



# Herramientas para análisis estático de seguridad: estado del arte

*III OWASP Spain Charter Meeting*



## Resumen:

- En esta ponencia se examina la situación actual de la clase de herramientas de análisis estático de seguridad, que sin ejecutar el código del software, tratan de identificar las vulnerabilidades explotables en las aplicaciones y servicios web.
- Se determina el estado del arte de las herramientas académicas y de código abierto, la oferta de OWASP en este dominio, los límites de esta tecnología, y se proponen algunas ideas para mejorar la difusión y la cobertura desde OWASP.

## Ponente:

Luis Rodríguez (lrodriguez@als-es.com)

Responsable laboratorio SW de ALS. CISSP.



- ¿Por qué las aplicaciones web son como anchas autopistas para los malos?
- Análisis estático de seguridad del SW: cómo funciona
- Ventajas (e inconvenientes) del análisis estático
- Técnicas aplicables para el análisis estático de seguridad
- Estado actual de las herramientas de código abierto. Limitaciones.
- Integración, en el ciclo de vida SW, de las revisiones de seguridad en el código.
- Conclusiones. ¿Qué más puede hacerse desde OWASP en este ámbito?



## Herramientas para análisis estático de seguridad: estado del arte

¿Por qué las aplicaciones son como  
anchas autopistas para los malos?



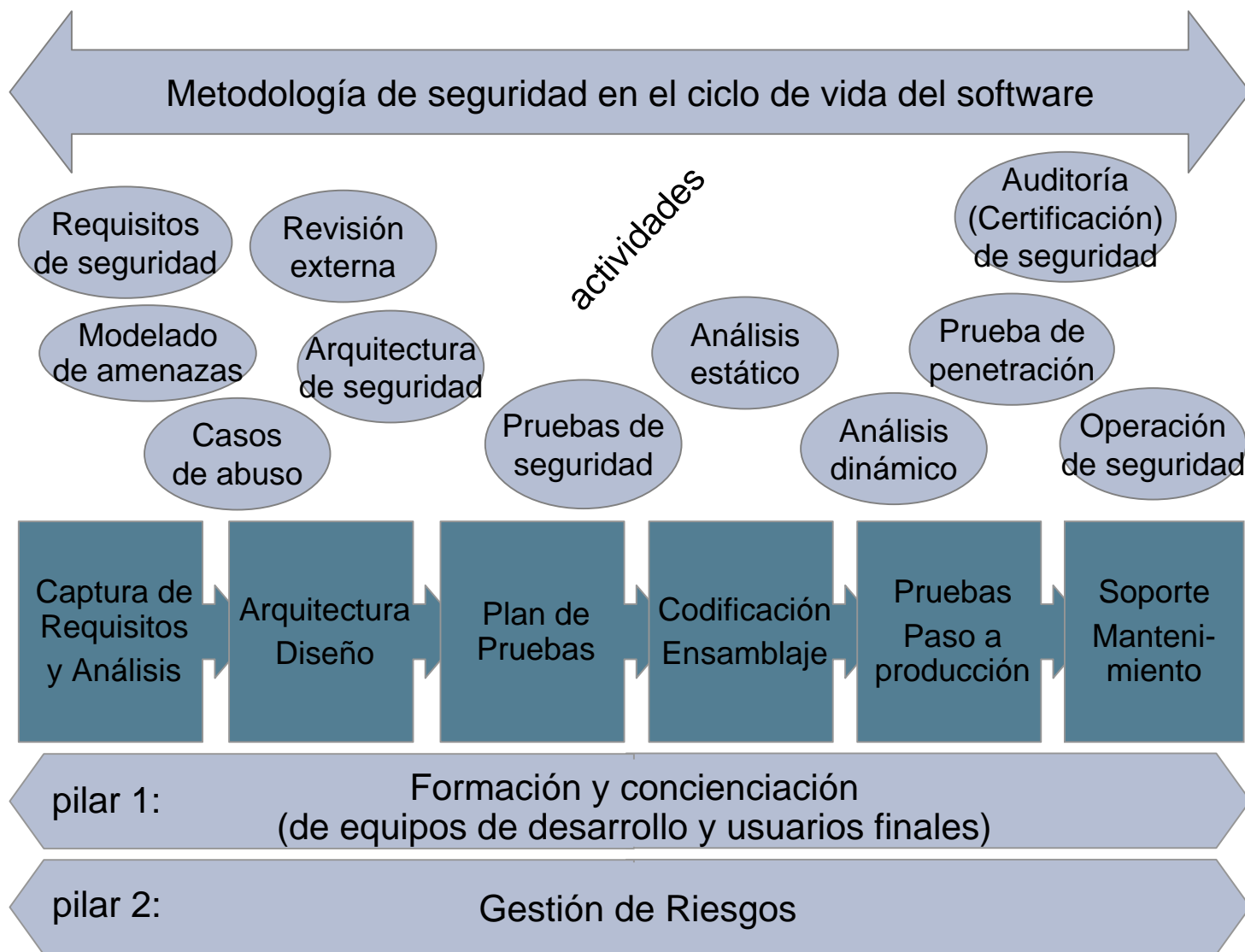
- “*I'm glad software developers don't build cars*”  
-- Eoin Keary, líder del proyecto OWASP Code Review Guide
- Modelo económico del software que no incorpora el riesgo o la “seguridad” del producto
  - Falta de recursos (tiempo, herramientas, personal...)
  - Falta de mercado (de vulnerabilidades, de seguros)
- Falta de conocimientos *elementales* de seguridad en los equipos de desarrollo (y pruebas) de software
  - Principios de diseño de seguridad
  - Desconocimiento de cómo abusan los malos del protocolo HTTP
  - ¿Validación de entradas? Ein?
  - Cómo operan las inyecciones, XSS/CSRF, *Path traversal*...
- Modelos de ciclo de vida / proceso de desarrollo SW que ignoran la seguridad



- A1. Entrada no validada
- A2. Control de accesos roto
- A3. Autenticación/Gestión de sesiones rotos
- A4. XSS, Cross Site Scripting (y su variante CSRF)
- A5. BO, desbordamiento de búfer
- A6. Inyección (SQL, comandos...)
- A7. Tratamiento de errores incorrecto
- A8. Almacenamiento inseguro / mal uso del criptografía
- A9. Denegación de servicio
- A10. Gestión de configuración insegura



- “Programación defensiva es lo que necesito”
  - = Codificar con la idea de que los errores son inevitables y, antes o después, algo irá mal y provocará condiciones inesperadas. Hay que codificar el software “de forma que pueda afrontar pequeños desastres”
  - No es una idea dirigida a producir software seguro, sino código que facilite la detección de defectos no debidos al uso “inesperado” (malicioso) del software.
- “Funciones de seguridad = funciones seguras”
  - Para que un programa sea seguro, no basta con las funciones de seguridad implantadas. La seguridad de software es más que añadir funciones de seguridad, como la autenticación o el control de accesos.
- “El proceso de pruebas y QA incluye la seguridad, ¿no?”
  - Debería, pero no... Pruebas / QA != pruebas de seguridad
  - El software **fiable** hace lo que se supone que debe hacer; el software **seguro**, hace *siempre* lo que se supone que debe hacer, y *nada más*.



(\*) Adaptado de *Software Security: Building Security In*. Gary McGraw. Addison Wesley, 2006





**tipos de  
ataque**

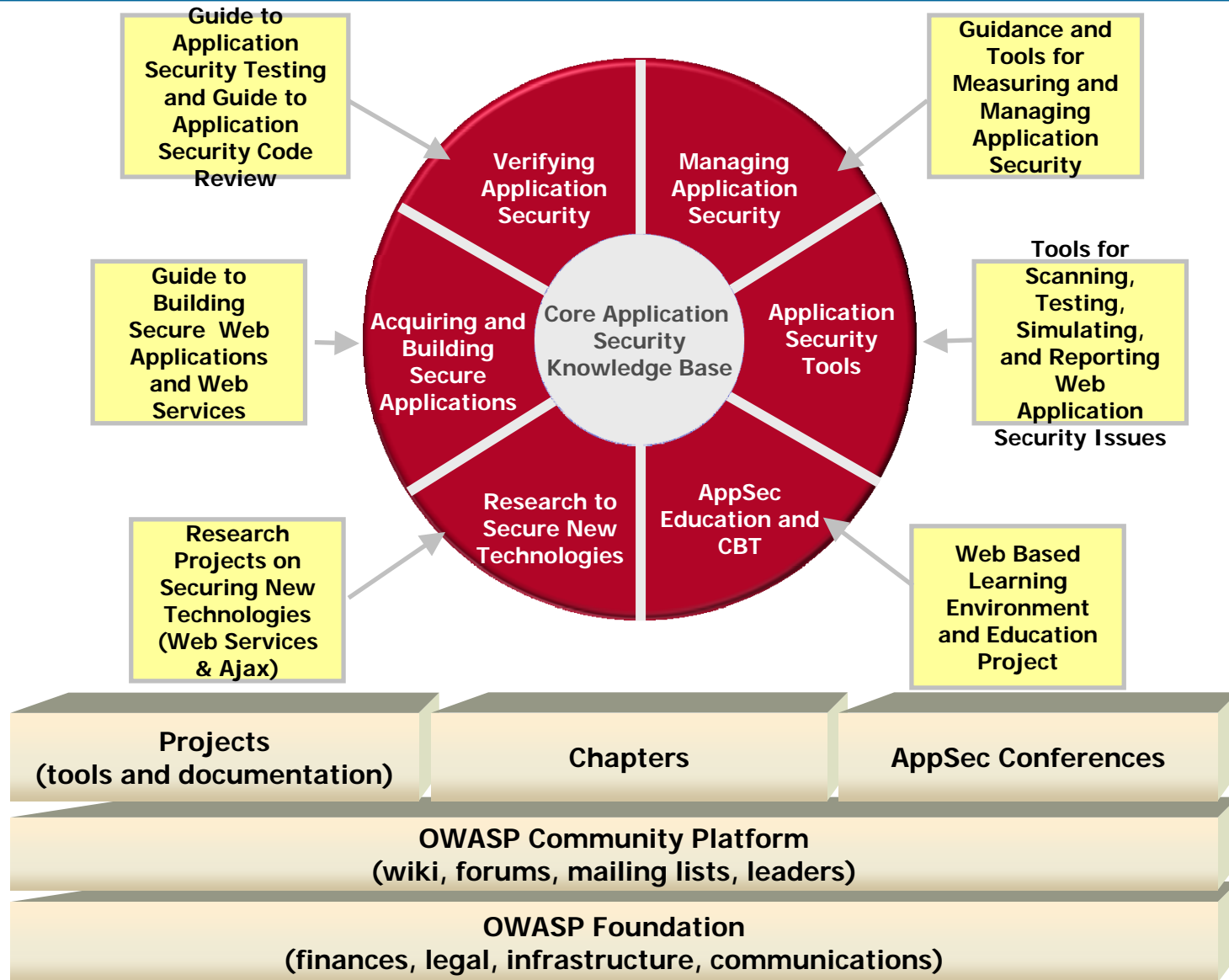
**Controles de  
infraestructura**

**controles  
genéricos**



**controles  
específicos**

tipos de ataque	Controles de infraestructura	controles genéricos	controles específicos
inyección SQL			SQL parametrizado
inyección de código en cliente web (XSS)		casos de abuso casos de prueba de seguridad	no mezclar código/datos 'desactivar' la salida
ataques a la autenticación secuestro de sesiones		validación (positiva) de entradas filtrado de salidas	'captcha' (reto anti-robot) autenticación robusta tickets de autenticación
captura no autorizada de información / configuración		análisis estático + revisión de código mínimo privilegio	cifrado gestión de config.
división HTTP navegación por el sistema de ficheros		criptografía (bien implantada)	canonicalización no derivar desde entradas
desbordamiento de buffer denegación de servicio		auditoría de seguridad prueba de penetración	APIs seguros protectores de pila limitadores de tráfico



(\*) Tomado de "Finding and Fighting the Causes of Insecure Applications", Jeff Williams



# Herramientas para análisis estático de seguridad: estado del arte

## Análisis estático de seguridad del SW

Cómo funciona,

Ventajas e inconvenientes

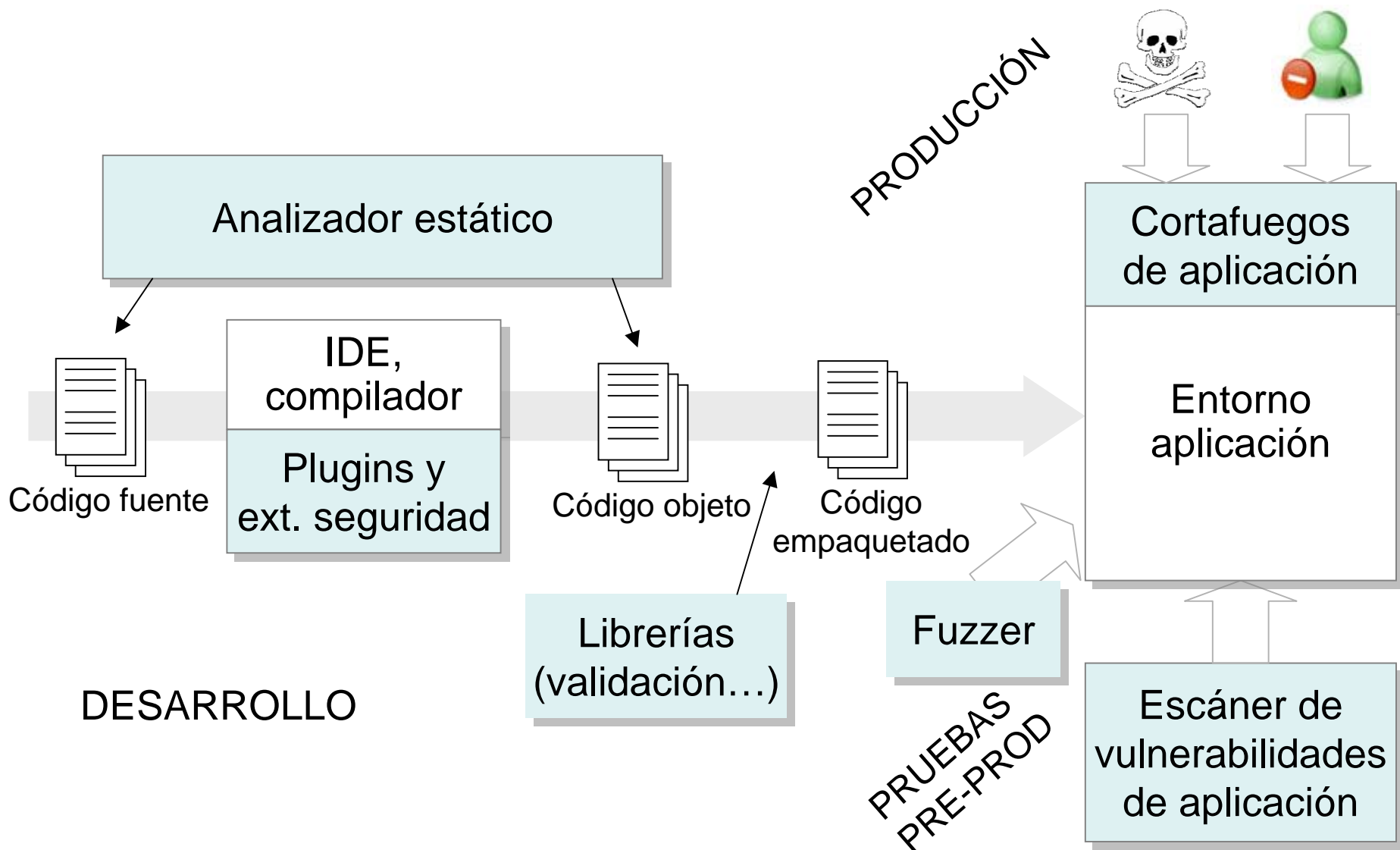
Técnicas aplicables

Herramientas disponibles. Limitaciones.



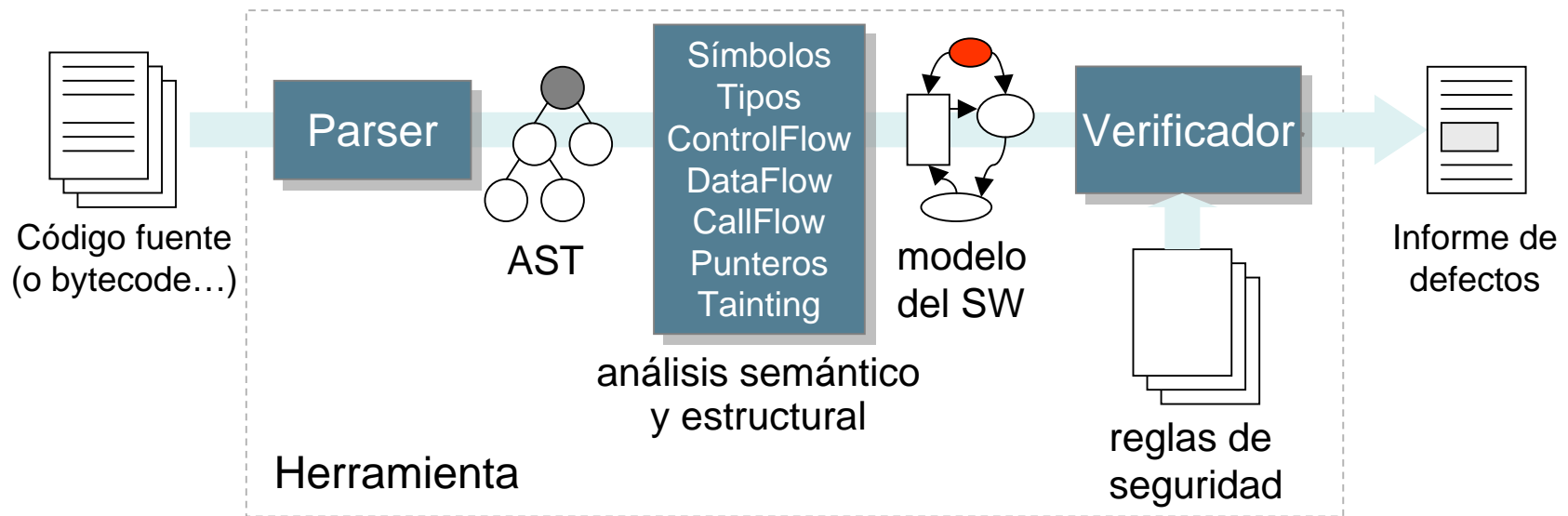
	manual ('creativo')	automático ('barato')
dinámico	<b>Prueba de penetración</b> Aplicación web desplegada. Conocimiento 'nulo'	<b>Escaneo</b> Aplicación web desplegada. Batería predefinida de pruebas. Variantes: fuzzers, scanners, protocol proxies, etc.
estático	<b>Revisión de código</b> Código y configuración. Uso de docs (specs y diseño) Orientado a defectos de diseño y codificación	<b>Análisis estático</b> Código y configuración. Reglas predefinidas. Orientado a defectos de codificación

- Recuérdese: Cada una tiene sus aplicaciones, fortalezas, debilidades y puntos ciegos. Es absurdo preguntarse si es mejor un martillo o un serrucho: para construir se necesitan distintas herramientas...
- Toda aplicación es diferente: úsese la combinación de técnicas más efectiva, caso por caso.





- Estático = se analiza sin ejecutar el software
- Ventajas:
  - **Consistencia.** La herramienta ve lo que ve, sin ideas preconcebidas (que normalmente tienen los desarrolladores o revisores).
  - Apuntan a la **causa raíz**, no a los síntomas. Una prueba de penetración puede establecer que hay un problema, pero no su causa final ni cómo corregirlo.
  - **Detección precoz.** La aplicación no tiene que estar integrada ni necesita ejecutarse.
  - Su ejecución es **barata**. Un sistema puede re-analizarse cuando se aplican cambios, o cuando se descubre una nueva vulnerabilidad de aplicación.
- Inconvenientes:
  - **Falsos positivos.** Impacto (coste) crece al tener que evaluar cada positivo.
  - **Falsos negativos.** Suelen ser incapaces de detectar vulnerabilidades de seguridad achacables al diseño, o específicas del contexto propio de la aplicación (se centran en vulnerabilidades genéricas, de codificación).
  - ¿Qué es mejor? En seguridad, sin duda, baja tasa de falsos negativos sin una tasa desproporcionada de falsos positivos.



- Las etapas iniciales son similares a las que sigue un compilador.
- El análisis semántico permite pasar a una representación interna del software en la que están representadas las nociones básicas para determinar defectos de seguridad: flujo de datos y llamadas, tipos y tamaños de las variables, entradas y recursos alcanzables, y qué entradas están bajo control (*taint*) del usuario. Este modelo incluye el entorno (librerías, funciones de sistema, etc.)
- La “inteligencia” está codificada en reglas que un verificador aplica sobre el modelo interno para determinar posibles defectos.



- Análisis semántico: Funciones anotadas con precondiciones y postcondiciones que resumen el comportamiento de la función.

```
void FillString(  
    __out_ecount(cchBuf) TCHAR* buf,  
    size_t cchBuf,  
    char ch) { ... }
```

```
TCHAR *b = (TCHAR*)malloc(200*sizeof(TCHAR));  
FillString(b,210,'x');
```

- Análisis estructural: Puede usar lenguajes de consulta de código, la mayoría académicos (como PQL):

```
query simpleSQLInjection()  
uses  
    object HttpServletRequest r;  
    object Connection c;  
    object String p;  
matches { p = r.getParameter(_); }  
replaces c.execute(p)  
with Util.CheckedSQL(c, p);
```





```
If( fgets(buf, sizeof(buf), stdin) == buf) {  
    strcpy(cmd, buf);  
    system(cmd);  
}
```

- 1) Fuente: fgets
  - 2) DFA conecta fuente fgets con buf
  - 3) Se propaga a cmd
  - 4) DFA propaga cmd a la llamada a system
  - 5) Sumidero alcanzado: system
- Se lanza problema: “*inyección de comando*”

- El objetivo es determinar si una entrada controlada por el usuario alcanza un recurso sin verificar
  - Inyección SQL, de comandos, *cross-site scripting*
  - Además de fuentes y sumideros, hay que considerar reglas de propagación (3) y reglas de filtrado o limpieza
  - El grado en que una entrada puede estar controlada por el atacante debe manejarse durante el análisis











- Tipos de análisis (AFD, semántico, estructural)
- Categorías de vulnerabilidades que pueden detectarse, y esquema de nombres que siguen (e.g. CWE)
- Precisión (falsos positivos y negativos)
- Capacidad de configuración (esp. del entorno: librerías de terceros), por ejemplo mediante reglas, filtros, marcas sobre items del código...
- Calidad de las recomendaciones correctivas
- Posibilidad de trabajar en modo 'híbrido' (análisis estático + confirmación en modo 'caja negra')
- Capacidades de integración (IDE, detectores dinámicos...)
- Posibilidades de detección de defectos de diseño
- Explicación razonable ("causa raíz"), y nivel de confianza
- NIST SAMATE (*Software Assurance Metrics And Tool Evaluation*)





Nombre	Licencia	Descripción
BOON	académico	Verificador de modelos orientado a detectar buffer overflow en C.
BugFind	open source	Defector de defectos (y escáner de seguridad muy básico) para Java.
Bugscam	open source	Analizador sobre IDA. Busca llamadas peligrosas en código binario ejecutable.
Compuware DevPartner SecurityChecker	comercial	Escáner de seguridad de código para .NET (C# y VB.NET)
CodeScan	comercial	Análisis estático de seguridad para ASP y PHP
Coverity Prevent	comercial	Detector de defectos y escáner de seguridad para C/C++.
Cqual	académico	Analizador de flujo de datos para C, basado en análisis de tipos.
Flawfinder	open source	Escáner de seguridad para lenguaje C.
Fortify SCA	comercial	Escáner de seguridad para diversos lenguajes.
Grammtech CodeSonar	comercial	Vulnerabilidades y otros defectos en C/C++ y ADA.
HP DevInspect	comercial	C#, Java, HTML/XML, JavaScript, SOAP
ITS4	freeware	Busca llamadas a funciones potencialmente peligrosas en código C.
Klockwork Insight	comercial	Escáner de seguridad para lenguajes C/C++ y Java.
McCabe IQ	comercial	Analizador de flujo de control general y cobertura. Soporta C, C++, C#, Java, Fortran, VB, COBOL, y otros.
Microsoft FxCop, prefast, CAT.NET/XSSDetect	freeware	Analizadores de código para lenguajes .NET, y C/C++ (flag /analyze)
MOPS	académico	Verifica vulnerabilidades en secuencias de llamadas a funciones C.
OunceLabs Ounce5	comercial	Escáner de seguridad para C/C++, Java/JSP, .NET (C#/VB.NET) y VB6/ASP
OWASP LAPSE	open source	Escáner simple de seguridad para Java (plugin Eclipse).
Parasoft *Test	comercial	Análisis estático general, con algunas reglas de seguridad, para Java, C/C++, .NET y HTML
Pixy	open source	Inyección SQL y XSS (para PHP)
PHP-SAT	open source	Analizador para PHP
Pscan	open source	Busca llamadas a funciones potencialmente peligrosas en código C.
RATS	open source	Busca llamadas a funciones potencialmente peligrosas en código C.
smatch	open source	Defector de defectos (y escáner de seguridad) para C/C++.
splint	open source	Busca vulnerabilidades potenciales y malas prácticas de codificación en el código C.



- Herramientas OWASP
  - LAPSE (tool) Madurez 
  - OWASP Orizon (tool) 
- Documentación OWASP
  - (Building) Guide 
  - Top Ten Project 
  - Testing Guide 
  - CLASP 
  - Code Review Project 
  - App Security Metrics Project 
- Conclusiones:
  - Herramientas muy por debajo de sus equivalentes comerciales
  - Mayor abanico de posibilidades en análisis “dinámico” (fuzzers, escáners dinámicos) que en estático



- Plugin Eclipse (Java)
- Muy básico
- Basado en tainting
- Detecta vulnerabilidades de tipo inyección

The screenshot displays the Eclipse IDE interface with the 'Resource - ChallengeScreen.java - Eclipse Platform' window. The 'Provenance Tracker' view is active, showing a hierarchical tree of code elements. Below it, the 'SQL Injection Sinks' view is open, displaying a table of suspicious calls.

Suspicious call	Function	Category	Project	File	Line
<input type="checkbox"/> da.getStatement().execute(sb.toString())	java.sql.Statement.execute(String)	SQL	BlogWelder	CreateTables.java	239
<input type="checkbox"/> da.getStatement().execute(sb.toString())	java.sql.Statement.execute(String)	SQL	BlogWelder	CreateTables.java	250
<input type="checkbox"/> statement3.executeQuery(query)	java.sql.Statement.executeQuery(String)	SQL	WebGoat	ChallengeScreen.java	207
<input type="checkbox"/> statement.executeQuery(query)	java.sql.Statement.executeQuery(String)	SQL	WebGoat	DatabaseXss.java	68
<input type="checkbox"/> statement.executeQuery(query)	java.sql.Statement.executeQuery(String)	SQL	WebGoat	DatabaseXss.java	196





Project: **Tomcat 5.5.20**
View: All | Unreviewed | Defects
Projects | Logout

Navigator

Group by **category**

- Bad casts of object references [3]
- Command Injection [56]
  - CGIServlet.java: 1690 (multiple issues)
- Dubious method invocation [8]
- Infinite Loop [1]
- Masked Field [8]
- Null pointer dereference [49]
- Password Management: Hardcoded Password [1]

```

1681 }
1682
1683     StringBuffer command = new StringBuffer(cgiExecutable);
1684     command.append(" ");
1685     command.append(cmdAndArgs.toString());
1686     cmdAndArgs = command;
1687
1688     try {
1689         rt = Runtime.getRuntime();
1690         proc = rt.exec(cmdAndArgs.toString(), hashToStringArray(env), wd
1691
1692         String sContentLength = (String) env.get("CONTENT_LENGTH");
1693
1694         if(!"".equals(sContentLength)) {
1695             commandStdIn = new BufferedOutputStream(proc.getOutputStream());

```

- CGIServlet.java:1105 - org.apache.catalina.connector.Request.getRequest
- CGIServlet.java:1105 - org.apache.catalina.servlets.CGIServlet\$CGIER
- CGIServlet.java:1105 - java.util.Hashtable.put return statement
- CGIServlet.java:1150 - this.env assignment statement
- CGIServlet.java:766 - org.apache.catalina.servlets.CGIServlet\$CGIER
- CGIServlet.java:584 - org.apache.catalina.servlets.CGIServlet\$CGIER
- CGIServlet.java:588 - org.apache.catalina.servlets.CGIServlet\$CGIER
- CGIServlet.java:587 - org.apache.catalina.servlets.CGIServlet\$CGIRU
- CGIServlet.java:597 - org.apache.catalina.servlets.CGIServlet\$CGIRU
- CGIServlet.java:1672 - java.util.ArrayList.get assignment statement
- CGIServlet.java:1672 - param assignment statement
- CGIServlet.java:1674 - java.lang.StringBuffer.append assignment sta
- CGIServlet.java:1685 - java.lang.StringBuffer.toString assignment sta
- CGIServlet.java:1685 - java.lang.StringBuffer.append assignment sta
- CGIServlet.java:1686 - cmdAndArgs assignment statement
- CGIServlet.java:1690 - java.lang.StringBuffer.toString assignment sta
- CGIServlet.java:1690 - java.lang.Runtime.exec assignment statement

**Abstract:** Executing commands from an untrusted source or in an untrusted environment ca

**Explanation:** Command injection vulnerabilities take two forms: An attacker can char  
the possibility that an attacker may be able to control the command that is execute  
The data is used as or as part of a string representing a command that is execute  
have. Example 1: The following code from a system utility uses  
code in Example 1 allows an attacker to execute arbitrary commands with the elev  
environment, if an attacker can control the value of the system property <code style=  
wrapper around the <code style='font-family: monospace'>rman</code> utility and  
restricted, the application runs the backup as a privileged user. <br><br><div styl  
<code style='font-family: monospace'>String&nbsp;btype&nbsp;=&nbsp;request.getParameter  
<code style='font-family: monospace'>backuptype</code> parameter read from the  
<code style='font-family: monospace'>Runtime.exec()</code>. Once the shell is invoked, it will happily exec  
the privileges necessary to interact with the database, which means whatever com  
environments is to run a <code style='font-family: monospace'>make</code> com  
specify an absolute path for make and fails to clean its environment prior to execut



Fortify Audit Workbench - qwik-smtpd - [/Applications/FortifySoftware/SCAS-EE4.0.0/Tutorial/c/audits/qwik-smtpd/qwik-smtpd.fpr] \*

Issues - Broad (Fortify Default)

Hot (14) Warning (45) Info (11) **All (70)**

Group by: Category

- Buffer Overflow (Data Flow) - [30 / 30]
- Format String (Data Flow) - [8 / 8]
  - qwik-smtpd.c:211 (Format String) (multiple issue)
    - from getc(return) - qwik-smtpd.c:506
    - from getline(0) - qwik-smtpd.c:182
    - from getc(return) - qwik-smtpd.c:584
    - from fgets(0) - qwik-smtpd.c:363
  - qwik-smtpd.c:434 (Format String) (multiple issue)
  - qwik-smtpd.c:436 (Format String)
- Setting Manipulation (Data Flow) - [8 / 8]
- Poor Style: Redundant Initialization (Structural) - [3 / 3]
- Poor Style: Variable Never Used (Structural) - [4 / 4]
- Missing Check against Null (Control Flow) - [2 / 2]
- Unreleased Resource (Control Flow) - [2 / 2]
- Code Correctness: Function Returns Stack Address (Control Flow) - [2 / 2]
- Uninitialized Variable (Control Flow) - [2 / 2]

qwik-smtpd.c

```

204 {
205 // unnecessary check for max_rec.
206 if(clientRcpts < max_recipients)
207 {
208     for(x = 0; x < max_recipients; x++)
209     {
210         if(clientRcptTo[x] == NULL) break;
211         fprintf(fpout,clientRcptTo[x]);
212         (void) fflush(fpout);
213         fprintf(fpout,"\n");
214         (void) fflush(fpout);
215         // TODO: add logging mechanism here
216         //syslog(LOG_MAIL,"From: %s To: %s ClientIP: %s %s",
217         //      clientMailFrom, clientRcptTo[x], clientIP, messageID);
218         //strcpy(logline,"");
219         //sprintf(logline, "%s qwik-smtpd[%s] %s %s From: %s To: %s", timebu
220         //fprintf(Log,logline);
221         //(void) fflush(Log);
222     }
223     fclose(fpout);
224 }
225 else
226 {

```

**código**

Summary Details

Location: qwik-smtpd.c:211

Analysis: Exploitable Impact: Medium

Status: Reviewed List: Hot

Comments: Unable to locate validation against formatting metacharacters dangerous if the filesystem is untrusted

**Format String (Input Validation and Representation, Data Flow)**

Allowing an attacker to control a function's format string can result in a buffer overflow.

**explicación**

File Bug... Suppress issue View More Details

Analysis Trace

- getc(return) - qwik-smtpd.c:506
  - [assignment to c] - qwik-smtpd.c:506
  - [assignment to line] - qwik-smtpd.c:522
- getline(0) - qwik-smtpd.c:182
- parseInput(0 --> 3) - qwik-smtpd.c:256
- push(0 --> {clientRcptTo}) - qwik-smtpd.c:378
- fprintf(1) - qwik-smtpd.c:211

**Traza (de flujo de datos)**



Microsoft FxCop - My FxCop Project [C:\Users\Jason\Documents\Visual Studio 2005\Projects\TestProject\TestProject\FxCop\\*]

File Edit Project Tools Windows Help

Analyze

Targets Rules

My FxCop Project

- Design Rules
- Globalization Rules
  - Avoid duplicate accelerators
  - Do not hardcode locale specific strings
  - Normalize strings to uppercase
  - Set locale for data types
  - Specify CultureInfo
  - Specify IFormatProvider
  - Specify marshaling for P/Invoke string arguments
  - Specify MessageBoxOptions
  - Specify StringComparison
  - Use ordinal StringComparison
- Interoperability Rules
- Mobility Rules
- Naming Rules
- Performance Rules
- Portability Rules

Active Excluded In Project Excluded In Source Absent

Level	Fix Category	Certainty	Rule	Item
!	Non Bre...	95%	Assemblies should have valid strong n...	testproject.exe
!	Non Bre...	95%	Mark assemblies with NeutralResourc...	testpro
!	Non Bre...	95%	Mark assemblies with CLSCompliantAt...	testpro
!	Breaking	90%	Nested types should not be visible	TestPr
!	Breaking	95%	Declare event handlers correctly	TestPr
!	Breaking	90%	Do not declare visible instance fields	TestPr
!	Non Bre...	95%	Specify MessageBoxOptions	TestPr
!	Breaking	95%	Do not raise reserved exception types	TestPr
!	Non Bre...	95%	Remove unused locals	TestPr
!	Non Bre...	75%	Avoid uncalled private code	TestPr
!	Non Bre...	95%	Mark members as static	TestPr
!	Depends ...	95%	Specify IFormatProvider	TestPr
!	Depends ...	95%	Specify IFormatProvider	TestPr

1 message(s) selected

Error, Certainty 95, for DoNotRaiseReservedExceptionTypes

```
{
  Target      : #DisplayError() (IntrospectionTargetMember)
  Location    : file:///C:/Users/Jason/Documents/Visual%2520Studio%25202005/Projects/TestProject
  Resolution  : "'Form1.DisplayError()' creates an exception of
                type 'Exception', an exception type that is not sufficiently
                specific and should never be raised by user code. If
                this exception instance might be thrown, use a different
                exception type."
  Help       : http://msdn2.microsoft.com/ms182338(VS.90).aspx (String)
  Category   : Microsoft.Usage (String)
  CheckId    : CA2201 (String)
  RuleFile   : Usage Rules (String)
  Info       : "User code should not create and raise exceptions of
                certain types that are reserved by the runtime or which
                are of a too general exception type. Exception types
```

## Security Rules

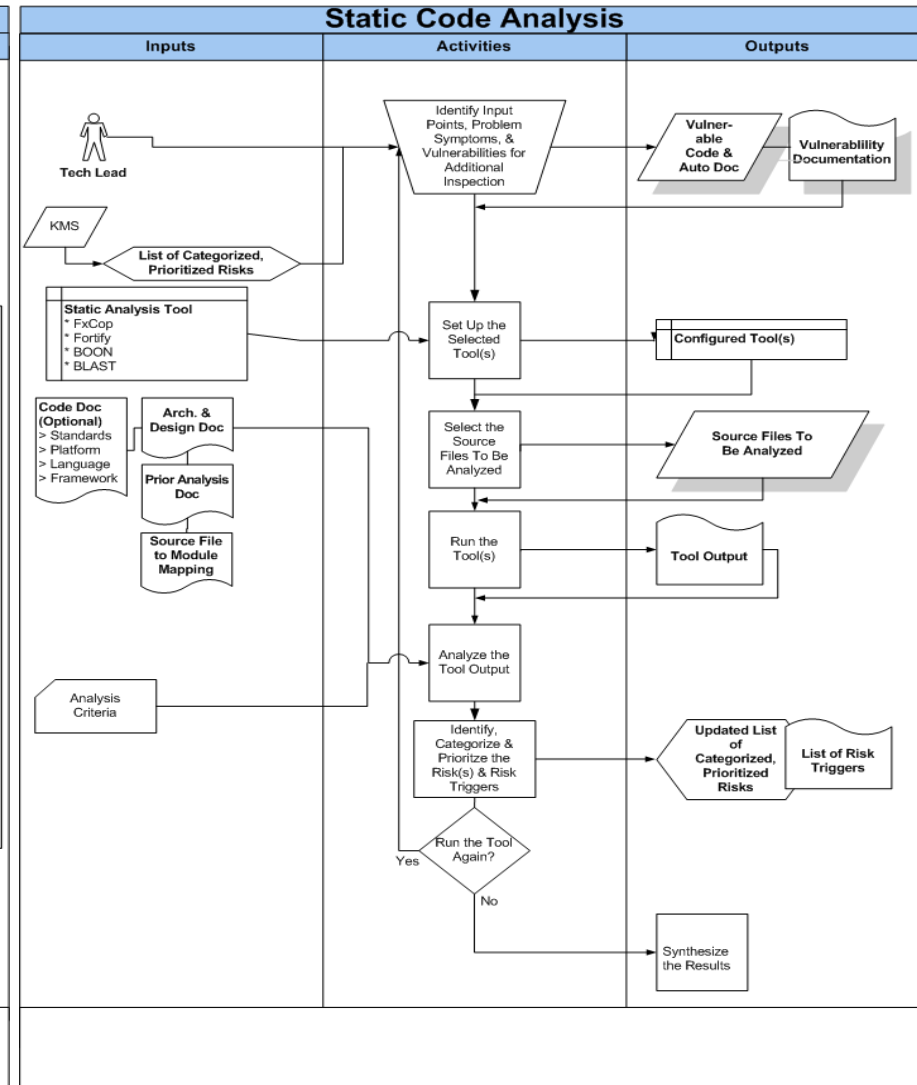
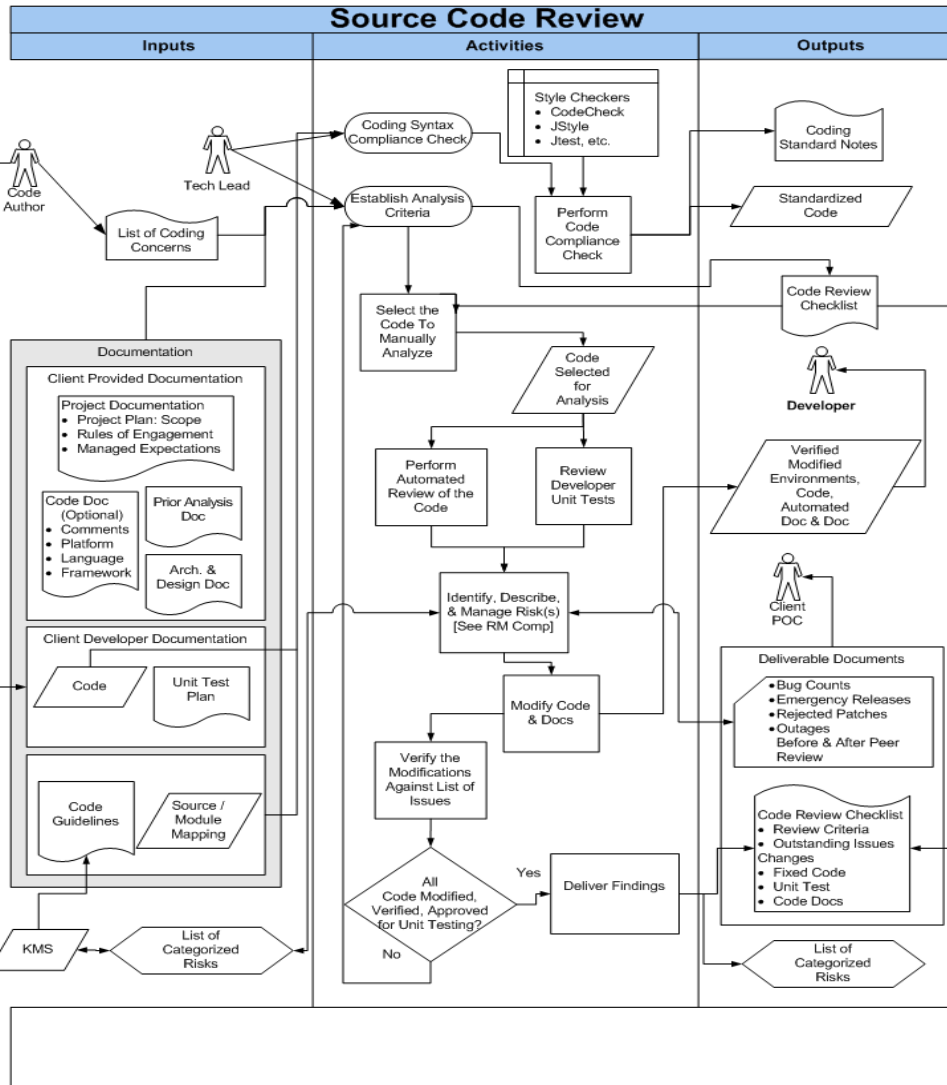
- [Aptca methods should only call aptca methods](#)
- [Aptca types should only extend aptca base types](#)
- [Array fields should not be read only](#)
- [Call GC.KeepAlive when using native resources](#)
- [Catch non-CLSCompliant exceptions in general handlers](#)
- [Do not declare read only mutable reference types](#)
- [Do not indirectly expose methods with link demands](#)
- [Method security should be a superset of type](#)
- [Override link demands should be identical to base](#)
- [Pointers should not be visible](#)
- [Review declarative security on value types](#)
- [Review deny and permit only usage](#)
- [Review imperative security](#)
- [Review sql queries for security vulnerabilities](#)
- [Review suppress unmanaged code security usage](#)
- [Review visible event handlers](#)
- [Seal methods that satisfy private interfaces](#)
- [Secure asserts](#)
- [Secure GetObjectData overrides](#)
- [Secure late-binding methods](#)
- [Secure serialization constructors](#)
- [Secured types should not expose fields](#)
- [Specify marshaling for pinvoke string arguments](#)
- [Static constructors should be private](#)
- [Type link demands require inheritance demands](#)
- [Wrap vulnerable finally clauses in outer try](#)





# Herramientas para análisis estático de seguridad: estado del arte

Integración de las revisiones de  
seguridad de código en el ciclo de vida



Fuente: <https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/code/214.html>



## 1. Establecer objetivos

- a. Priorizar qué partes a analizar
- b. Entender el software (a alto nivel)
- c. Comprender riesgos

## 2. Lanzar herramientas

- a. Introducir reglas específicas
- b. Compilar el código; lanzar herramienta

## 3. Revisar resultados

- a. Revisión manual a partir de problemas potenciales
- b. Si falso positivo: Reconfigurar; Si falso negativo: Añadir regla
- c. Registro de problemas (informe formal, gestor de defectos...)

## 4. Introducir correcciones

- a. Revisar (manual/automáticamente) cambios
- b. Actualizar “Buenas prácticas”, objetivos alcanzados, y reglas

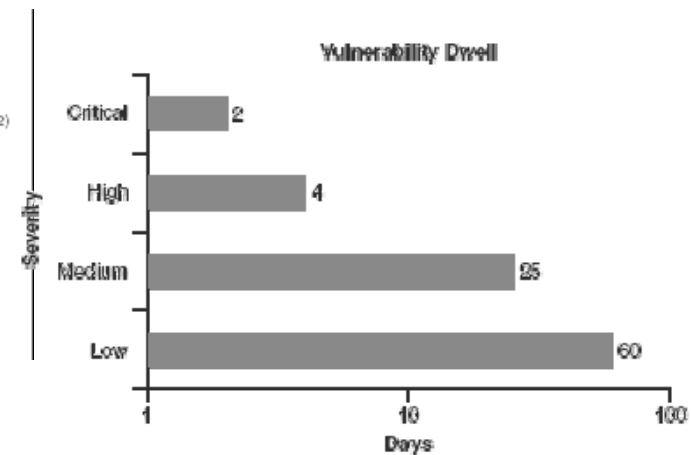
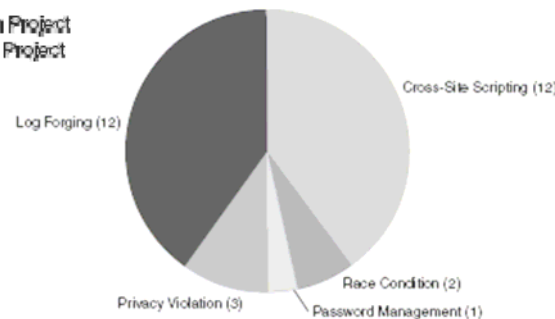
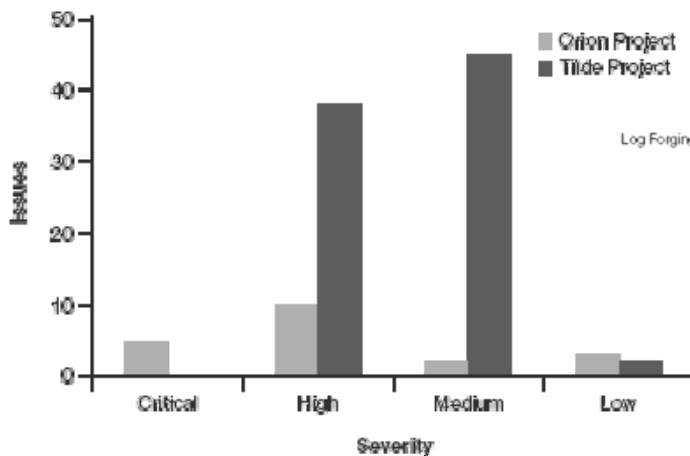




- ¿Quién ejecuta la herramienta?
  - Seguridad del software, desarrolladores, o mejor ambos.
- ¿Cuándo?
  - Desde el IDE del programador; en subida de código al control de versiones; en script de build; al librerar una versión; en procesos formales de auditoría de código.
  - No olvidar que la MAYOR CARGA (90%) de trabajo se consume al analizar resultados e introducir correcciones.
- ¿Cómo tratar los resultados?
  - “Si hay riesgo, se bloquea la liberación de la versión”
  - “Una autoridad central filtra y decide”
- Usar aproximación incremental al adoptar técnicas:
  - Recordar las tácticas empleadas con los IDS/IPS



- Densidad de defectos =  $\# \text{ vuln. potenciales} / \text{KLOC}$ 
  - Cuidado! La densidad de defectos depende mucho de factores no cuantificados, como el estilo de programación
  - Útil para establecer prioridades, por comparación a nivel de sistemas / módulos / en el tiempo
- Es importante la evolución en el tiempo, y el tiempo medio de corrección
- Las métricas deberían desglosarse por severidad del defecto





- Las 6 excusas
  - “No creo que sea explotable: demuéstremelo...”
  - “Confío en los administradores de redes / sistemas”
  - “Tienes que estar autenticado para acceder a esa página”
  - “A nadie se le ocurriría hacer eso”
  - “Esa función nunca ha dado problemas (en los tests, nodo X...)”
  - “Ese código no estaba previsto que acabase en producción; tenlo en cuenta...”
- Los gestores deben transmitir este modo de proceder:
  - Si hay un defecto de seguridad, incluso potencial, debe resolverse. Punto.
  - El equipo de revisión *no tiene que demostrar nada a desarrollo*. El equipo de desarrollo, si lo desea, puede argumentar las razones por las que creen que el código es seguro, sin usar ninguna de las “6 excusas” arriba enumeradas.
  - Siempre es mejor cubrir con código defensivo una vulnerabilidad potencial no explotable, que dejar sin tratar una vulnerabilidad explotable.

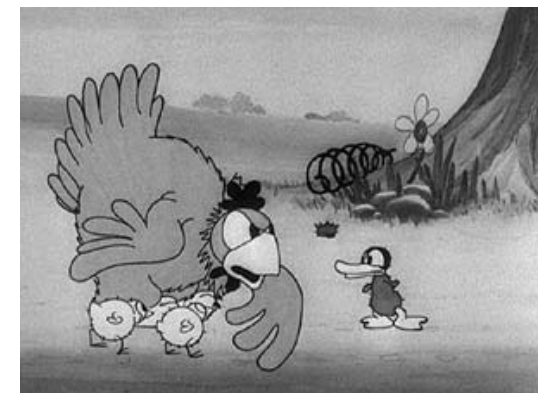


# Herramientas para análisis estático de seguridad: estado del arte

Conclusiones. Áreas de futuro trabajo



- Los resultados emitidos por herramientas de análisis estático necesitan de evaluación humana
- Es más efectivo (aunque difícil) el cambio activo de los hábitos de desarrollo SW, que seguir una aproximación puramente reactiva (ciclos auditoría – corrección )
- Tecnologías reactivas (cortafuegos de red o de aplicación) pueden aliviar el problema, no lo corrigen.
  - El problema es el software malo. Y surge desde la mesa de diseño.
  - Las herramientas de análisis estático de la seguridad en código pueden (deben?) formar parte del proceso de desarrollo.
- ¿Qué puede hacerse dentro de OWASP?
  - Help! Aportar allí donde más necesidad hay:
    - Herramientas de análisis estático
    - OWASP CodeReview Guide
    - OWASP AppSecurity Metrics Project



*Papel actual del análisis  
estático en OWASP*





- “*Economics and Security Resource Page*” (esp. *Economics of vulnerabilities*), R. Anderson.  
<http://www.cl.cam.ac.uk/~rja14/econsec.html>
- *Secure Programming with Static Analysis*, B. Chess and J. West. Addison-Wesley. ISBN: 0-321-42477-8
- *Exploiting Software: How to Break Code*. G. Hoglund, G. McGraw. Addison-Wesley. ISBN: 0-201-78695-8
- *Software Security: Building Security In*. G. McGraw. Addison-Wesley. ISBN: 0-321-35670-5
- *19 Deadly Sins of Software Security*. M. Howard, D. Leblanc, J. Vieiga. McGraw-Hill. ISBN 0-07-226085-8
- Static Analysis for Security -  
<http://www.cigital.com/papers/download/bsi5-static.pdf>
- “*El análisis de código, fuente de seguridad*”. J. Crespo. Revista SIC, nº 74, Abril 2007.