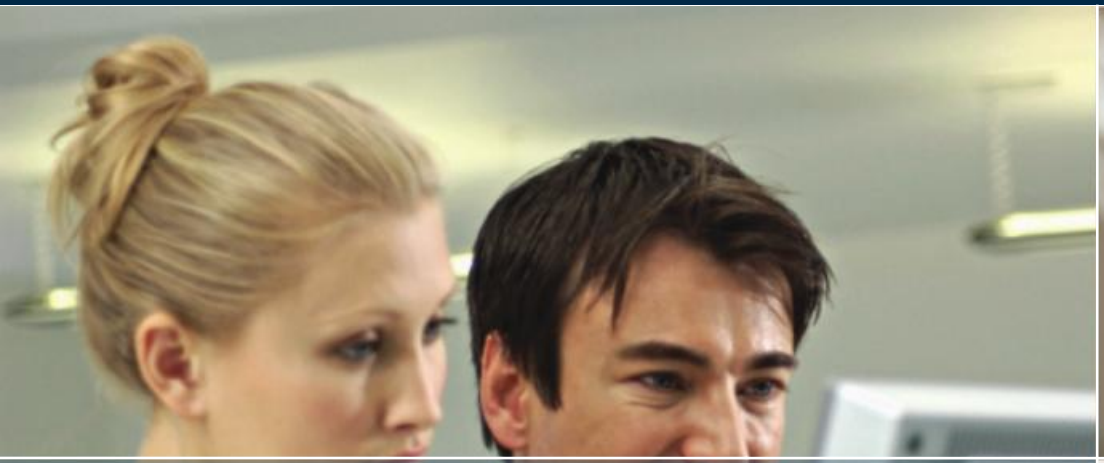
Su Seguridad es Nuestro Éxito

internet security auditors

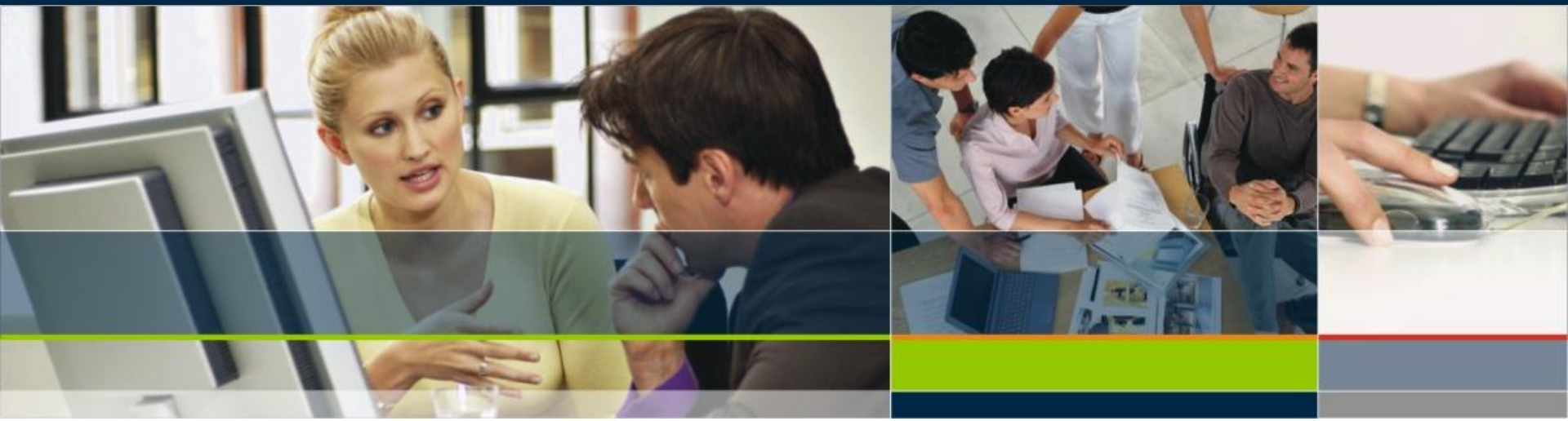


c. Santander, 101. Edif. A. 2º I 08030 Barcelona I Tel.: 93 305 13 18 I Fax: 93 278 22 48 I **info@isecauditors.com** I **www.isecauditors.com**

Análisis de Eco

OWASP Conference, Noviembre 2008

Jesus Olmos Gonzalez (jolmos@isecauditors.com)



Su Seguridad es Nuestro Éxito

internet security auditors

Índice

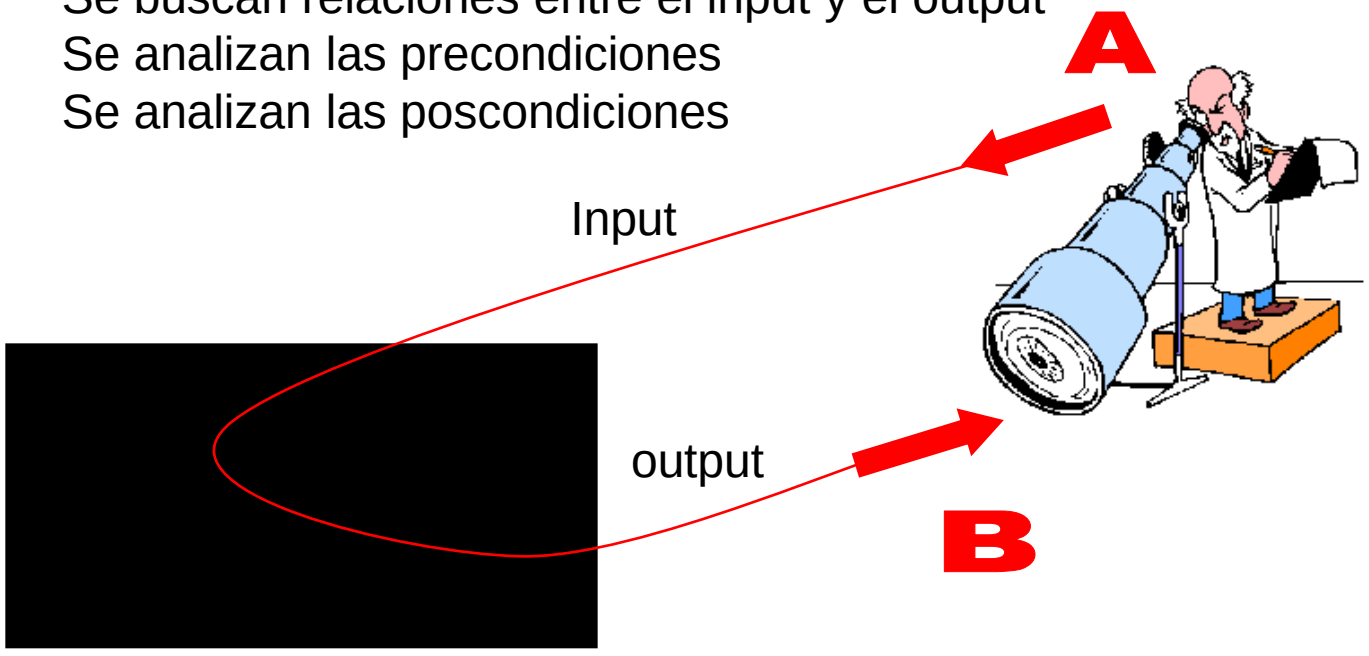
1. Problemática en la auditoría de Caja Negra.

- No vemos el código.
- Imposible probar todos los casos.
- Los tipos de vulnerabilidades Web hoy día son “Well Known”
  (la humanidad cuenta con un rico histórico de vulnerabilidades)
- Los desarrolladores validan los datos cada vez más. (pero no mejor)
- Se realizan validaciones deficientes que no protegen la vulnerabilidad y además dificultan su detección.

Ejemplo: ‘ → \’
/^’.*(or|and)/

2. Caja Negra sin eco.

- Se induce un input
- Se observa el output
- Se buscan relaciónes entre el input y el output
- Se analizan las precondiciones
- Se analizan las poscondiciones



internet security auditors

3. Deducción de código.

- En vez de ir realizando ataques hasta que alguno “funcione” es más efectivo tantear en busca del fallo y posteriormente explotarlo.
- Este tanteo, consiste en estudiar los input/output de cara a elaborar un pseudocódigo.
- Importante no atacar, sino analizar.

3. Deducción de código.

- Es más importante una interpretación correcta del output, que emitir un input acertado.

- Que componente interno nos da el error (BBDD, SA, SW, FWA, ..)

(Ejemplo real: percepción subconsciente de un FWA mediante un cambio de fuente)

- Que inputs “hacen daño”

(cancelación de operativa, timmings largos, errores no controlados)

- Cada aplicación funciona diferente, entender a los programadores, estudiar código de cliente, analizar errores.

- Variables de decisión, variables permanentes, variables de llamada externa.

3. Deducción de código.

- Correlación input → error, que input provoca que error.
- Significado subyacente de los errores y de los timmings.

Ejemplo de análisis de una operativa:

‘ → Error genérico.
> → Continua la operativa correctamente.
} → Tiempo de espera largo + Error genérico.
| → Continúa la operativa + Error genérico.

¿Qué byte ha hecho más daño?

3. Deducción de código – Timmings.

Acceso a Datos	Segundos
Acceso a Sistema de Ficheros	Milisegundos
Acceso a memoria	Nanosegundos

Retardo Internet	Milisegundos

4. Filtros vs Saneos.

- Los filtros permiten detectar y registrar el intento de ataque.
- Los filtros deniegan la operativa, el atacante puede darse cuenta que no se ha realizado la operativa. De manera que són más fáciles de analizar.
- Los saneos limpian las variables y proceden con la operativa.
- Los filtros y saneos se pueden hacer en una sola instrucción de código o diversas, el resultado no será el mismo. (análisis de orden de filtros)
- Los saneos se pueden entorpecer entre ellos.
- En ambos casos hay que tener en cuenta las transformaciones de datos, por ejemplo decodificaciones. En cuanto se codifique o decodifique o se transforme el valor se puede crear el ataque.

4. Filtros vs Saneos.

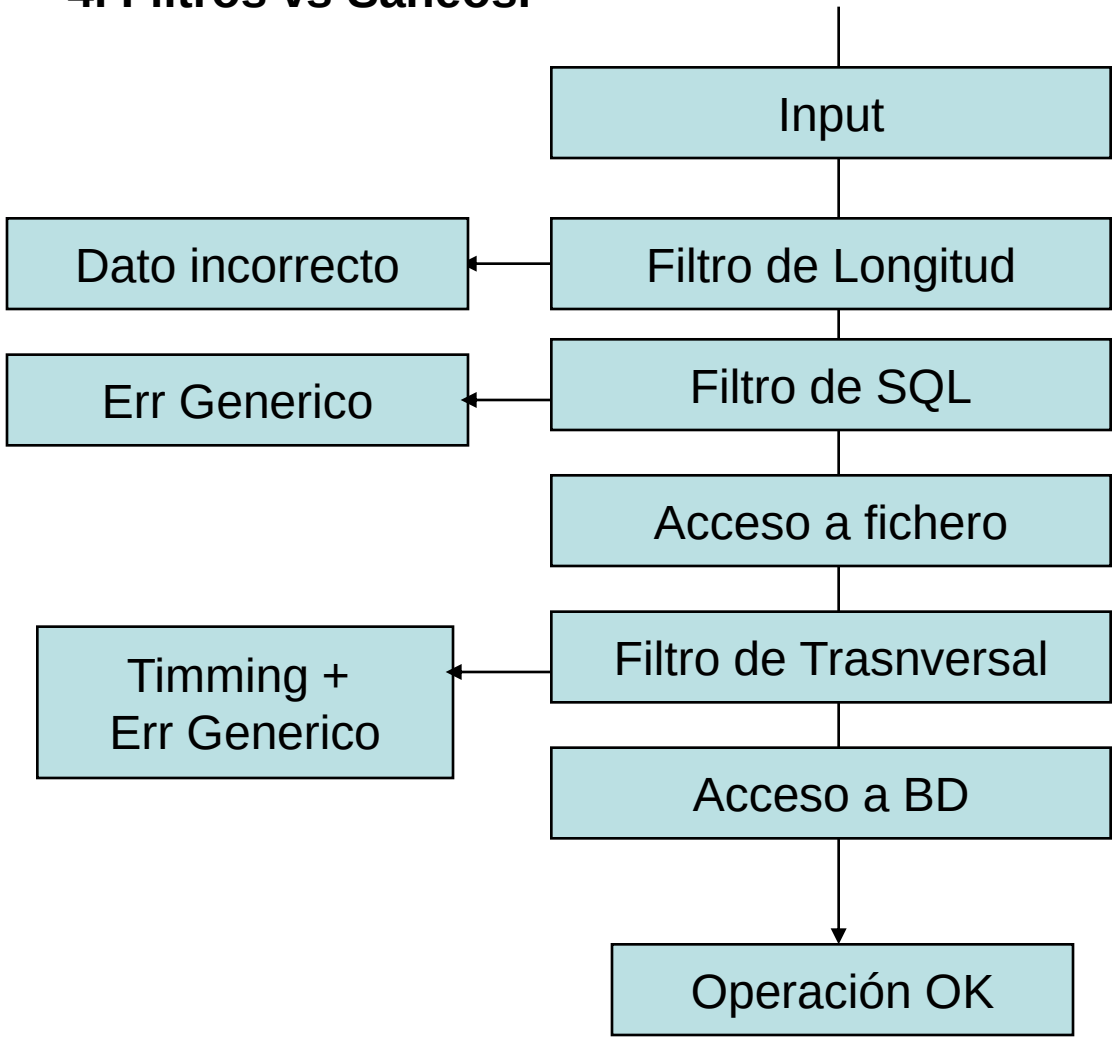
- La mayoría vulnerabilidades existen simplemente por el escape de contexto.

Ejemplo:

‘input’ → El input no deberá de contener ‘
input → El input no deberá de contener *
comando del sistema ping ‘input’ → input no deberá contener ‘

- Filtraremos o sanearemos el delimitador, para evitar escapes de contexto.

4. Filtros vs Saneos.



¿El código conduce al usuario o el usuario conduce al código?

5. Búsqueda de salidas.

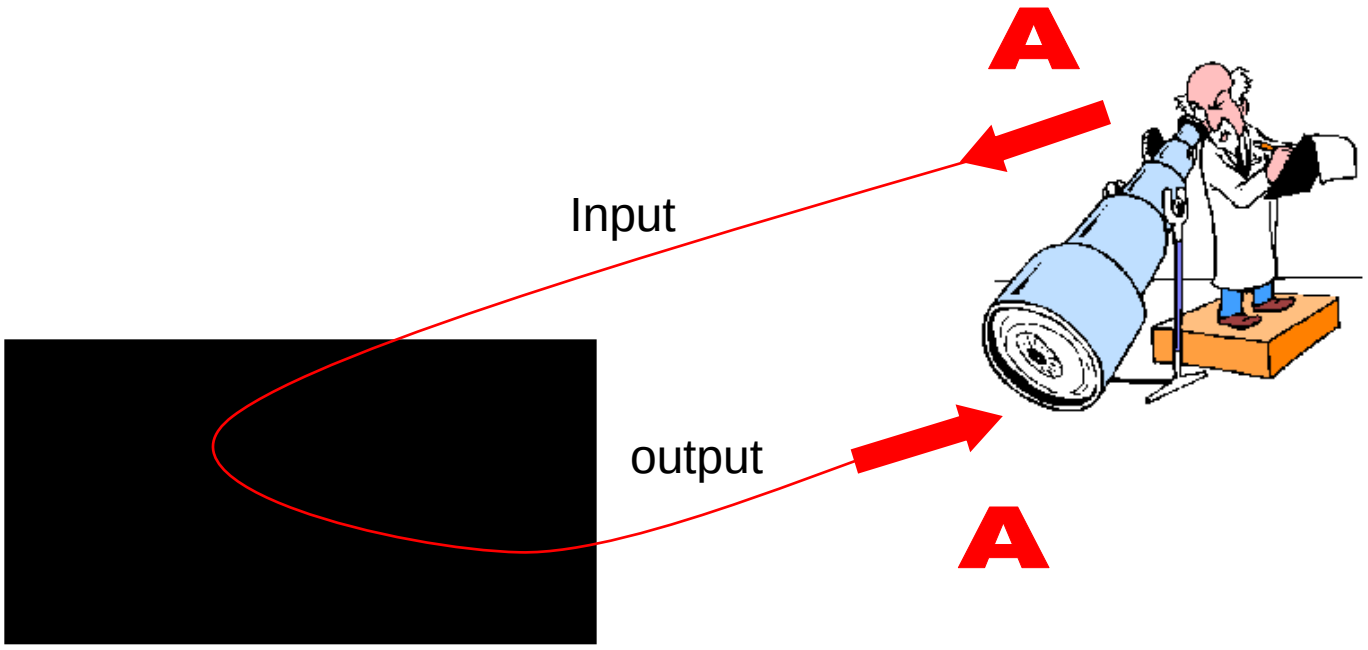
- Existe una vulnerabilidad, pero el filtro nos impide ver su existencia.
  → Analizar los filtros en vez de atacar directamente.
  → Aplicar evasiones.

- Es posible incluso estar explotándola correctamente pero no percibirlo.
  → Buscar una salida.

  - Provocar retardos.
    (sleeps de SO, BD, …, carga de ficheros largos, SQL pesada)
  - Retroconexiones.
  - Envío por email
  - Aprovechar componentes que permitan una salida
    (ej envío de sms)
  - DoS en terceras aplicaciones

6. ¿Que es el eco o echo?

- Si input ~= output → eco



6. ¿Que es el eco o echo?

7. Análisis de eco

Ejemplo:
eco'|"< → Búsquedas relacionadas con: eco'|"<
eco* → <title>eco* - Buscar con Google</title>

- Saneados = { ‘,“,< }
- Aceptados = {e,c,o,|,*}

- Se puede determinar el alfabeto permitido con el cual crear palabras de ataque.

- El eco permite analizar con mayor exactitud las alteraciones que ha sufrido el input antes de llegar al output.

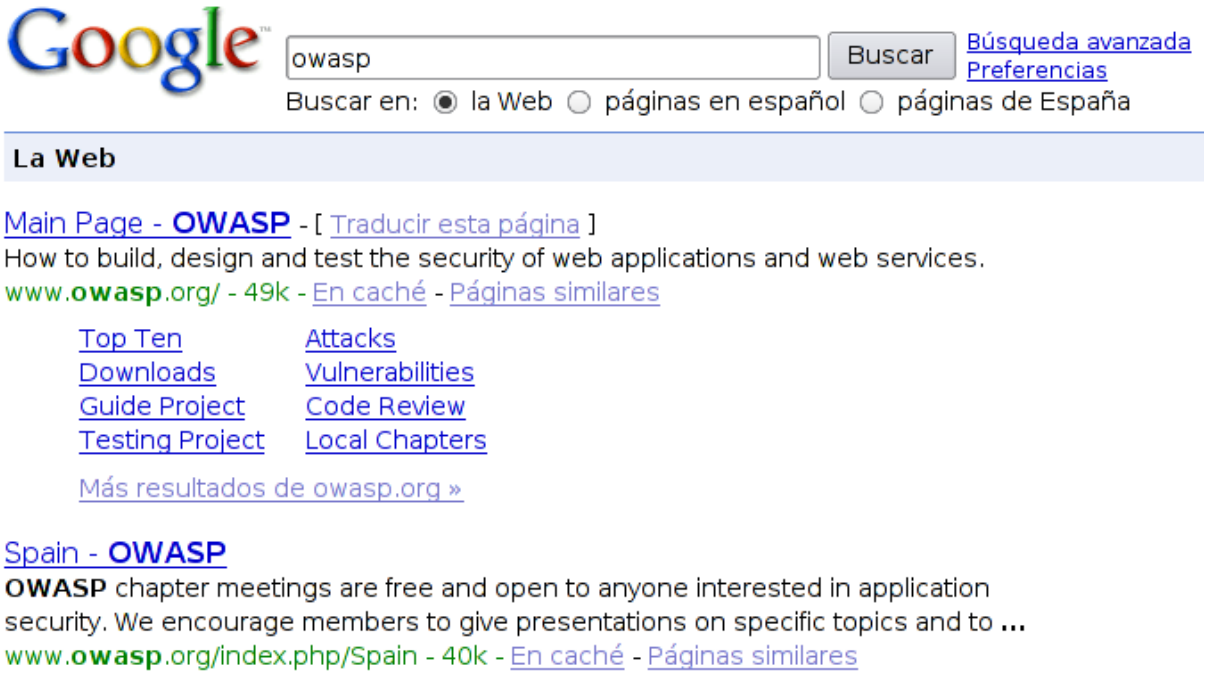
7. Análisis de eco

- Chequeo automatizado:

Is vulnerable to echo analysis
' ==> ' Sanitized
" ==> " Sanitized
` ==> ` Accepted
$ ==> $ Accepted
- ==> - Accepted
* ==> * Accepted
%3f(?) ==> ? Accepted
%26(%26) ==> & Sanitized
< ==> < Sanitized
> ==> > Sanitized
(==> (Accepted
) ==>) Accepted
…
%39(9) ==> 9 Accepted
%3a(:) ==> : Accepted
%3b(;) ==> ; Accepted
%3c(%3c) ==> < Sanitized
%3d(=) ==> = Accepted
%3e(%3e) ==> > Sanitized
%3f(?) ==> ? Accepted
%40(@) ==> @ Accepted
%41(A) ==> A Accepted
%42(B) ==> B Accepted
%43(C) ==> C Accepted
…

8. Eco indirecto

- No hay eco pero se puede intuir a partir de la respuesta.

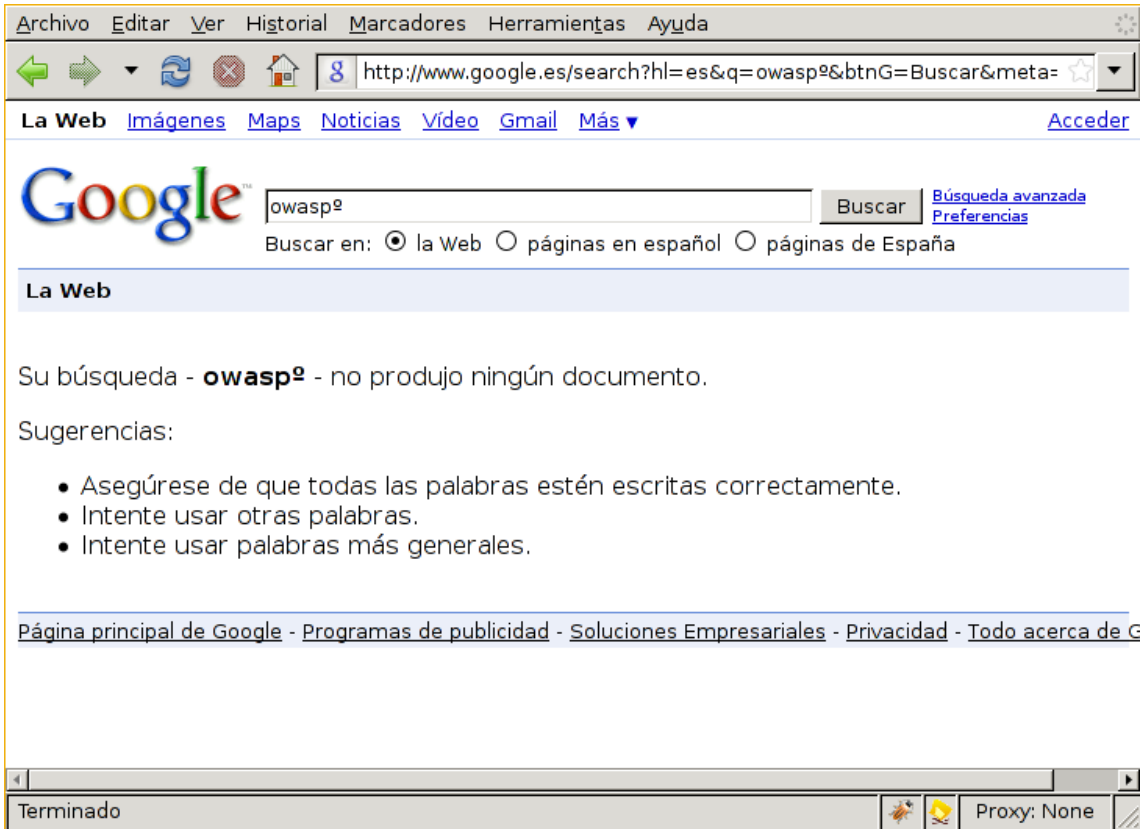


8. Eco indirecto

8. Eco indirecto

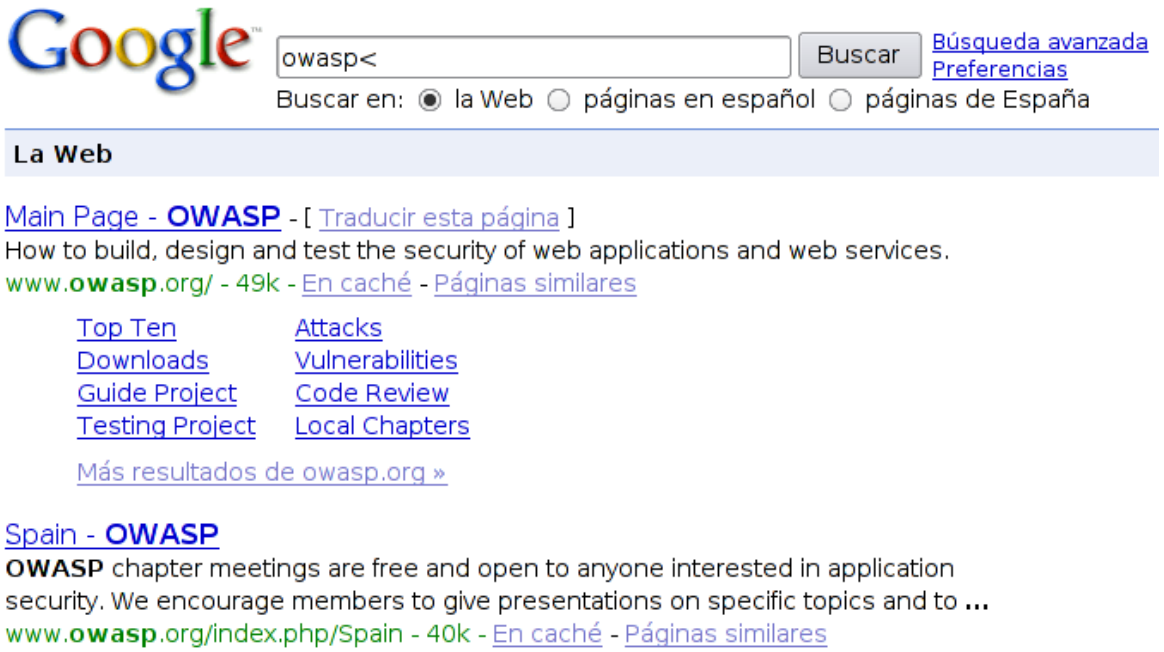
- No filtran ni sanean el asterisco.

8. Eco indirecto



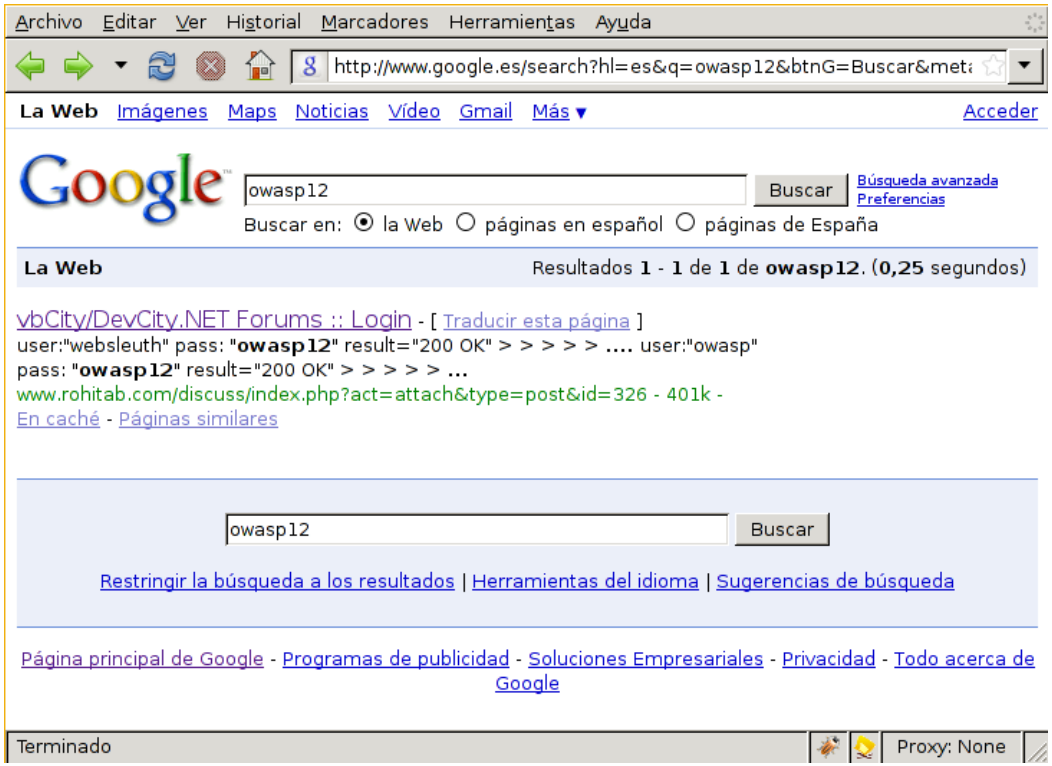
- No filtra el byte º owaspº
- Parece un error interno bien disimulado, pero no lo es.

8. Eco indirecto



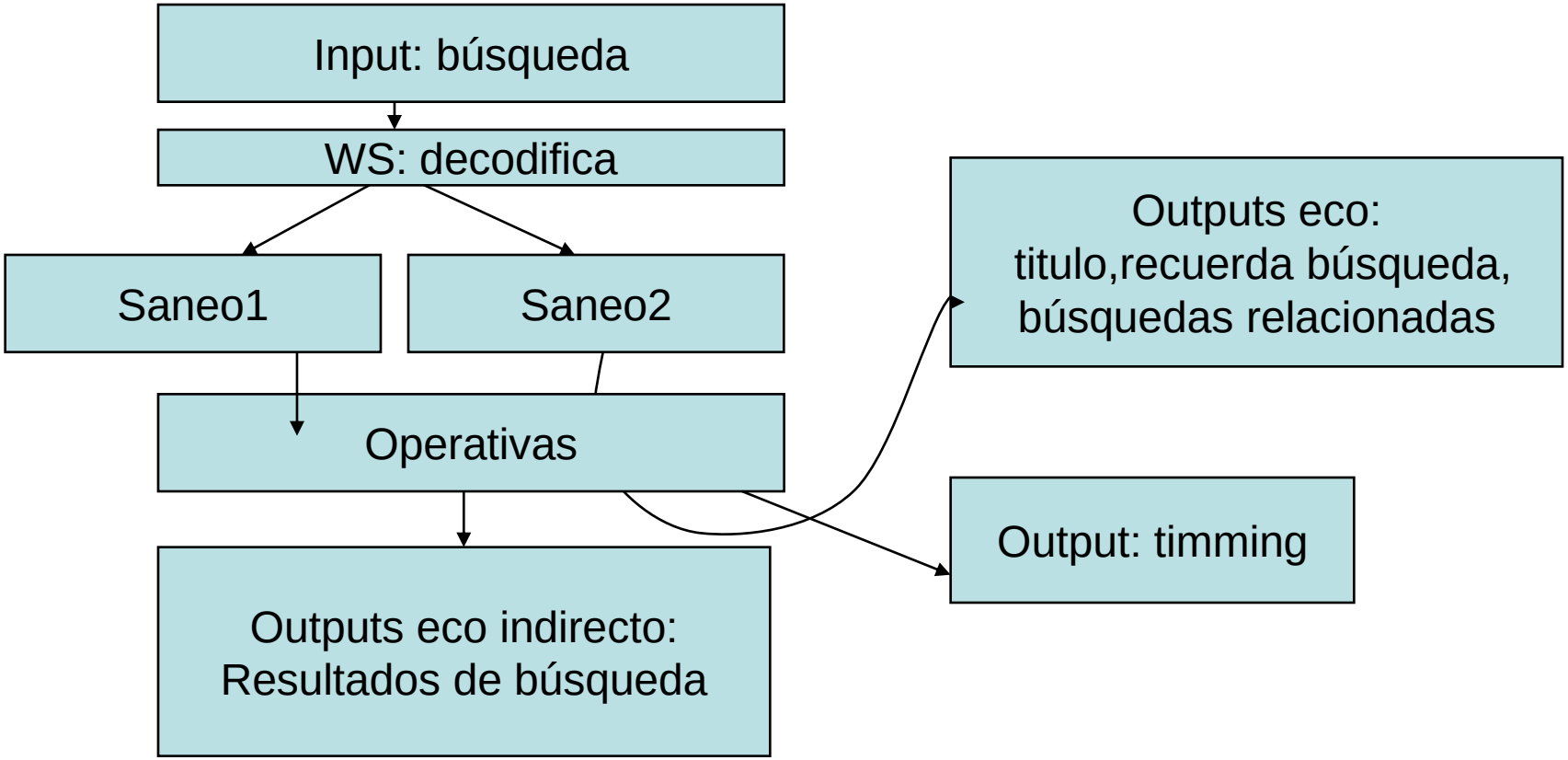
• En una variable cambia < por < sin embargo en otra lo suprime y no influye en la búsqueda ni a 3º party apps.

8. Eco indirecto



.

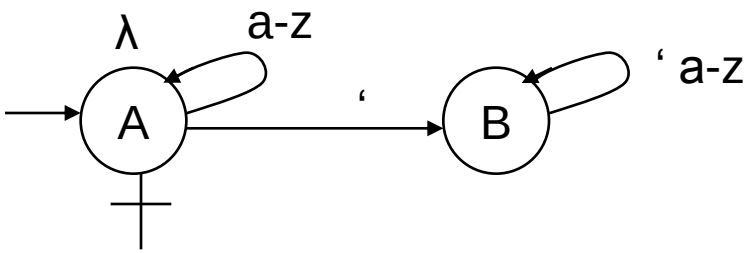
8. Eco indirecto



• En el eco el input está conectado con el output, en el “indirecto” tenemos un output generado a partir del input

9. Autómatas finitos de los filtros

- Una vez identificada la validación, se puede analizar los estados de aceptación de su autómata finito.
- Ejemplo:
  - Regexp: /^‘.*/ (todo lo que comience por comilla es un ataque)
  - Alfabeto del atacante E = { ‘ a-z }
  - Serán aceptadas: λ (null) o comillas que no estén al inicio



10. Conclusiones y Recomendaciones

- Analisis de filtro en vez de probar ataques y probar evasiones. (un analizador automático que haga esto está bien, detectará rápidamente las vulnerabilidades menos escondidas)
- Solventar los problemas desde diseño, el poner una validación en algunos casos implica tapar el problema.
- Diseñar evasiones personalizadas para el filtro / saneo identificado.
- Usar variables de decisión para esquivar validaciones.
- Reducir los inputs y outputs.
- Reducir la inferencia de los outputs.