By termination: terminate execution path

8

Sanitization is an operation that removes the taint from a data. Sanitization can be done by *replacement* or *termination*. Replacement replaces out-of-domain values with in-domain values, while termination terminates the execution path of the entire program or the current operation.