



Attacking WordPress plugins with no-style nor time



OWASP

The Open Web Application Security Project

FARADAY

Attacking WordPress plugins with no-style nor time

- #howeare
- #whythis
- #introduction
- #demonstrations
- #conclusions



#howeare

info**byte**



?



#whythis

- ¿Por qué **Wordpress**?

Content Management Systems

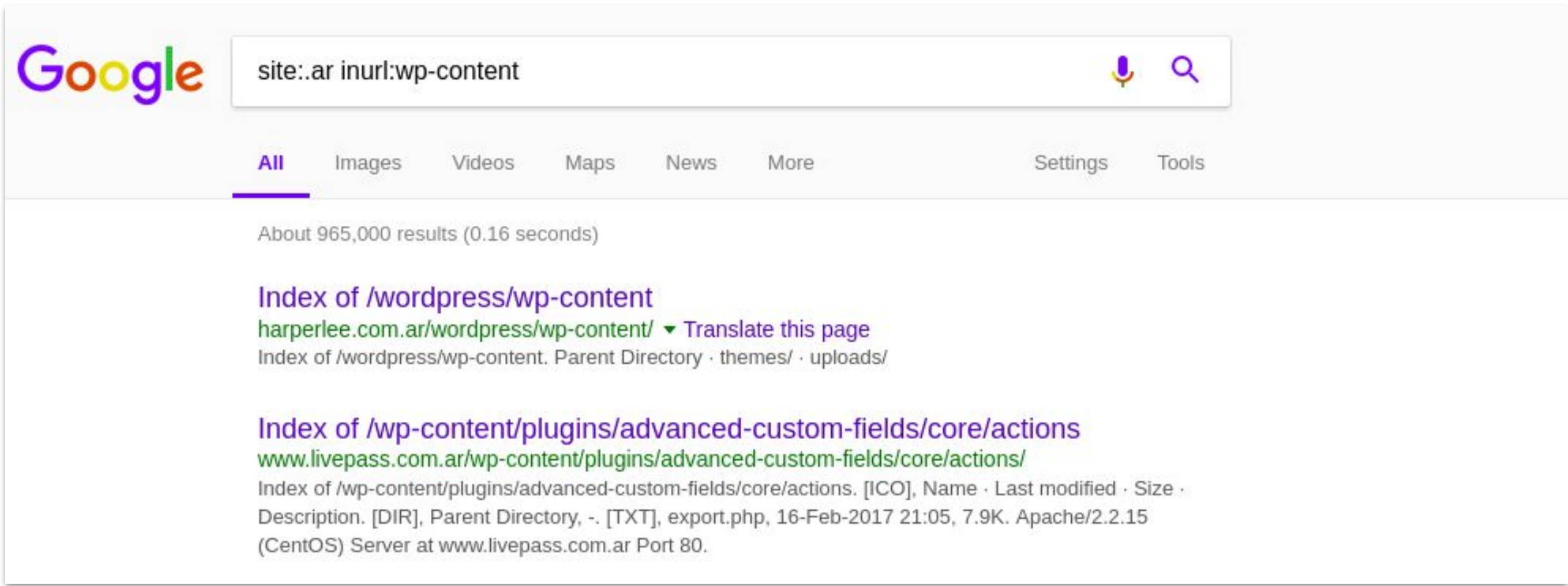
Most popular content management systems

© W3Techs.com	usage	change since 1 March 2018	market share	change since 1 March 2018
1. WordPress	30.4%	+0.5%	60.1%	-0.1%
2. Joomla	3.1%		6.2%	-0.1%
3. Drupal	2.2%		4.3%	-0.1%
4. Magento	1.2%		2.3%	-0.1%
5. Shopify	1.0%		2.0%	+0.1%

percentages of sites

#whythis

- 965.000 resultados solo hosteados en Argentina.

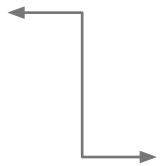


- ¿Por qué analizar los **plugins de Wordpress** y no el *core*?
 - El **core** es auditado siempre por muchos researchers.
 - Menos posibilidades de éxito.
- “La cadena se rompe siempre por el **eslabon mas debil**”

#whythis

Entonces, ¿plugins de **Wordpress**?

Están en
todos
lados



Varios que
no son
mantenidos



Miles y miles
de
instalaciones



#whythis

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2004	2						1		1						
2005	10		5			3	2				3				
2006	16	1	2			1	5	1			3				
2007	40	2	13			7	19			3	5		2		1
2008	27	2	4			3	9	4		1	2		2		
2009	14	3	1				3			1	3	1			4
2010	2		1			1									
2011	11					1	2				4				
2012	24	2	2			2	9			5	3		3		7
2013	19	1	1				8			3	2		1		
2014	29	3	3			1	8	1		6	2		3	1	
2015	11	1	2			1	7			1	1		1		
2016	20	1					9			6	1		1		
2017	46	1	1			4	17	4		5	3		5		
2018	2	1					1								
Total	273	18	35			24	100	10	1	31	32	1	18	1	12
% Of All		6.6	12.8	0.0	0.0	8.8	36.6	3.7	0.4	11.4	11.7	0.4	6.6	0.4	

Featured Plugins

[See all](#)

- Buscamos la manera de descargar todo el repositorio de plugins de Wordpress.
 - Encontramos el dominio plugins.svn.wordpress.org
 - Contiene los plugins, tanto los activos como los deprecados.



- Desarrollamos unos scripts para descargar todos los repositorios y generar las URLs de las mismas.

```
~/owasp >>> cat downloadAllPluginsSourceCode.sh
#!/bin/bash

for i in $(cat urlsPluginsSourceCode)
do
    svn co $i plugins.svn.wordpress.org/$(basename plugins.svn.wordpress.org/$(dirname $i))
done
```



#introduction

FASE 1

¿Cuales son las vulnerabilidades más comunes ?

Empecemos a *grepear* **todo**.



GREP



oops!

404. That's an error.



#introduction

FASE 2

`$_GET`
`$_POST`

Mmm, ¿ *user input* ?

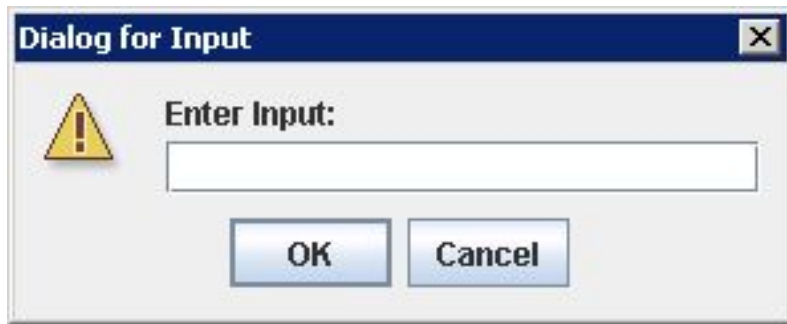
Necesitamos seguir cada entrada del usuario, si es usada en una función vulnerable.

¿ Están sanitizando con las funciones correctas ?





Migrar nuestras **regex** a Python.



Realizar el análisis del user input por nuestra parte.



oops!


404. That's an error.



#introduction (changing_the_strategy)

FASE 2'

Empezamos a buscar herramientas para análisis estático de código PHP.

 README.md

WPSploit

Aggressive Code Scanner for Wordpress Themes/Plugins

python **2.7** license **GPL**

This tool is intended for Penetration Testers who audit WordPress themes or plugins or developers who wish to audit their own WordPress code. This script should be used for learning purposes only. By downloading and running this script you take every responsibility for wrong or illegal uses of it.

For more informations about the vulnerabilities tested [click here](#).

phpvulhunter

phpvulhunter是一款PHP源码自动化审计工具，通过这个工具，可以对一些开源CMS进行自动化的代码审计，并生成漏洞报告。##安装 首先从github上进行获取：

```
git clone https://github.com/OneSourceCat/phpvulhunter
```

下载完成后，将工程目录放置于WAMP等PHP-Web运行环境中即可访问 `main.php`：

```
http://127.0.0.1/phpvulhunter/main.php
```

##使用 搭建好环境，访问main.php后，效果如下：



Phortress

A PHP static code analyser for potential vulnerabilities

Setting up

1. You will need [Composer](#).
2. Clone the Git repository: `git clone https://github.com/lowjoel/phortress.git`
3. Install dependencies: `composer.phar install`
4. Run unit tests: `phpunit`

build **passing**

PHP-Reaper

PHP tool to scan [ADODB](#) code for SQL Injections

Why

The main idea is to be able to detect problems as early as possible, when the code is fresh in your mind. Shift as much checks as possible to the left. Automate as much as possible.

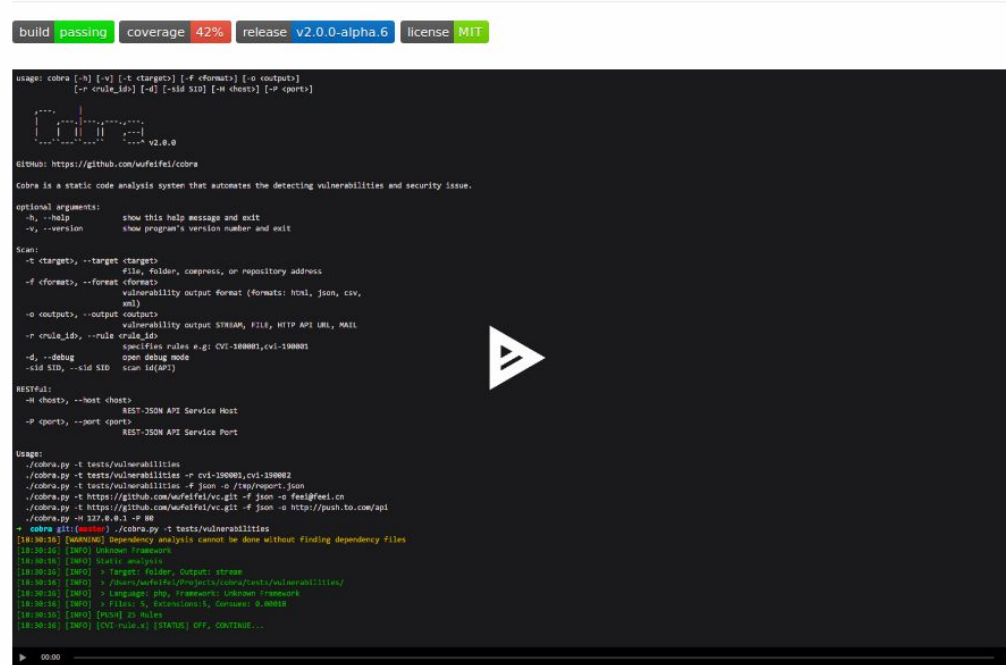
Running PHP-Reaper is far less time consuming than running full fledged automated security scanner at your application. The web security scanner might not locate all possible SQL Injections vulnerabilities, because of hard to reach code from the UI (or needs to set rare conditions). PHP-Reaper is fast and pinpoints the exact line where the problem lies, scanning all your PHP [ADODB](#) source code.

You'll get the most out of PHP-Reaper if you run it on every commit. It's made to be CI friendly and fast.



#introduction (the_chosen_one)

Cobra



Introduction (介绍)

Cobra是一款源代码安全审计工具，支持检测多种开发语言源代码中的大部分显著的安全问题和漏洞。

Features (特点)

Multi-language Supported (支持多种开发语言)

支持PHP、Java等开发语言，并支持数十种类型文件。

Multi-Vulnerabilities Supported (支持多种漏洞类型)

- Basada en **regex**.
- Soporta PHP y *tracea* los *user-input*.
- Soporta varios tipos de vulnerabilidades por defecto
- No incluye soporte nativo para WordPress



#introduction (the_chosen_one >> tuning)

- Agregamos tracking para las variables \$get & \$post (Especiales de los plugins de Wordpress)
- Detectaba como user input \$wpdb: *lo blacklistamos.*
 - *Creamos una regla custom para encontrar SQL Injection flaws en WordPress.*
- Tradujimos un poco el código a Inglés (**Chinese pls**)

文学家

#introduction (the_chosen_one >> test_cases)

```
-/D/cobra >>> source ./cobraVirtualEnv/bin/activate
(cobraVirtualEnv) ~/D/cobra >>> python2 ./cobra.py -t tests/vulnerabilities
17:08:19] [INFO] [REPORT] Report URL: ?sid=a938e23ahw5j
17:08:19] [INFO] [CLI] Target directory: /home/ezequiel/Downloads/cobra/tests/vulnerabilities
17:08:19] [INFO] [CLI] [STATISTIC] Language: php Framework: Spring
17:08:19] [INFO] [CLI] [STATISTIC] Files: 14, Extensions:13, Consume: 0.000862
17:08:19] [INFO] Can't find the rules, please update rules
17:08:19] [WARNING] [SCAN] [CVE] CVE rule is None
17:08:19] [INFO] [PUSH] 6 Rules
17:08:19] [INFO] [AST] String's variables: `$_GET`
17:08:19] [INFO] [AST] String's variables: `$table1`
17:08:19] [INFO] [AST] String's variables: `$_GET`
17:08:19] [INFO] [SCAN] Trigger Rules/Not Trigger Rules/Off Rules: 6/0/0 Vulnerabilities (7)
+-----+-----+-----+-----+-----+-----+
# | CVI      | Rule                                | Level | Target      | Source Code Content
+-----+-----+-----+-----+-----+-----+
1 | 260001   | PHP serialization                  | M-05  | /v.php:78   | $test_uns = unserialize($test);
2 | 160005   | SQLi Wordpress plugin              | H-08  | /v.php:270  | $results = $wpdb->get_results("SELECT * FROM" .$_G
3 | 160005   | SQLi Wordpress plugin              | H-08  | /v.php:273  | $results2 = $wpdb->query("SELECT * FROM ". $table1
4 | 167001   | XML external entity (XXE)          | M-05  | /v.php:81   | $data = simplexml_load_string($xml);
5 | 140003   | XSS                                | M-04  | /v.php:215  | echo $_GET[c];
6 | 181001   | Command injection                  | C-10  | /v.php:16   | system('ls'+$cmd);
7 | 170002   | LFI/RFI                            | H-07  | /v.php:61   | require_once $cmd;
+-----+-----+-----+-----+-----+-----+
17:08:19] [INFO] [INIT] Done! Consume Time:0.435732126236s
```



#introduction (the_chosen_one >> test_cases)





**KEEP
CALM
IT IS
DEMO
TIME**



XSS

- Se inyecta código JS/HTML a través de un *user input* para que el navegador lo interprete.
- Si la aplicación no valida la información que recibe y/o no la escapa antes de incluirla en la respuesta, el contenido malicioso es interpretado por el navegador.

XSS

```
<?xml version="1.0" encoding="UTF-8"?>
<cobra document="https://github.com/WhaleShark-Team/cobra">
  <name value="XSS"/>
  <language value="php"/>
  <match mode="function-param-controllable"><![CDATA[(echo|print|print_r|exit|die|printf|vprintf|trigger_error|user_error|
odbc_result_all|ovrimos_result_all|ifx_htmltbl_result)]]></match>
  <repair block="in-function"><![CDATA[(htmlspecialchars)]]></repair>
  <level value="4"/>
  <test>
    <case assert="true"><![CDATA[print_r ($_GET['test']);]]></case>
    <case assert="true"><![CDATA[
      $filename = $_POST['filename'];
      $file = fopen($filename, 'r')
      or exit("unable to open file ($filename)");
    ]]></case>
  </test>
  <solution>
    XSS - Sanitize user input if you are going to append that to HTML or JS content.
  </solution>
  <status value="on"/>
  <author name="Feei" email="feei@feei.cn"/>
</cobra>
```

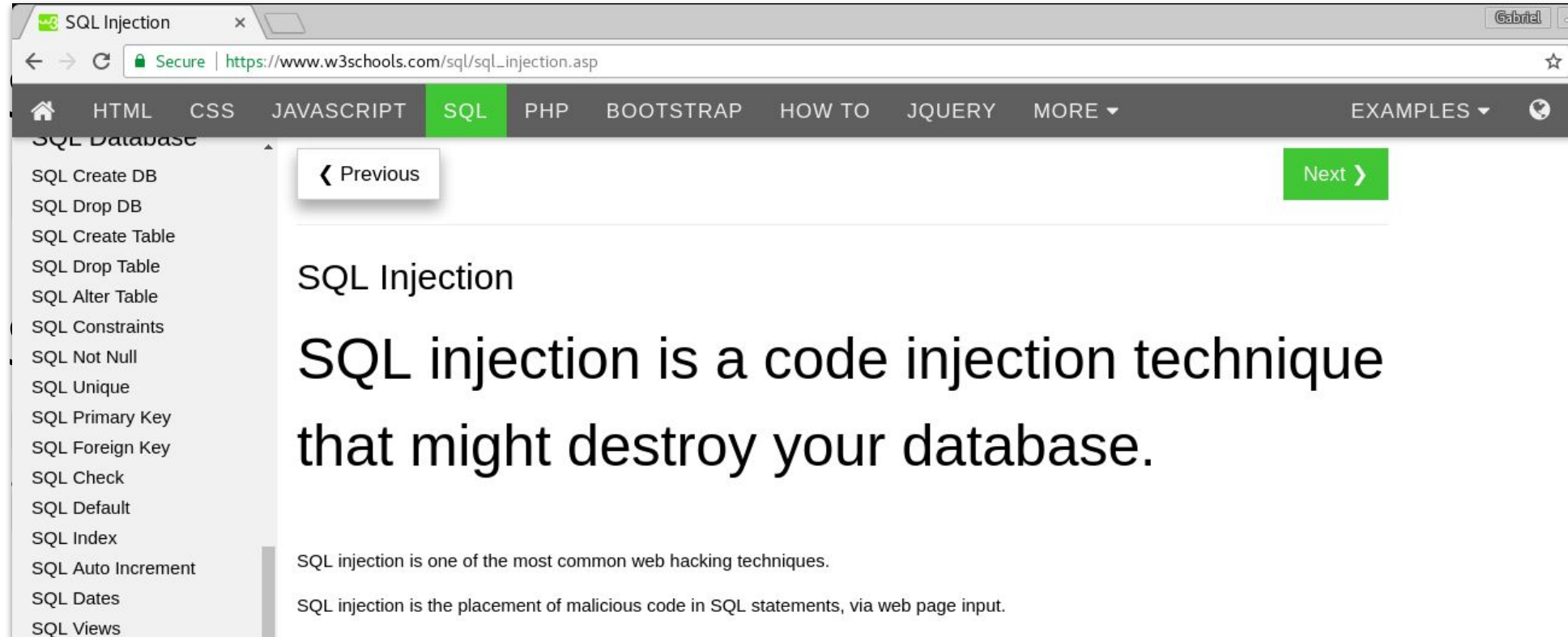
CSV Injection

- Se inyectan funciones de Excel dentro de parámetros que el usuario pueda controlar.
- Si la aplicación no valida la información que recibe, al momento que el documento sea abierto con Excel será ejecutado.

SQLi

- Se inyecta código SQL dentro de un parámetro controlado por un usuario.
- Si la aplicación no valida la información que recibe o no la escapa al momento de interpretarla, el motor la ejecutará.

SQLi



SQLi

```
<?xml version="1.0" encoding="UTF-8"?>
<cobra document="https://github.com/WhaleShark-Team/cobra">
  <name value="SQLi Wordpress plugin"/>
  <language value="php"/>
  <match mode="regex-param-controllable"><![CDATA[
    (?:\$wpdb->query|\$wpdb->get_var|\$wpdb->get_row|\$wpdb->get_col|\$wpdb->get_results|\$wpdb->replace)\(((.+)\))
  ]]></match>
  <repair block="in-current-line"><![CDATA[(\$wpdb->prepare|mysql_real_escape_string|addslashes)]]></repair>
  <test>
    <case assert="true"><![CDATA[
      $results = $wpdb->get_results( "SELECT * FROM ". $_GET["id"] . ".edn_subscriber");
    ]]></case>
    <case assert="false"><![CDATA[
      $results = $wpdb->get_results( "SELECT * FROM ". "false" . "edn_subscriber");
    ]]></case>
    <case assert="false"><![CDATA[
      $table = $_POST['table'];
      $table = mysql_real_escape_string($table);
      $results = $wpdb->get_results( "SELECT * FROM ". $table);
    ]]></case>
  </test>
  <level value="8"/>
  <solution>
    The programmer need sanitize user input before append this to SQL query.
  </solution>
  <status value="on"/>
  <author name="EzequielTBH" email="ezequieltbh@infobytesec.com"/>
</cobra>
```

SO what's next?

- Expandir la herramienta con más vulnerabilidades.
- Implementar e investigar técnicas de análisis estático (*tainted analysis*, por ejemplo)
- No tocamos cosas sencillas, como encontrar keys, algoritmos de cifrados débiles, uso de librerías inseguras, etc...
- Integrar esto en el *Ciclo de Vida de Desarrollo de Software Seguro (SDLC)*, si desarrollas plugins de Wordpress o utilizas los mismos en producción.
- Tomar mucho (mas) Fernet



#conclusions

- Es bastante simple encontrar vulnerabilidades comunes en los desarrollos de plugins. (good)
- Es realmente difícil *tracear* las variables y controlar la manipulación por parte del usuario. (sad)
- No existen muchas herramientas públicas para el análisis estático orientado a WordPress.
(good | sad)
- Haremos un merge request con los cambios a Cobra luego de OWASP Buenos Aires. (awesome)



#more_information

@EzequielTBH

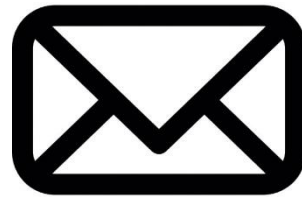
@gaaabifranco

@q3rv0

ezequieltbh @ infobytesec.com

gabrielf @ infobytesec.com

fedef @ infobytesec.com



still got
doubts?

FARADAY



"That's all Folks!"

l s b e r g[®]