Fundamentos del Software: introducción a la programación verificada

Programación segura

Buenas prácticas de programación

Roberto Blanco[†] & Ricardo J. Rodríguez[‡]

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

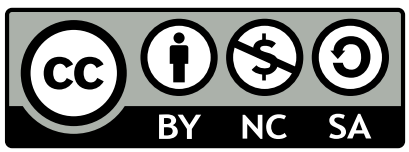
Universidad Zaragoza

[†]Max Plank Institute for Security and Privacy
Bochum, Alemania

[‡]Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza (España)

Septiembre 2020

Universidad de Zaragoza



Índice

Universidad Zaragoza

Índice

Universidad Zaragoza

Motivación

- **Los desarrolladores tienen fallos cuando escriben software**
  - Si no se detectan, pueden llevar a vulnerabilidades que comprometan el sistema
- **Desarrollo de código seguro: minimizar vulnerabilidades en el código**
- **Diferentes formas**:
  - Definir estándar de programación
  - Seleccionar un lenguaje apropiado (y seguro)
  - Uso de entornos y librerías específicas
  - Uso de herramientas de análisis automático de código
  - Auditorías del código (mediante revisión manual)

Universidad Zaragoza
1542

Índice

Universidad Zaragoza

Estándares de programación

Secure Development Lifecycle

- **Entender los problemas de seguridad de la tecnología usada y decidir cómo afrontarlos**
- **Estándares y convenciones de programación**:
  - **Permiten escribir código seguro**
  - Características de seguridad intrínsecas que pueden crearse y comunicarse a todo el equipo de desarrollo
- **Cuanto más desacoplado esté un componente en el sistema, mejor**
  - Facilita que pueda reemplazarse o actualizarse si es necesario

Universidad Zaragoza
1542

Estándares de programación

- Los estándares tienen que ser **realistas y realizables**
- **Importante**: fases de pruebas y validación
  - Herramientas automáticas vs. manual para comprobar que se adhiere al estándar escogido

Universidad Zaragoza

Estándares de programación

Open Web Application Security Project (OWASP)

- **Comunidad que produce (libremente accesibles) artículos, metodologías, documentación, herramientas y otras tecnologías relacionadas con seguridad de aplicaciones web**
- Fundado en 2001
- **OWASP Foundation**
  - Organización sin ánimo de lucro 501(c)(3) (en EEUU)
  - Creada en 2004 para soportar la infraestructura y los proyectos de OWASP

Universidad Zaragoza

Estándares de programación

Open Web Application Security Project (OWASP)

OWASP Secure Coding Practices – Quick Reference Guide

(`https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf`)

- Los defectos de seguridad en software que pueden aparecer durante el desarrollo de software:
  - No identificar los requisitos de seguridad con anterioridad
  - Creación de diseños conceptuales con errores lógicos intrínsecos
  - Uso de guías de programación que introducen vulnerabilidades
  - Despliegue de software de manera incorrecta
  - Fallos introducidos durante la fase de mantenimiento o actualización
- Proporciona listas de comprobación de diferentes categorías, a modo de "auditoría rápida"

Universidad Zaragoza

Estándares de programación

Open Web Application Security Project (OWASP)

Categorías definidas

- *Input Validation*
- *Output Encoding*
- *Authentication and Password Management*
- *Session Management*
- *Access Control*
- *Cryptographic Practices*
- *Error Handling and Logging*
- *Data Protection*
- *Communication Security*
- *System Configuration*
- *Database Security*
- *File Management*
- *Memory Management*
- *General Coding Practices*

Universidad Zaragoza
1542

Estándares de programación

OWASP Top 10 2017

A1:2017-Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2:2017-Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
A3:2017-Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
A4:2017-XML External Entities (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
A5:2017-Broken Access Control	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
A6:2017-Security Misconfiguration	Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.
A7:2017-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8:2017-Insecure Deserialization	Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
A9:2017-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
A10:2017-Insufficient Logging & Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

Estándares de programación
Desarrollo seguro con JAVA SE

Oracle Secure Coding Guidelines for Java SE

(`http://www.oracle.com/technetwork/java/seccodeguide-139067.html`)

- *Fundamentals*
- *Denial of Service*
- *Confidential Information*
- *Injection and Inclusion*
- *Accessibility and Extensibility*
- *Input Validation*
- *Mutability*
- *Object Construction*
- *Serialization and Deserialization*
- *Access Control*

Universidad Zaragoza
1542

Estándares de programación
SEI CERT

SEI CERT Coding Standards

(https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards

- **CERT: Computer Emergency Response Team**
- SEI CERT es el CERT de Carnegie Mellon University's Software Engineering Institute
  - Autoridad mundial reconocida dedicada a la mejora de seguridad de sistemas y redes
  - **Objetivo: reducir el número de vulnerabilidades** a un nivel tal que se puedan mitigar
    - Evitar errores durante la programación
    - Descubrir y eliminar fallos de seguridad durante la implementación y pruebas

Universidad Zaragoza

Estándares de programación

SEI CERT Top 10+ Secure Coding Practices

1 *Validar entradas*

- Validar las entradas de datos de todas aquellas fuentes no confiables
- Una validación de entrada adecuada puede eliminar la mayoría de vulnerabilidades

2 Prestar atención a los avisos del compilador

- Compilar el código usando los niveles más altos del compilador y eliminar todos los avisos, modificando el código
- Uso de herramientas de análisis estático y dinámico para eliminar fallos de seguridad no detectados en compilación

3 Diseñar pensando en políticas de seguridad

- Crear una arquitectura software y diseñar software para implementar y cumplir políticas de seguridad

4 Simplicidad

- El diseño tendría que ser lo más simple y pequeño posible
- Los diseños complejos incrementan la probabilidad de que existan errores durante la implementación, configuración o el uso

5 Denegación por defecto

- Basar las decisiones de acceso en permisos. Por defecto, se deniega el acceso y el control de acceso identifica las condiciones necesarias para que el acceso sea permitido

Estándares de programación

SEI CERT Top 10+ Secure Coding Practices

6 Adherirse al principio de mínimo privilegio

- Cada proceso debe de ejecutarse con el mínimo privilegio que sea necesario para realizar su función
- Cualquier permiso privilegiado debería de darse únicamente el tiempo necesario para realizar esa actividad privilegiada

7 Sanitizar datos mandados a otros sistemas

- Sanitizar los datos mandados a otros subsistemas, como terminales, bases de datos, u otras componentes tipo COTS

8 Defensa en profundidad

- Gestionar el riesgo con múltiples estrategias defensivas, de modo que si una capa de defensa es inadecuada, exista otra que logre prevenir que el fallo se convierta en vulnerabilidad y/o limitar las consecuencias de una explotación exitosa

9 Uso efectivo de técnicas *quality assurance*

- Útiles para identificar y eliminar vulnerabilidades
- Fuzzing, penetration testing, y auditorías de código deberían de ser técnicas parte de la *quality assurance*
- Revisiones de seguridad independientes ayudan a obtener sistemas más seguros

10 Adoptar un estándar de codificación segura

- Desarrollar y/o aplicar un estándar de codificación segura adecuado para el lenguaje de programación y plataforma objetivos

Estándares de programación
SEI CERT Top 10+ Secure Coding Practices

Bonus track

1 Definir requisitos de seguridad

- Identificar y documentar los requisitos de seguridad tan pronto como sea posible en el ciclo de desarrollo del software
- Evaluar de manera periódica (durante el desarrollo) que el sistema sigue cumpliendo con los requisitos

2 Modelar amenazas

- Permite anticiparse a las posibles amenazas que sufrirá el sisteam
- Identificar los activos, descomponer la aplicación, identificar y categorizar las amenazas a cada activo o componente y analizar los riesgos
- Desarrollar estrategias de mitigación de amenazas e implementarlas en diseño, desarrollo, y pruebas

Universidad Zaragoza
1542

Estándares de programación
SEI CERT

The easiest way to break system security is often to circumvent it rather than defeat it

Estándares de programación
SEI CERT Coding Standard

- **Reglas y recomendaciones agrupados en categorías**
- **Reglas**:
  1. Violación de una práctica de programación que provoca un fallo de seguridad potencialmente explotable
  2. Existen ciertas condiciones excepcionales en los que esta violación es necesaria para el funcionamiento normal del programa
  3. El cumplimiento se puede verificar fácilmente
- **Recomendaciones**
  - Su aplicación probablemente mejore la seguridad del sistema
  - Útiles cuando existen uno o más requisitos de una regla no se pueden cumplir
- **Descripción de la nomenclatura**
  - **Acrónimo de la categoría (3 letras) + número de la regla o recomendación (2 dígitos) + guión (-) + lenguaje de programación referido**
  - **Las recomendaciones empiezan en 00, mientras que las reglas empiezan en 30**
  - Ejemplo: INT15-C (recomendación 15 para los enteros en lenguaje C)

Universidad Zaragoza
1542

Estándares de programación
SEI CERT Coding Standard

Categorías

1. Preprocessor (PRE)
2. Declarations and Initialization (DCL)
3. Expressions (EXP)
4. Integers (INT)
5. Floating Point (FLP)
6. Arrays (ARR)
7. Characters and Strings (STR)
8. Memory Management (MEM)
9. Input Output (FIO)
10. Environment (ENV)
11. Signals (SIG)
12. Error Handling (ERR)
13. ApplicationProgramming Interfaces (API)
14. Concurrency (CON)
15. Miscellaneous (MSC)

Universidad Zaragoza
1542

Estándares de programación
SEI CERT Coding Standard

Algunos ejemplos

- INT00-C. *Understand the data model used by your implementation(s)*
- INT02-C. *Understand integer conversion rules*
- INT30-C. *Ensure that unsigned integer operations do not wrap*
- STR30-C. *Do not attempt to modify string literals*
- STR34-C. *Cast characters to unsigned char before converting to larger integer sizes*

Universidad Zaragoza
1542

Índice

Universidad Zaragoza

Uso de funciones seguras

- Muchos lenguajes de programación tienen **funciones con alto impacto en la seguridad**
  - Cuando se desarrollaron no se apreciaron estos problemas, pero ahora se conocen
- Por ejemplo, **funciones de manejo de cadenas y buffers en C/C++**
- Otros lenguajes también sufren de estas funciones inseguras:
  - JavaScript y PHP (las funciones que generan nuevo código en tiempo de ejecución)
- **Un estándar de codificación segura proporciona a los desarrolladores información acerca de estas funciones inseguras**

Universidad Zaragoza

Uso de funciones seguras

- La mayoría de los compiladores o IDEs integran ya **mecanismos para notificar a los desarrolladores si detectan una función potencialmente insegura**
- En el caso de C/C++, es útil la **librería de Microsoft** `banned.h`
  - Disponible en `https://www.microsoft.com/security/blog/2012/08/30/microsofts-free-security-tools-banned-h/`
  - Simplemente añadiéndola en el proyecto (`#include "banned.h"`), provoca que el compilador falle si se detecta alguna función insegura

Universidad Zaragoza

Índice

Universidad Zaragoza

Compilación segura

- **Usar siempre las últimas versiones de los compiladores, enlazadores, e intérpretes**
- Existen **mecanismos de seguridad incorporados a nivel de compilación**, incorporados tanto a nivel estático como dinámico
  - Hay que conocer qué opciones del compilador permiten incorporarlas, y añadirlas a la *toolchain*
- **Activar todas las opciones de compilación segura** permitirá obtener un software más protegido, donde sea más complicado (que no imposible) explotar una vulnerabilidad
- **Evitar desactivar mecanismos de seguridad por temas de rendimiento o compatibilidad hacia atrás**

Universidad Zaragoza
1542

Índice

Universidad Zaragoza

Manejo de datos de forma segura

Cualquier entrada externa al sistema ha de considerarse no confiable

Diferente origen

- Internet u otra red
- Contenido de un fichero
- De otra aplicación (vía IPC u otros canales de comunicación)

Validación de la entrada

- Ha de considerarse como una aproximación de defensa en profundidad
- Forzar la segregación de datos puede ayudar a prevenir que los datos se conviertan en código
  - **Codificación**: transformar los datos de modo que sólo se puedan interpretar como datos cuando se estén usando
  - *Data binding*: asociando los datos a un tipo de dato se previene que se puedan interpretar como instrucciones

Universidad Zaragoza 1542

Manejo de datos de forma segura

Canonicalización

- **Proceso de conversión de los datos que establece cómo los diferentes formatos equivalentes de los datos se resuelven en una forma estándar, canónica o normal**
- Permite garantizar que las diferentes formas de interpretar los datos no van a lograr evitar cualquier mecanismos de filtro o de seguridad
- **Último paso para poder tomar una decisión con una entrada**

Flujo con una entrada

1. **Aplicar métodos de decodificación a la entrada hasta que la codificación de los datos queda resuelta**
2. **Canonicalizar la entrada**
3. **Validar la entrada** (aceptarla o rechazarla)

Universidad Zaragoza
1542

Manejo de datos de forma segura

Sanitización

- **Asegurar que la entrada cumple los requisitos antes de consumirla**
- Puede implicar eliminar, reemplazar, o codificar partes de la entrada o su totalidad
- **Imprescindible realizarlo en cualquier entrada no confiable**

Universidad Zaragoza

Manejo de datos de forma segura

Canonicalización, validación y sanitización

- Términos **fácilmente confundibles, dado que se suelen usar combinados**
- *Ejemplo*: aplicación que trabaja con fechas
  - La escritura de las fechas depende del país e incluso de la cultura
  - "Sep 7, 2020", "07-09-2020" y "2020/09/07" representan la misma fecha
  - Canonicalización: representar la fecha en formato definido (e.g., "MM/DD/AA") para evitar confusiones
  - Sanitización: reconocer que "<script>alert(1)</script>" no es una fecha y la devolverá vacía
  - Validación: reconocer que "<script>alert(1)</script>" no es una fecha válida y rechazarla

Universidad Zaragoza
1542

Índice

Universidad Zaragoza

Gestión de errores

- **Anticipar todas las formas de interactuar con una aplicación es imposible**
- En algún momento, se producirán errores durante la ejecución
- Capacidad de reaccionar ante errores no anticipados de modo controlado, recuperándose o presentando un mensaje de error
- **Posibles errores conocidos**:
  - Gestión de error específico
  - Código de error propio
- **Errores desconocidos a priori**:
  - Manejadores de errores genéricos
  - Estado de la aplicación desconocido
- **La gestión de errores tiene que integrar también sistema de logging**
  - Nivel de detalle del error diferente para los usuarios que para los administradores
  - Detalles técnicos del error no deberían de notificarse al usuario

Universidad Zaragoza
1542

Fundamentos del Software: introducción a la programación verificada

Programación segura

Buenas prácticas de programación

Roberto Blanco[†] & Ricardo J. Rodríguez[‡]

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

Universidad Zaragoza

[†]Max Plank Institute for Security and Privacy
Bochum, Alemania

[‡]Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza (España)

Septiembre 2020

Universidad de Zaragoza