

[www.segu-info.com.ar](http://www.segu-info.com.ar) agradece a Sebastián Bortnik por la traducción de este documento.

Original en inglés: [http://www.owasp.org/index.php/How\\_to\\_write\\_insecure\\_code](http://www.owasp.org/index.php/How_to_write_insecure_code)

# Cómo escribir código inseguro

Desde [OWASP](http://www.owasp.org)

## Introducción

En el interés de asegurar que habrá un futuro para los hackers, delincuentes, y otros que deseen destruir el futuro digital, este artículo presenta costumbres de los maestros en crear código inseguro. Con un uso un poco creativo de estas ideas, usted podrá asegurar también su futuro financiero. Pero tenga cuidado, usted no querrá que su código parezca desesperadamente inseguro, o su inseguridad podrá descubrirse y arreglarse.

La idea para este artículo proviene de [Cómo escribir código inmantenible](#), de Roedy Green.

De hecho, haciendo su código inmantenible, es un gran primer paso para hacerlo inseguro y encontrará otras grandes ideas en este artículo, especialmente en la sección de camuflaje. Agradecemos también a Steven Christey de MITRE quien contribuyó en algunos temas de inseguridad.

*Nota especial para los lentos en entender **ironías**: ¡Este ensayo es un chiste! Diseñadores y arquitectos están a menudo aburridos con conferencias sobre cómo escribir código seguro. Quizás ésta sea otra manera de comunicar el punto.*

## Principios generales

### Evite las herramientas

Para asegurarse que una aplicación sea para siempre insegura, usted tiene que pensar cómo se identifican y corrigen las vulnerabilidades de seguridad.

Muchos desarrolladores de software creen que las herramientas automatizadas pueden resolver sus problemas de seguridad. Por ello, si quiere asegurarse que existan vulnerabilidades, simplemente hágalas difícil de encontrar para las herramientas automatizadas. Esto es más fácil de lo que parece. Todo lo que tiene que hacer es asegurarse que sus vulnerabilidades no se ajusten con el banco de datos de la herramienta de firmas. Su código puede ser tan complejo como usted quiera, por lo que será bastante fácil evitar que sea encontrado. De hecho, la mayoría de las vulnerabilidades involuntarias, no pueden ser encontradas por estas herramientas.

### Complejidad

### Distribuya mecanismos de seguridad

Los controles de seguridad deben diseñarse para que estén tan distribuidos como sea posible a lo largo del código. Intente no seguir un patrón coherente y dificulte la posibilidad de encontrar aquellos lugares donde los controles se usan. Con esto prácticamente asegurará que la seguridad se ha implementado de forma irregular.

## **Distribuya los agujeros**

Otra gran manera de evitar ser encontrado es asegurarse que sus agujeros de seguridad no se localicen en un único lugar del código. Es muy difícil para un analista guardar todo el código en su cabeza, por lo que esparciendo los agujeros, se previene que cualquiera los encuentre y entienda.

## **Use código dinámico**

La mejor manera de dificultarle la tarea a un analista en seguridad (o herramienta de seguridad en este caso) que lleve a cabo una aplicación para destapar una falla es usar código dinámico. Será casi imposible rastrear el flujo a través del código que está cargado en el runtime. Características como la reflection y classloading son preciosas para esconder vulnerabilidades. Permita tantos "plugins" como sea posible.

## **Mezcle idiomas**

Los idiomas diferentes tienen diferentes reglas de seguridad, de manera que cuantos más idiomas usted incluye, es más difícil aprenderlos todos. Es suficientemente duro para los equipos de desarrollo entender las ramificaciones de seguridad de un idioma, mucho más para tres o cuatro. Puede utilizar las transiciones entre los idiomas también para esconder vulnerabilidades.

## ***Relaciones de confianza***

### **Confíe en chequeos de seguridad hechos en otra parte**

Es redundante hacer verificaciones de seguridad dos veces, entonces si alguien más dice que ha hecho un chequeo, no hay ningún motivo para hacerlo de nuevo. Cuando sea posible, es mejor simplemente asumir que otros están implementando correctamente la seguridad y no perder el tiempo haciéndolo usted. Servicios Web y otras interfaces de servicio son generalmente bastante seguras, así que no se moleste verificando qué se envía o qué devuelven.

## ***Logs***

### **Use sus logs para depurar**

Nadie podrá rastrear ataques en su código si usted llena los logs con depuraciones sin sentido. Puntos extras por hacer sus mensajes de logs indescifrables por cualquiera excepto usted. Más puntos aún, si los mensajes parecen ser relevantes a la seguridad.

### **No use un framework para los logs**

El mejor logging es escribir en stdout (salida estándar, generalmente por pantalla), o quizá a un archivo. Los frameworks hacen los logs demasiado fáciles de encontrar y gestionar para estar seguro que nadie los ha visto.

## ***Cifrado***

### **Construya su propio esquema de cifrado**

Los mecanismos de cifrado normales son demasiado complicados de usar. Mucho más fácil es crear un algoritmo que posea cifrados confidenciales. No es necesario que se moleste con una llave o algo por el estilo, simplemente mezcle todo y nadie lo deducirá.

## ***Autenticación***

### **Construya su propio esquema de la autenticación**

La autenticación es simple, tan sólo use su sentido común e implemente. No se preocupe por cuestiones esotéricas que habrá oído como la predicción de sesión, credenciales hashing, ataques de fuerza bruta, entre otros. Sólo intelectuales de la Agencia de Seguridad podrían deducir al respecto. Y si los usuarios quieren escoger contraseñas débiles, es su propia falta.

## ***Validación de la entrada***

### **La validación es para los bobos**

Su aplicación ya está protegida por un cortafuego (o firewall), para que usted no tenga que preocuparse por los ataques en las entradas del usuario. Los expertos de seguridad que sigan criticándolo, solo se preocupan por mantener su trabajo y no saben nada sobre programar.

### **Permita a diseñadores validar su manera**

No se moleste con una forma estándar de validación, usted está poniendo obstáculos al estilo del desarrollador. Para crear código verdaderamente inseguro, usted debe intentar validar de tantas maneras diferentes como sea posible, y en tantos lugares diferentes como posible.

## ***Control de acceso***

### **Simplemente permita que su esquema de autorización evolucione**

Es muy difícil entender todas las reglas de control de acceso en su aplicación, así que use una metodología de desarrollo just-in-time. De esta manera, usted simplemente puede codificar las nuevas reglas cuando usted las conozca. Si usted termina con centenares o miles de líneas de código de autorización esparcidos a lo largo de su aplicación, usted está en el camino correcto.

## ***Documentación***

### **No documente cómo trabaja la seguridad**

No hay ninguna razón para escribir todos los detalles del diseño de la seguridad. Si alguien quiere conocer cómo funciona, que busque en el código. Después de todo, el código puede cambiar y entonces la documentación sería inútil.

## **Codificando**

### **No use patrones de seguridad**

Asegúrese que no haya ninguna forma estándar de llevar a cabo la validación, el logueo, el manejo de errores, etc... en su proyecto. Es mejor cuando los desarrolladores quedan libres para expresarse y expresar su musa interna en el código. Evite establecer lineamientos de seguridad para la codificación, ya que inhibirán la creatividad.

### **Asegúrese que el proceso de construcción tiene muchos pasos**

Usted quiere maximizar el número de pasos en el proceso de construcción que deben ocurrir en el orden correcto para hacer una creación exitosa. Es mejor si una sola persona conoce cómo funcionan los archivos de configuración y construye la distribución. Si usted tiene los pasos escritos, tendrá muchas notas distribuidas en un grupo de archivos y howto's en muchas ubicaciones.

## **Testeo**

### **Pruebe con un sólo browser**

Todo lo que usted necesita para probar una aplicación web es un browser. De esa forma, usted se asegurará que su código trabajará perfectamente para todos los usuarios legítimos. ¿Y qué le importa si no funciona para los hackers, asaltantes, y delincuentes? Usar una elegante herramienta de prueba en seguridad como [WebScarab](#) es una pérdida de su valioso tiempo.

### **No use herramientas de análisis estáticas**

Estas herramientas no son perfectas, así que solo olvídense de ellas. Es probable que las herramientas de análisis estáticas creen mensajes de advertencia molestos que reducirán la velocidad de su tiempo de desarrollo. Además, es un sufrimiento aprenderlas, setearlas, y usarlas.

### **Cree código de prueba en cada módulo**

Usted debe construir código de prueba en todas las partes de la aplicación. De esta manera, será muy difícil de extraerlo de la aplicación antes de su utilización.

### **Las advertencias de seguridad son todas falsas alarmas**

Todos sabemos que las advertencias de seguridad son todas falsas alarmas, por lo que puede simplemente ignorarlas. Por qué usted debe perder su tiempo localizando posibles problemas cuando no es su dinero el que está en juego. Si una herramienta no produce resultados perfectos, entonces simplemente olvídense de ella.

## **Bibliotecas**

### **Use muchas bibliotecas precompiladas**

Las bibliotecas son estupendas porque usted no tiene que preocuparse si el código tiene algún problema de seguridad. Usted puede confiar en cualquier

biblioteca que descargue de Internet e incluya en su aplicación. No se preocupe por verificar el código fuente, probablemente no está disponible y será demasiado problema recompilarlo. Y, de todas formas, tomaría demasiado tiempo verificar todo ese código fuente.

Original en inglés: [http://www.owasp.org/index.php/How\\_to\\_write\\_insecure\\_code](http://www.owasp.org/index.php/How_to_write_insecure_code)