



GRUPO GESFOR



ARGENTINA

COLOMBIA

CHILE

ESPAÑA

EE.UU.

MÉXICO

PANAMÁ

PERÚ

VENEZUELA

WAPITI

David del Pozo González  
[dpozog@grupogesfor.com](mailto:dpozog@grupogesfor.com)

Escaner de vulnerabilidades de aplicaciones  
web y auditor de seguridad

Junio 2010



■ ■ ■ © Gesfor

[www.gesfor.es](http://www.gesfor.es) | [www.grupogesfor.com](http://www.grupogesfor.com)

VI OWASP Spain  
Chapter Meeting



Patrocina:



Colabora:



GRUPO GESFOR



## Agenda

- ▶ Introducción
- ▶ Wapiti
  - Vulnerabilidades que detecta
    - OWASP Top 10
  - Técnica de fuzzing
  - Informes
  - Caso práctico
  - Página web
  - Futuro



# Introducción

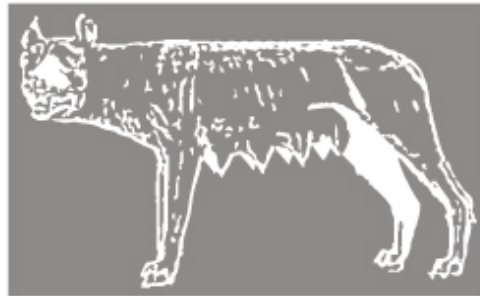


## Introducción

- ▶ Wapiti es un proyecto creado por Nicolas Surribas
- ▶ Grupo Gesfor comenzó a colaborar con Wapiti a través del proyecto europeo ICT Romulus
- ▶ Escaner de aplicaciones web.
- ▶ Licencia GNU General Public License 2.0
- ▶ Orientación al desarrollador



# Proyecto Romulus



**ROMULUS**



## Proyecto Romulus

- ▶ Proyecto financiado por la Comisión Europea dentro del “Seventh Framework Programme for Research and Technological Development”.
- ▶ Desarrollo pragmático, **fiable** y **seguro** de aplicaciones web mediante un diseño dirigido por el dominio y desarrollo orientado a mashups basado en un metaframework de Java de código abierto.
  - Mejorar la calidad del desarrollo del software haciendo hincapié en la **fiabilidad**, trazabilidad, **seguridad** y rendimiento, así como otros atributos de calidad.





## Proyecto Romulus: Objetivo “Seguridad”

- ▶ Problema principal:
  - Los desarrolladores no tienen conocimientos de los aspectos de seguridad
- ▶ Soluciones:
  - El metaframework debe generar aplicaciones seguras (generación automática de código).
  - El metaframework debe permitir la definición de restricciones de seguridad
  - **Los desarrolladores deben poder testear la seguridad de sus aplicaciones => WAPITI**





# Wapiti

**Escaner de vulnerabilidades de aplicaciones web  
y auditor de seguridad**







## Wapiti: Introducción

- ▶ Proyecto creado en 2006 por Nicolas Surribas
- ▶ Contribuciones de Gesfor desde 2008
- ▶ Características:
  - Escrito en Python: multiplataforma
  - Interfaz a través de comandos por consola
  - Enfoque de caja negra (Técnica utilizada: fuzz testing)





## Wapiti: Características

- ▶ Vulnerabilidades que detecta:
  - Inyección XSS (persistentes y no persistentes)
  - Inyecciones SQL (incluyendo SQL ciegas)
  - Inyección CRLF
  - Inyección LDAP
  - Errores en la gestión de ficheros
  - Ejecución de comandos en el servidor
  - Búsqueda de copias de seguridad (backups)
  - Configuraciones htaccess débiles





## Wapiti: Características

- ▶ **¿A quién está dirigido?**
  - Desarrolladores
    - Configuración por defecto es sencilla
    - Informes explicativos
  - Auditores de seguridad
    - Recoge todas las URL de la aplicación
    - Pueden explotar la potencia de Wapiti





# Vulnerabilidades comunes de seguridad (OWASP TOP 10)



## Vulnerabilidades comunes de seguridad (OWASP Top 10)

- ▶ 1. Injection ✓
- ▶ 2. Cross-Site Scripting (XSS) ✓
- ▶ 3. Broken Authentication and Session Management
- ▶ 4. Insecure Direct Object References
- ▶ 5. Cross-Site Request Forgery (CSRF)
- ▶ 6. Security Misconfiguration ✓
- ▶ 7. Insecure Cryptographic Storage
- ▶ 8. Failure to Restrict URL Access
- ▶ 9. Insufficient Transport Layer Protection
- ▶ 10. Unvalidated Redirects and Forwards



# Funcionamiento de Wapiti

## (Fuzz Testing)





## Detección de vulnerabilidades

- ▶ Tres tipos de técnicas
  - Pruebas de caja blanca
    - Análisis estático y dinámico de código
  - **Pruebas de caja negra**
    - Enfoque de atacante externo
  - Pruebas de caja gris
    - Enfoque mixto





## Wapiti

### Fuzz Testing (Funcionamiento)

- ▶ **1: Detección de vectores de entrada**
  - Enlaces
  - Formularios
- ▶ **2: Ataque**
  - Inyección de cadenas maliciosas para probar los diferentes ataques
- ▶ **3: Estudio de la respuesta**
  - Errores, cadenas inyectadas, etc.







## Wapiti: Primera etapa (Web Crawler)

- ▶ **Objetivo:** descubrimiento de vectores de entrada
  - Formularios y enlaces
- ▶ Utiliza la librería httpplib2 (antes urllib2 de Python)
  - Más eficiente
  - <http://code.google.com/p/httpplib2>





## Wapiti: Primera etapa (Web Crawler)

### ► Problemas encontrados I:

- Autenticación HTTP:
  - Solución: Opción *auth*: -a <login%password>
- Mantenimiento de sesión con cookies:
  - Solución: Opción *cookie*: -c <cookie\_file>
  - Wapiti incluye una herramineta con la que es posible la generación de cookies.





## Wapiti: Primera etapa (Web Crawler)

### ► Problemas encontrados II:

- HTML Mal formado:
  - Beautiful Soap Library:
  - [www.crummy.com/software/BeautifulSoup/](http://www.crummy.com/software/BeautifulSoup/)
- Enlaces infinitos (Problema “calendarios”)
  - Solución: Opción *nice*: -n <limit>
    - <http://www.server.com/p?a=x&b=1&c=x>
    - <http://www.server.com/p?a=x&b=2&c=x>
    - <http://www.server.com/p?a=x&b=2&c=y>





## Wapiti: Primera etapa (Web Crawler)

### ► Problemas encontrados III:

- Test largos, escaneo muy grande
  - Solución: Archivos temporales
    - Se puede parar y reanudar el escaneo (opción -i [<archivo>]):
      - » URLs
      - » Cabeceras
      - » Formularios



## Wapiti: Primera etapa (Web Crawler)

```
-<root>
- <rootURL>
  http://localhost:8080/webgoat/attack
</rootURL>
<toBrowse/>
- <browsed>
  - <url_data uri="http://localhost:8080/webgoat/attack">
    <header name="status" value="200"/>
    <header name="content-length" value="3774"/>
    <header name="content-location" value="http://localhost:8080/webgoat/attack"/>
    <header name="set-cookie" value="JSESSIONID=018A467706D155FC409BDEE72E297B4B; Path=/webgoat"/>
    <header name="expires" value="Thu, 01 Jan 1970 01:00:00 CET"/>
    <header name="server" value="Apache-Coyote/1.1"/>
    <header name="link_encoding" value="iso-8859-1"/>
    <header name="pragma" value="No-cache"/>
    <header name="cache-control" value="no-cache"/>
    <header name="date" value="Wed, 09 Jun 2010 08:47:48 GMT"/>
    <header name="content-type" value="text/html; charset=ISO-8859-1"/>
  </url_data>
</browsed>
- <forms>
  - <form encoding="iso-8859-1" to="http://localhost:8080/webgoat/attack" url="http://localhost:8080/webgoat/attack">
    - <inputs>
      <input name="start" value="Start WebGoat"/>
    </inputs>
  </form>
</forms>
<uploads/>
</root>
```



## Wapiti: Primera etapa (Web Crawler)

### ► Limitaciones:

- Enlaces en Javascript (Wapiti no interpreta JavaScript => los links no son seguidos)
- Páginas con la misma URL sin parámetros (Interpreta que son la misma página)
- Internet profunda u oculta => No la ve

Limitaciones inherentes del  
enfoque de **Web Crawler**





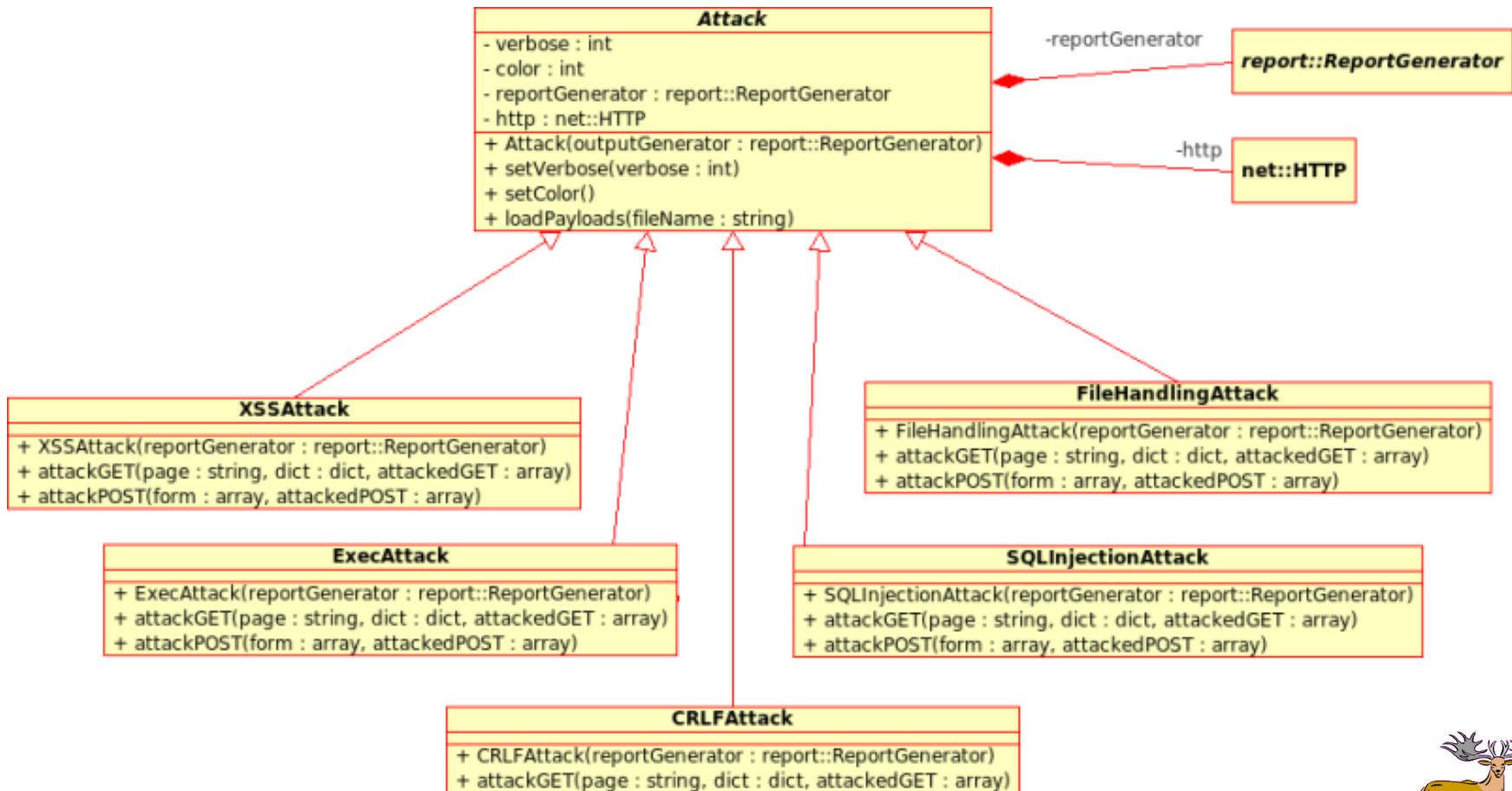
## Wapiti: Segundo paso (Fuzzing Testing)

- ▶ **Ataque de los vectores identificados en el primer paso:**
  - Inyección de cadenas maliciosas para descubrir vulnerabilidades existentes
  - Fácilmente extensible mediante:
    - Implementación de nuevos ataques
    - Nuevas cadenas “maliciosas”





## Wapiti: Segundo paso (Fuzzing Testing)







## Wapiti: Segundo paso (Fuzzing Testing)

### ► Clases de ataques implementan los métodos:

- def attackGET(self, page, dict, headers = {})
  - *page*: la URL de la página
  - *dict*: parámetros de la URL (clave y valor)
- def attackPOST(self, form)
  - *form*: array con información del formulario (página donde se encuentra el formulario, action del formulario y parámetros del formulario)





## Wapiti: Segundo paso (Fuzzing Testing)

### ► Ficheros con cadenas maliciosas

```
<script>alert('__XSS__')</script>
<script>alert("__XSS__")</script>
<ScRiPt>alert('__XSS__')</sCrIpT>
<ScRiPt>alert("__XSS__")</sCrIpT>
<script>String.fromCharCode(0,__XSS__,1)</script>
<ScRiPt>String.fromCharCode(0,__XSS__,1)</sCrIpT>
<script src=http://__XSS__/x.js></script>
<ScRiPt src=http://__XSS__/x.js></sCrIpT>

<img src=javascript:alert('__XSS__') />
<img src=javascript:alert("__XSS__") />
<img src=javascript:String.fromCharCode(0,__XSS__,1) />
<img src=JaVaScRiPt:String.fromCharCode(0,__XSS__,1) />
<img src=JaVaS\tcRiPt:String.fromCharCode(0,__XSS__,1) />
<img src=jav&#x09;ascript:alert('__XSS__'); />
<img src=jav&#x09;ascript:alert("__XSS__"); />
<img src=validimg.png onload=alert("__XSS__") />
<img src=validimg.png onload=alert('__XSS__') />
<img src=validimg.png onload:String.fromCharCode(0,__XSS__,1) />
```





## Wapiti: Segundo paso (Fuzzing Testing)

### ► Problemas encontrados I:

- Reescaneo de aplicación con nuevo test
  - Solución: Archivos temporales
    - Se puede probar otra vez la aplicación con las mismas URL que encontró el crawler (opción -k <archivo>)



## Wapiti: Tercer paso (Análisis de la respuesta)

- ▶ **Estudio de la respuesta obtenida para descubrir vulnerabilidades:**
  - HTTP Status
  - Contiene trazas de error
  - Contiene los mismos datos inyectados





## Wapiti: Informes de vulnerabilidades

- ▶ **Informes en diferentes formatos:**
  - XML
  - HTML
- ▶ **Información sobre las vulnerabilidades encontradas:**
  - URL y sus parámetros vulnerables
  - Información sobre la vulnerabilidad
  - Referencias sobre ella
  - ¿Cómo evitarla?



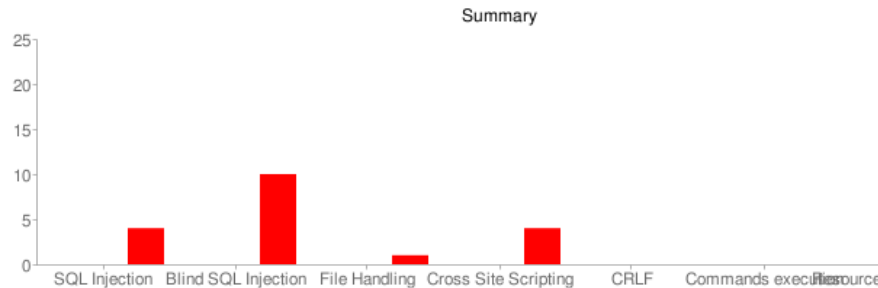


# Wapiti: Informes de vulnerabilidades



## Vulnerabilities report -- Wapiti

### Summary



	SQL Injection	Blind SQL Injection	File Handling	Cross Site Scripting	CRLF	Commands execution	Resource consumption
High	4	10	1	4	0	0	0
Medium	0	0	0	0	0	0	7
Low	0	0	0	0	0	0	0

### Attacks details

#### SQL INJECTION

**Description:** SQL injection is a technique that exploits a vulnerability occurring in the database of an application.

**Solution:** To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, user input must be escaped or filtered or parameterized statements must be used.

**References:**

- [http://www.owasp.org/index.php/SQL\\_injection](http://www.owasp.org/index.php/SQL_injection)
- [http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)



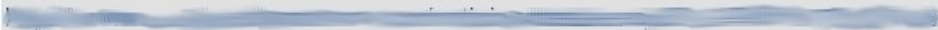


# Wapiti: Informes de vulnerabilidades

## Attacks details

### SQL INJECTION

- Description:** SQL injection is a technique that exploits a vulnerability occurring in the database of an application.
- Solution:** To protect against SQL injection, user input must not directly be embedded in SQL statements. Instead, user input must be escaped or filtered or parameterized statements must be used.
- [http://www.owasp.org/index.php/SQL\\_Injection](http://www.owasp.org/index.php/SQL_Injection)
  - [http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection)
- References:**

Risk Level	High
Url	
Parameter	ID=%BF%27%22%28
Info	MSSQL-Based Injection (ID)

Risk Level	High
Url	
Parameter	ID=%BF%27%22%28
Info	MSSQL-Based Injection (ID)

Risk Level	High
Url	
Parameter	ID=%BF%27%22%28
Info	MSSQL-Based Injection (ID)





## Wapiti: Caso práctico

### ► Atacar Webgoat:

- Configuración normal con informe HTML:
  - `python wapiti.py http://localhost/webgoat/attack -v 2 -f html -o webgoat`
- Configuración añadiendo autenticación HTTP:
  - `python wapiti.py http://localhost/webgoat/attack -v 2 -f html -o webgoat -a guest%guest`
- Configuración añadiendo cookie:
  - `python wapiti.py http://localhost/webgoat/attack -v 2 -f html -o webgoat -a guest%guest -c cookie`





## Wapiti: Caso práctico

### ► Cookie:

- Script de generación de cookies:  
net/getcookie.py

```
<?xml version="1.0" encoding="UTF-8"?>
<cookies>
  <domain name="localhost">
    <cookie path="/webgoat"
      name="JSESSIONID"
      value="1AA56D2C50031E6BA62E7AE773A323F0"
      version="0"/>
  </domain>
</cookies>
```



## Wapiti: Caso práctico

### ► Atacar Webgoat:

#### – Configuración añadiendo opción “nice”

- `python wapiti.py http://localhost/webgoat/attack  
-v 2 -f html -o webgoat -a guest%guest -c cookie  
-n 3`



## Wapiti: Ventajas y desventajas

### ▶ Desventajas:

- Wapiti no puede descubrir todas las vulnerabilidades

### ▶ Ventajas:

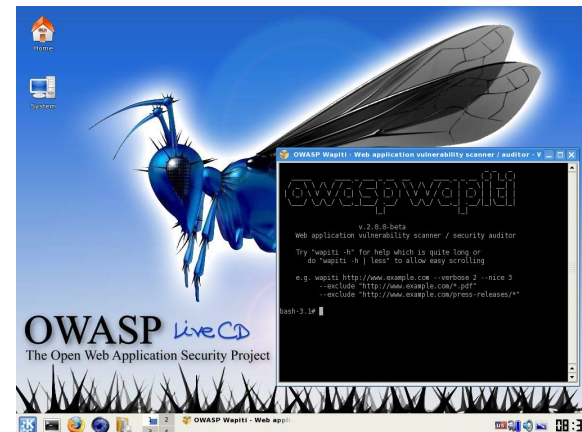
- El usuario no necesita conocimientos de seguridad
- Wapiti descubre las vulnerabilidades más comunes (según el OWASP Top Ten)
- Nuevos ataques pueden ser añadidos fácilmente





## Wapiti: Resultados

- ▶ Más de 33.000 descargas de SourceForge
- ▶ Incluido como Proyecto Alpha OWASP (revisión)
- ▶ Incluido en las distribuciones GNU/Linux de seguridad más importantes:
  - OWASP Live CD
  - BackTrack





## Wapiti: Últimas actualizaciones

Versión	Mejoras
2.0	<b>Generación de informes</b> <b>Refactorización a un enfoque Orientado a Objetos</b> <b>Extensibilidad de las cadenas maliciosas</b> <b>Opción “nice”</b> <b>Mejora de la documentation</b> <b>Nuevo portal para Wapiti</b> <b>Mejoras en ataques XSS</b>
2.1	<b>Mejora de eficiencia (usando librería httplib2)</b> <b>Ataques de inyecciones SQL ciegas</b> <b>Herramienta de creación cookies</b> <b>Mejora en ataque XSS</b>
2.2	<b>Nuevos ataques basados en base de datos de Nikto</b> <b>Opción <i>scope</i> (ámbito)</b> <b>Archivos temporales</b> <b>Internationalización (Inglés, Español y Francés)</b>





## Wapiti: Página web

- ▶ Características de cada versión
- ▶ Blog
- ▶ Wiki
  - Introducción
  - Comienzo rápido
  - Guías de usuario
  - FAQ
- ▶ Roadmap
- ▶ Vídeos...



### Wapiti

Web application vulnerability scanner / security auditor

#### Presentation

Wapiti allows you to audit the security of your web applications.

It performs "black-box" scans, i.e. it does not study the source code of the application but it will scan the webpages of the deployed webapp, looking for scripts and forms where it can inject data.

Once it gets this list, Wapiti acts like a fuzzer, injecting payloads to see if a script is vulnerable.

Wapiti can detect the following vulnerabilities :

- File Handling Errors (Local and remote include/require, fopen, readfile...)
- Database Injections (PHP/JSP/ASP SQL Injections and XPath Injections)
- XSS (Cross Site Scripting) Injection



## Wapiti: Futuro

- ▶ Version 2.3 (en progreso). Mejoras:
  - Inyecciones SQL
  - Paralelizar los ataques mediante hilos
  - Gestión automática de cookies





# VulneraNET

**Herramientas y Procesos Colaborativos de  
Detección, Predicción y Corrección de  
Vulnerabilidades de aplicaciones web para  
desarrolladores y auditores de seguridad**

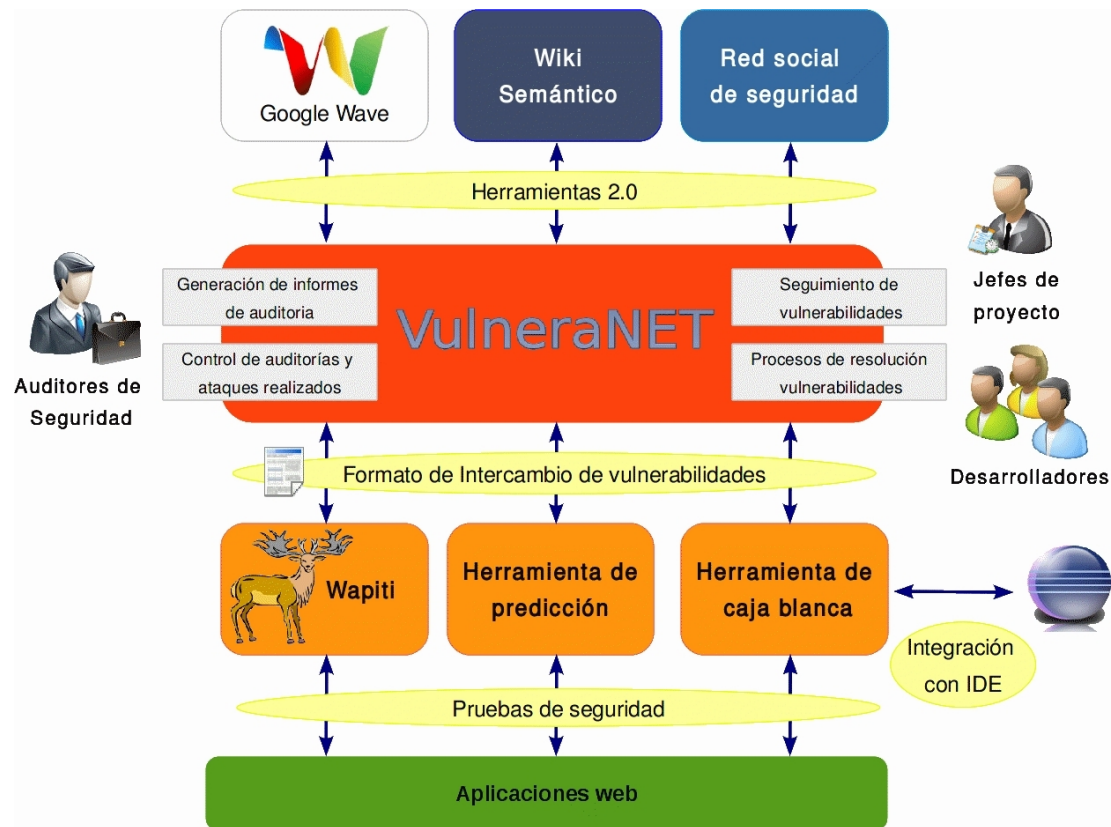




# VulneraNET

## Objetivos:

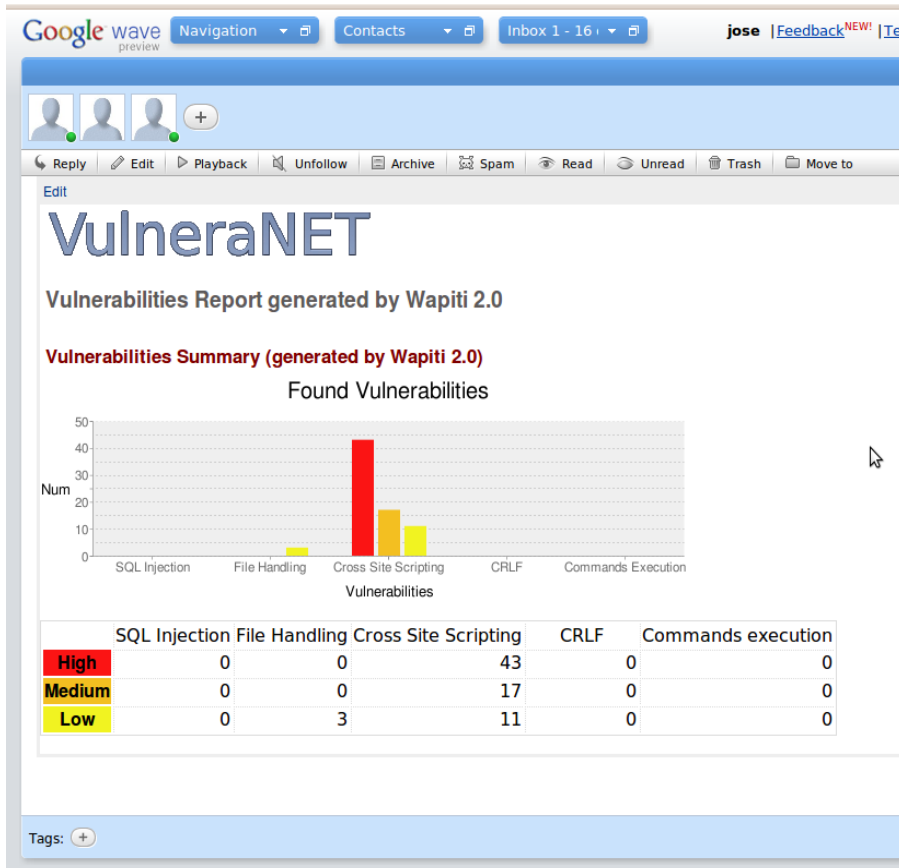
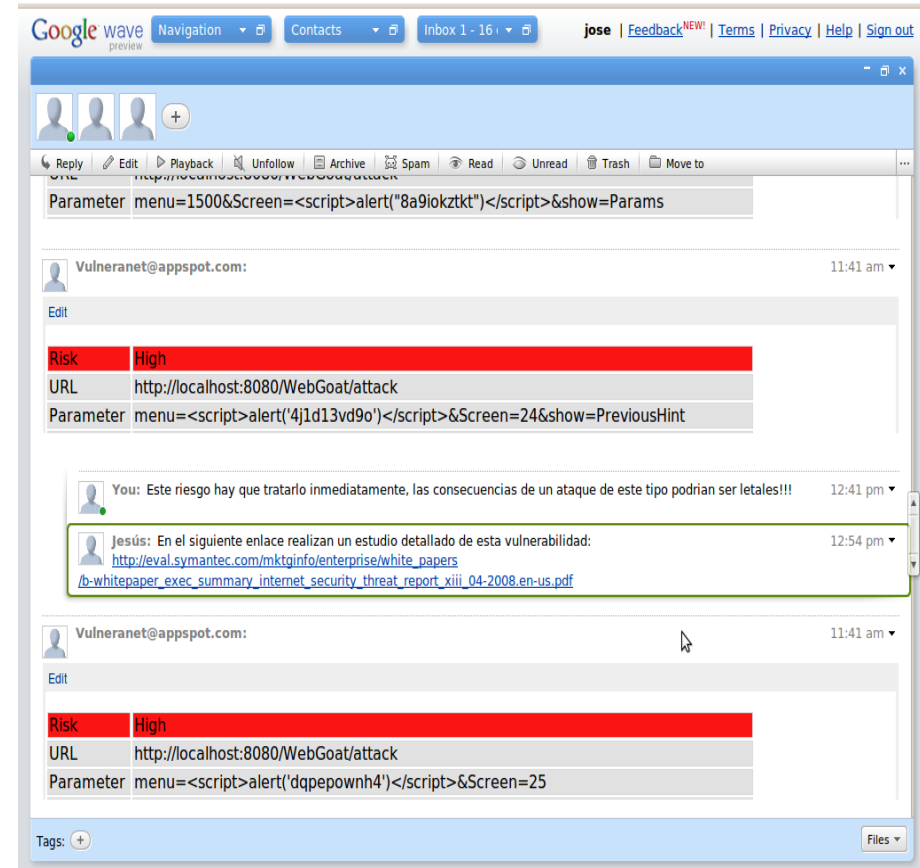
- Detección de vulnerabilidades (caja negra y caja blanca)
- Gestión de información de vulnerabilidades
- Aplicación de tecnologías web 2.0 y semánticas
- Gestión centralizada y generación de informes de seguridad



VulneraNET



# VulneraNET: Wapiti

Parameter menu=1500&Screen=<script>alert("8a9iokztk")</script>&show=Params

Vulneranet@appspot.com: 11:41 am

**Risk** High

URL http://localhost:8080/WebGoat/attack

Parameter menu=<script>alert('4j1d13vd9o')</script>&Screen=24&show=PreviousHint

You: Este riesgo hay que tratarlo inmediatamente, las consecuencias de un ataque de este tipo podrian ser letales!!! 12:41 pm

Jesús: En el siguiente enlace realizan un estudio detallado de esta vulnerabilidad:  
[http://eval.symantec.com/mktginfo/enterprise/white\\_papers/b-whitepaper\\_exec\\_summary\\_internet\\_security\\_threat\\_report\\_xiii\\_04-2008.en-us.pdf](http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf) 12:54 pm

Vulneranet@appspot.com: 11:41 am

**Risk** High

URL http://localhost:8080/WebGoat/attack

Parameter menu=<script>alert('dqpepownh4')</script>&Screen=25

# VulneraNET



**David del Pozo González**  
**dpozog@grupogesfor.com**



**GRUPO GESFOR**

**Proveedor Global de TI y RR.HH.**



**[www.gesfor.es](http://www.gesfor.es)**

**<http://innovacion.grupogesfor.com>**

**WAPITI: <http://www.ict-romulus.eu/web/wapiti>**

**VulneraNET: <http://vulneranet.grupogesfor.com>**



**Un grupo sin fronteras**