Remote code execution in WordPress

By Tom Van Goethem

About

- Tom Van Goethem
- PhD Student at KU LEUVEN
- Security Researcher
- Blogger – **http://vagosec.org**
-  – **@tomvangoethem**

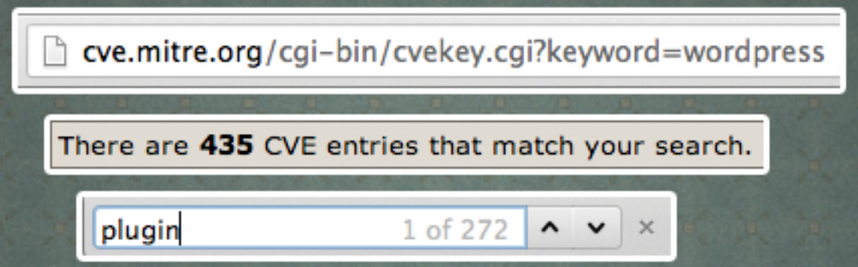
Agenda

- WordPress
- PHP Object Injection
- UTF-8 and MySQL
- Vulnerability
- Exploit
- Demo
- Disclosure timeline

WordPress

- Free and open source web blogging system and CMS
- PHP, MySQL
- Plugin & template architecture
- 60,000,000 websites
- approx. 19% of top 10mil

WordPress



- 435 vulnerabilities since 2004
- Most from plugins
- 2013: 16 vulnerabilities
- CVE-2013-4338

CVE-2013-4338

wp-includes/functions.php in WordPress before 3.6.1 does not properly determine whether data has been serialized, which allows remote attackers to execute arbitrary code by triggering erroneous PHP unserialize operations.

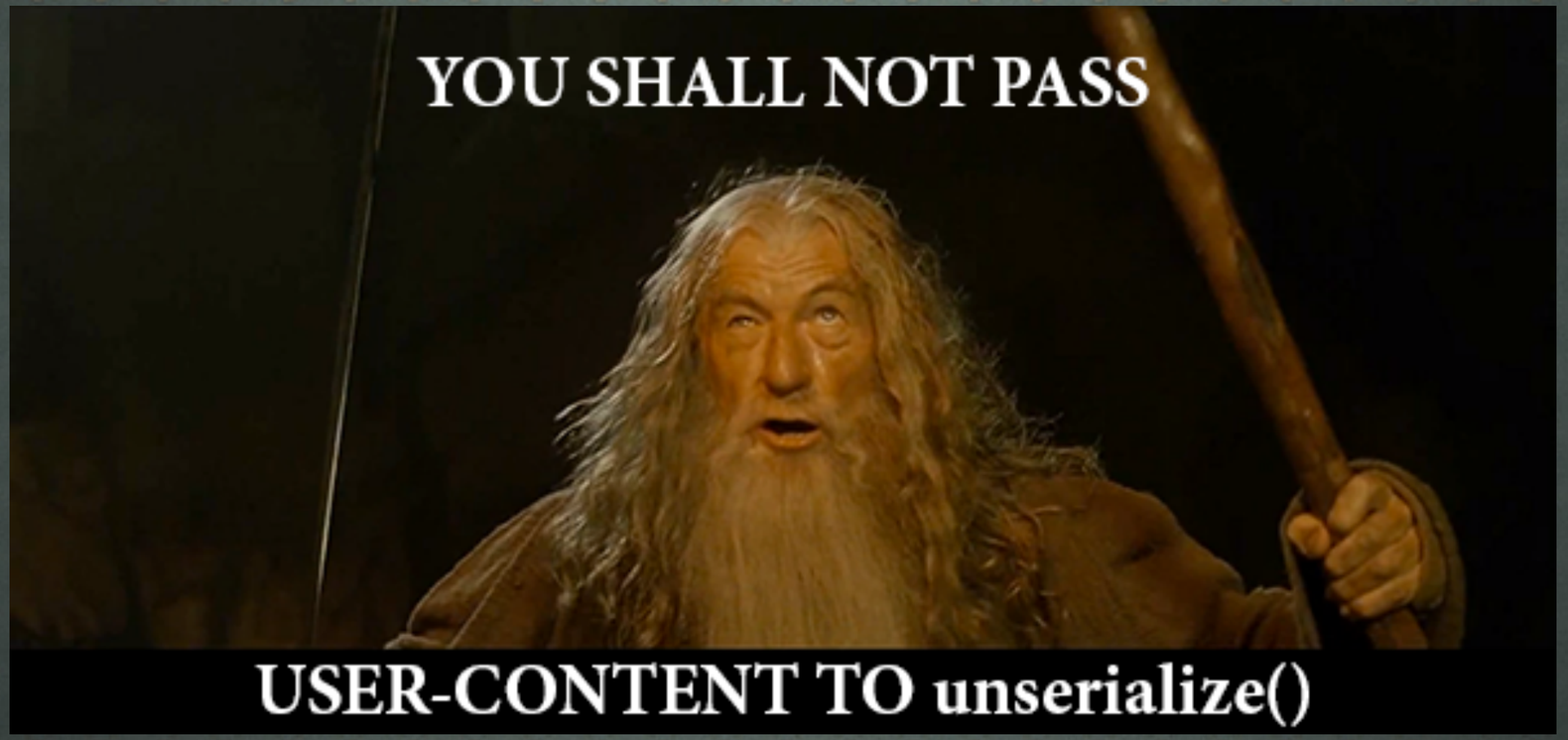
PHP Object Injection

- PHP's unserialize() can instantiate objects
- Some "magic methods" are executed on instantiation/when printed/…
- Passing user-input to PHP's unserialize() may have disastrous effects

Warning

Do not pass untrusted user input to ***unserialize()***. Unserialization can result in code being loaded and executed due to object instantiation and autoloading, and a malicious user may be able to exploit this. Use a safe, standard data interchange format such as JSON (via json_decode() and json_encode()) if you need to pass serialized data to the user.

PHP Object Injection



UTF-8

- In the beginning… there was ASCII
  - American Standard Code for Information Interchange
  - 7 bits
  - 127 characters
- I 💖 すし
- Support for a lot more characters needed

UTF-8

- Then came Unicode
  - map characters to a number ranging U+0000 → U+10FFFF
  - still requires encoding
- UTF-8
  - backward compatible with ASCII
  - 1-4 bytes long
  - I 💖 すし = U+0049 U+0020 U+1F496 U+0020 U+3059 U+3057

    I = 01001001

    💖 = 11110000 10011111 10010010 10010110

UTF-8 and MySQL

- MySQL has utf8 charset
  - All we need, right?

```
CREATE SCHEMA utf8test DEFAULT CHARACTER SET utf8;

CREATE TABLE utf8test.utf8test_table (
  utf8test_column VARCHAR(255) CHARACTER SET 'utf8' NULL)
DEFAULT CHARACTER SET = utf8;

INSERT INTO utf8test_table (utf8test_column) VALUES ('I love すし');
# Query OK, 1 row affected (0.00 sec)

INSERT INTO utf8test_table (utf8test_column) VALUES ('I 💖 すし');
# Query OK, 1 row affected, 1 warning (0.00 sec)

SHOW WARNINGS;
# Incorrect string value: '\xF0\x9F\x92\x96 \xE3...' for column
'utf8test_column' at row 1

SELECT * FROM utf8test.utf8test_table;
# +-----------------+
# | utf8test_column |
# +-----------------+
# | I love すし      |
# | I               |
# +-----------------+
```

UTF-8 and MySQL

- From MySQL Reference Manual:

> `utf8`, a UTF-8 encoding of the Unicode character set using one to three bytes per character.

- MySQL's utf8 only supports U+0000 → U+FFFF
- What happens with U+10000 → U+10FFFF?
  - Depends on character set
    - ➡ with utf8: drop character and everything that follows
  - to fully support Unicode in MySQL, use utf8mb4

UTF-8 and MySQL

Vulnerability

- WordPress user-meta data can be serialized
- user-meta?
  - first name, last name, contact info, ...
  - stored in wp_usermeta (default charset utf8)
- can be serialized?
  - normal string → normal string
  - object → serialize(object)
  - serialized string → serialize(serialized string)

Vulnerability

- When stored in DB, content is serialized
  - only if is_serialized() returns true
- When retrieved from DB, content is unserialized
  - only if is_serialized() returns true

```php
function is_serialized($data) {
   if (!is_string($data)) { return false; }
   $data = trim($data);
   if ('N;' == $data) { return true; }
   $length = strlen($data);
   if ($length < 4) { return false; }
   if (':' !== $data[1]) { return false; }
   $lastc = $data[$length-1];
   if (';' !== $lastc && '}' !== $lastc) { return false; }
   $token = $data[0];
   switch ($token) {
      case 's' :
         if ('"' !== $data[$length-2]) { return false; }
      case 'a' :
      case 'O' :
         return (bool) preg_match("/^{$token}:[0-9]+:/s", $data);
      case 'b' :
      case 'i' :
      case 'd' :
         return (bool) preg_match("/^{$token}:[0-9.E-]+;\$/", $data);
   }
   return false;
}
```

Vulnerability

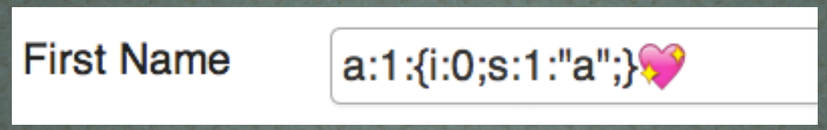
- What we need:
  - when inserted in DB, is_serialized() should return false
  - when retrieved from DB, is_serialized() should return true
- Let's put one and one together
  - append 4-byte UTF-8 character to serialized string
  - is_serialized() returns false:

```
if (';' !== $lastc && '}' !== $lastc)
  return false;
```

  - when stored in DB: last character dropped
  - when retrieved: is_serialized() returns true
  - unserialize() is called on arbitrary user-content

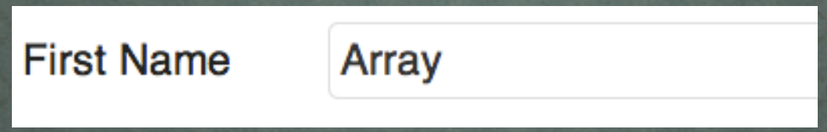
Vulnerability

- Before:



Update Profile

- After:



Vulnerability

Exploit

- Vulnerability: ✓
- Needed for a working exploit:
  - class with “useful” magic method
    - ➡ __destruct(), __toString(), __wakeup()
  - is included at right time

WHAT

MARCELLUS WALLACE

LOOKS LIKE

		YES	NO
BALD		✓	
BLACK		✓	
A BITCH			✓

```php
class Foo {
    private $final_command;

    public function setFinalCommand($command) {
        $this->final_command = $command;
    }

    public function __destruct() {
        if ($this->final_command) {
            shell_exec($this->final_command);
        }
    }
}
```

- Not found in WordPress core...

```php
$object = new Foo();
$object->setFinalCommand('echo "pwned!" > /tmp/pwned.txt');

$serialized = serialize($object);

$payload = $serialized . '💖';
```

Exploit

Plugins can extend WordPress to do almost anything you can imagine. In the directory you can find, download, rate, and comment on all the best plugins the WordPress community has to offer.

28,163 PLUGINS, 553,479,561 DOWNLOADS, AND COUNTING

- ...anything you can imagine... ☺


WHAT
DEMO ?

Disclosure Timeline

- 02/04/2013: Vulnerability discovered
- 04/04/2013: Initial report to security@wordpress.org
- 10/04/2013: Resent report
- 18/06/2013: Initial fix by WordPress
- 18/06/2013 → 27/06/2013: Provided feedback + updates on initial fix
- 21/06/2013: WordPress 3.5.2 released (fix not included)

Disclosure Timeline

- 01/08/2013: WordPress 3.6 released (fix not included)
- 06/09/2013: Second WordPress fix
- 11/09/2013: WordPress 3.6.1 released **(fix included)**
- 11/09/2013: Public disclosure of vulnerability
- 29/11/2013: Disclosure of PoC at OWASP BeNeLux
- 0./12/2013: Public disclosure of PoC

Questions?

http://vagosec.org

 – @tomvangoethem