

**TECNOLÓGICO DE ESTUDIOS SUPERIORES DE IXTAPALUCA**

MATERIA:

TALLER DE SISTEMAS OPERATIVOS

PROFESOR:

KEVIN GYOVAMI RAMIREZ VITE

ESTUDIANTES:

FLORES AGUILAR JOSE ADRIAN

PEREZ BRIONES MARIA VICTORIA

ILLESCAS ROBLEDO JUAN SEBASTIAN

GRUPO: 1801 TURNO: MATUTINO

TRABAJO:

PRACTICA GITHUB

FECHA DE ENTREGA: 5/04/2020

**INTRODUCCIÓN**

**PHP**

En equipo se seleccionó “PHP” para realizar el trabajo de esta semana y dejarlo reflejado en “GitHub”, por lo que empezamos con que el cual es un lenguaje de “scripting” escrito en los lenguajes “Perl y C”, procesador de Hipertexto utilizado principalmente en la programación de páginas Web y aplicaciones Web dinámicas. Es considerado como un lenguaje de programación Web apto para inexpertos y es compatible con HTML. Por estas razones, suele ser uno de los primeros lenguajes que aprenden los futuros programadores.

PHP considerado por algunos como un lenguaje desactualizado, muchos propietarios de páginas web siguen dependiendo de él hoy en día. Entre las principales ventajas que ofrece, se incluye el hecho de que es un lenguaje con soporte de basa de datos e integración eficiente con el protocolo de Internet. PHP ha publicado varias actualizaciones desde sus inicios y actualmente se encuentra en la versión 7. Se trata de un lenguaje con licencia de código abierto y disponible de forma gratuita.

PHP es el código de programación del lado del servidor evitando que la interpretación por parte del navegador, como ocurre en el caso de otros lenguajes de programación web muy conocidos. Por ello, PHP está incorporado en HTML (el cual no contiene información de estado) y provoca una mayor carga en el servidor que otros lenguajes de programación que solo transfieren el código fuente a un navegador web dedicado.

Las críticas normalmente se basan en su escritura débil y en la falta de opciones dentro de la gestión de errores estandarizada, aunque el hecho de que muchas páginas web importantes sigan utilizando PHP es una prueba de la popularidad de este lenguaje. No obstante, pueden aparecer problemas si los programadores utilizan versiones de PHP no actualizadas que ponen en peligro la seguridad y la estabilidad de la página web.

**PRINCIPALES VULNERABILIDADES WEB**

Lo importante que es la información acerca de nosotros que tenemos publicada, y que debemos intentar controlar, las vulnerabilidades más importantes que se pueden presentar un sitio web pueden ser:

Primer tenemos una plataforma montada sobre un sistema operativo, con su motor de base de datos, lenguajes, normalmente interpretados, certificados, etc. Ésta es la parte del servidor web en sí. Las vulnerabilidades que tendrá desde ese punto de vistas serán:

**Configuración Débil** (por defecto o mal configurado).

Suele producirse por intentar subir el servicio lo más rápido posible o por una confianza excesiva en el software utilizado, incluso por desconocimiento. Cuando se expone a Internet un sistema con su configuración por defecto, si se encuentra una vulnerabilidad en el mismo, el “exploit” por defecto funcionará. Es necesario revisar el servicio a desplegar y buscar una configuración suficientemente fuerte.

**Comunicación Insegura entre Cliente y Servidor**

Es de suma importancia que la comunicación entre cliente y servidor esté cifrada, sobre todo, cuando se trata de envíos de formularios, por ejemplo, para autentificarse en sitio web. En muchas ocasiones, se configura el sitio web para usar cifrado, con una configuración débil, permitiendo protocolos inseguros, como SSLv2 y SSLv3, o suites de cifrado vulnerables, como MD5. Esto dará una falsa sensación de seguridad, el cliente verá que hay un certificado, que el sitio web, aparentemente está cifrado, y, sin embargo, por culpa de los protocolos permitidos, podría ser relativamente fácil descifrar esta comunicación. Es por ello que es necesario revisar el estado de la calidad en la configuración de nuestros certificados.

**Software Desactualizado**

Después de desplegar el sistema, muy independientemente de que se hiciera correctamente el trabajo o no, pasado el tiempo, el software se desactualiza. Se localizan vulnerabidades y se corrigen en versiones posteriores, etc.

**Es necesario llevar un control de las versiones utilizadas, así como mantener el software actualizado, en el menor tiempo posible**, desde que se libera una nueva versión. En caso contrario, al cabo de un tiempo, nuestro sistema será vulnerable y existirán “exploits” públicos que permitirán atacarlo.

**VULNERABILIDADES DE APLICACIÓN**

Se puede decir que las vulnerabilidades son propias de la aplicación que se quiere desplegar, del código de la misma, independientemente de que la plataforma sobre la que despleguemos la aplicación esté correctamente fortificada, si la aplicación posee vulnerabilidades, corremos el riesgo de que puedan encontrarse y ser atacadas.

Entre las más comunes en una aplicación web podemos encontrar las siguientes vulnerabilidades:

**Cross-Site Scripting (XSS)**

Es una vulnerabilidad o conjunto de vulnerabilidades que permiten, utilizando los parámetros de entrada de la aplicación, modificar y añadir código a la misma. Son vulnerabilidades que se encuentran en el servidor, pero que están destinadas a atacar al cliente.

Generalmente, se necesita que el cliente sigua un enlace de la aplicación, en el que se ha modificado algún parámetro, para permitir añadir código, que se ejecutará en el navegador del cliente. Normalmente, el código inyectado será “HTML o JavaScript” y la intención será un robo de cookies del cliente, predicción del id de sesión, etc.

Esta vulnerabilidad se aprovecha de la confianza en el cliente en el sitio web: Verá que es el dominio de la aplicación y que, al seguir el enlace, llega realmente al sitio web que quería, no hay suplantación del mismo.

 Es posible que agrupemos los XSS en dos grupos:

**XSS reflejados**

El código modificado se elimina al cargar la página de nuevo. Está basado en la URL y, por tanto, al recargar la página, se elimina el mismo.

**XSS persistentes**

El código modificado queda almacenado en la web.

La mejor forma para corregir esta vulnerabilidad es filtrar y validar todas las entradas de la aplicación, sin utilizar nunca una variable que se recibe desde el cliente, confiando en que ésta tendrá un valor correcto.

Los lenguajes de programación entre ellos Php incluyen diferentes funciones que permiten filtrar el contenido de las variables, dependiendo de dónde las vayamos a utilizar.

**Cross Site Request/Reference Forgery (CSRF)**

Es evolución de XSS, donde se va a explotar la confianza en el servidor sobre el cliente. Es decir, nos haremos pasar por un cliente legítimo, utilizando datos parciales del mismo. Está presente normalmente en formularios esta vulnerabilidad, cuando se envían al servidor, es necesario asegurarse que la petición es legítima y debemos asegurarnos que el cliente ha realizado la petición, realizando los pasos previos necesarios.

Para eliminar esta vulnerabilidad la forma más común o, al menos, mitigarla, es la inclusión de tokens dinámicos, en los actuales “FrameWorks”, suelen incluirse mecanismos para añadir esta protección a los formularios, de una forma muy sencilla. Por ejemplo, Laravel (utilizado para desarrollo de aplicaciones en PHP), basta añadir una etiqueta a la plantilla del formulario, para que se añada al mismo el token “anti-csrf”.

Los nombres más comunes de los diferentes XSS, que dependen del lugar donde se consiga inyectar el código deseado son:

* DOM Cross Site Scripting (DOM XSS)
* Cross Site Flashing (XSF)
* Cross Frame Scripting (XFS)
* Cross Zone Scripting (XZS)
* Cross Agent Scripting (XAS)
* Cross Referer Scripting (XRS)
* …

**SQL INJECION**

Los ataques XSS son peligrosos porque pueden provocar un robo de sesión, los “SQLi” son aún más, ya que permiten acceder y manipular la BBDD. La idea es modificar las consultas que hace la aplicación a la base de datos, aprovechando las entradas de usuario a la aplicación.

La estructura de una consulta SQL típica.

SELECT [Columnas] FROM [Tabla] WHERE [Condiciones]

Donde primero se seleccionan las columnas que queremos mostrar, después se indica la BBDD y la tabla de la que queremos obtener las columnas y, por último, las condiciones para filtrar esos datos. Cuando se van a realizar consultas, se realizan con un usuario de BBDD. Este usuario tendrá diferentes permisos.

No debemos perder de vista los motores de BBDD, tienen sus propias tablas para almacenar metadatos, por ejemplo, las bases de datos que hay, qué tablas tiene cada BBDD, las columnas y tipo de cada una, información de usuarios, etc. Debemos tener presente cuando se puede hacer un SQL injection, se hará con los permisos que disponga el usuario con el que se ejecuta.

Los diferentes ataques de SQL injection, dependiendo de cómo se consiga inyectar y modificar la consulta, para obtener los datos que se desean.

* Bypass de un login mediante SQL Injection
* SQL Injection (Mediante UNION)
* Serialized SQL Injection
* Boolean-Base SQL Injection
* Time-Base SQL Injection
* Heavy-Queries SQL Injection
* Stack-Queries

Para nuestra práctica utilizaremos “inyección de datos” con el siguiente código, donde:

La manipulación arbitraria del contenido de un objeto que será retirado de su serie y posteriormente a través del uso de la función “unserialize()” de PHP. Este tipo de vulnerabilidades suelen requerir una mayor complejidad a la hora de su explotación, por lo que tanto la explotabilidad como el impacto asociado puede ser malinterpretado si no se realiza un análisis en profundidad.

Código 1

<?php

// Exercise - 1

class login {

public $username = "X-C3LL";

public $password = "Insanity";

public $role = "MUGGLE";

}

$one = new login();

$a = serialize($one);

echo "Example of an object:\n$a\n\n";

echo "FLAG: \n";

$test = unserialize($argv[1]);

$check = $test->role - 1337;

if ($check == "ADMIN") {

$flag = file\_get\_contents("flag.txt");

echo $flag;

} else {

echo "No flag for you!! Better luck next time!\n";

}

?>

Código 2

<?php

// Exercise - 2

class File {

public function flag() {

$this->innocent();

}

public function innocent() {

echo "Aquí no pasa nada :D\n";

}

}

class GiveFlag extends File {

public $offset = 23;

public function innocent() {

$stuff = fopen("flag.txt", "r");

fseek($stuff, $this->offset);

print fread($stuff, filesize("flag.txt"));

}

}

class entry {

public function \_\_destruct(){

$this->awesome->flag();

}

}

unserialize($argv[1]);

?>

Otra técnica que se utilizo es **“ataque fishing”**

Método que una o varias personas malvadas duplica una web a la que accedemos mediante usuario y contraseña, y hace que nuestras credenciales en lugar de llegar a la original, queden almacenadas en su servidor, como ejemplo usaremos facebook.

**Facebook:**

Donde en cualquier espacio en blanco damos clic con el botón derecho del ratón para obtener el menú contextual y seleccionar la opción “Ver código fuente de la página”. Lo que nos abrirá una nueva pestaña que nos mostrará dicho código, el cual seleccionaremos y lo copiaremos a un documento nuevo del bloc de notas al que llamaremos “index.html”.

Luego en el bloc de notas, usaremos el comando Ctrl+B para buscar la cadena “action=” dentro del extenso código, esta “action=” apunta hacia una web donde se nos requerirá un inicio de sesión o login; donde empezamos a trabajar. Sustituiremos esa dirección a la que apunta por “next.php”, y si leemos inmediatamente antes de nuestro “action=” vemos que el código especifica el método por el que se enviará la información de dicho login, que es ‘post’.

También deberemos cambiar el método ‘post’ por el método ‘get’ (otro día nos paramos a explicar la diferencia entre estos dos métodos de envío de datos) para que quede tal y como muestra la imagen siguiente, y sólo faltará guardar el documento como index.html.

Ahora crearemos el script al que hemos redirigido la petición de “login”, para esto abriremos un nuevo documento en el blog de notas que llamaremos next.php, y en el que copiaremos el siguiente código:

Después crearemos un segundo archivo con el bloc de notas que llamaremos yaeresmio.txt y que cerraremos dejándolo en blanco.

Siguiente paso sería subir estos tres archivos de texto a algún dominio o subdominio de algún hosting que tengamos (obviamente, este hosting debe proveer servicios php), además de contar con un servicio de redirección segura que oculte nuestra ip (nunca está de más prevenir…). El proceso es simple ya que hoy día hay muchos servicios hosting que ofrecen un plan básico de forma gratuita, que es lo que necesitamos en este momento, para ello nos registramos e iniciamos el proceso de creación de cuenta seleccionando el subdominio (si elegimos un plan gratuito no vamos a pagar ahora por un dominio, ¿no?) y la contraseña que usaremos.

No olvidar que el hosting que seleccionemos provea servicios php para que nuestro script se ejecute sin problemas. Y dicho esto nos dirigimos al apartado de acceso FTP para subir nuestros archivos a la red. Usando el propio gestor de archivos del hosting que hemos encontrado hemos subido con éxito nuestros archivos.

Vamos a entrar a nuestra dirección web para ver cómo verán nuestras víctimas la página que hemos copiado, comprobamos que por ser una página copiada vamos a dejar estos símbolos tal cual. Por lo que lo único que queda es intentar loguearnos en “nuestro Facebook”. Usaremos como credenciales la dirección de correo electrónico correo@deprueba.es y la contraseña “contraseña de prueba”.

Al hacer clic en ‘Entrar’, se nos redirige a la página auténtica de Facebook porque “parece ser que hemos escrito algo mal y debemos volver a introducir nuestros datos”, pero si en este momento nos dirigimos a nuestro hosting, y comprobamos el contenido del archivo “yaeresmio.txt” vemos esto que aparece. El correo y contraseña que hemos usado para nuestro ejemplo se ha almacenado en el archivo tal y como esperábamos , por lo que nuestra web de phishing está lista para funcionar.

Llegados a este punto sólo nos quedaría crear una cuenta de correo que usaríamos para captar posibles víctimas, en la que deberá aparecer la palabra Facebook para darle un mínimo de credibilidad (aunque no lo creáis hay gente que responde a direcciones como Facebook@hotmail.com o similares).

Nosotros solo utilizaremos el siguiente código:

pishing

<?php

$handle = fopen("yaeresmio.txt", "a");

foreach($\_GET as $variable => $value) {

fwrite($handle, $variable);

fwrite($handle, "=");

fwrite($handle, $value);

fwrite($handle, "\r\n");

}

frité($ande, "\r\n");

fclose($handle);

exit

?>

La tercera es “**detectar errores de Sql inyección**”, con “php-reaper” que es una herramienta PHP para escanear código ADOdb para Inyecciones SQL.

Lo ideal es detectar lo antes posible o cuando el código esté en fase de desarrollo y no en producción, lo que implica costos más altos “Ejecutar [PHP-Reaper](https://github.com/emanuil/php-reaper)” consume mucho menos tiempo que ejecutar un escáner de seguridad automatizado completo en su aplicación. Posiblemente el escáner de seguridad web no encuentre todas las posibles vulnerabilidades de Inyecciones de SQL debido a que es difícil de alcanzar el código de la “IU” (o necesita establecer condiciones extra”).

PHP-Reaper” normalmente es rápido e identifica la línea exacta donde se encuentra el problema, escaneando todo el código fuente de PHP ADOdb.

Código usado:

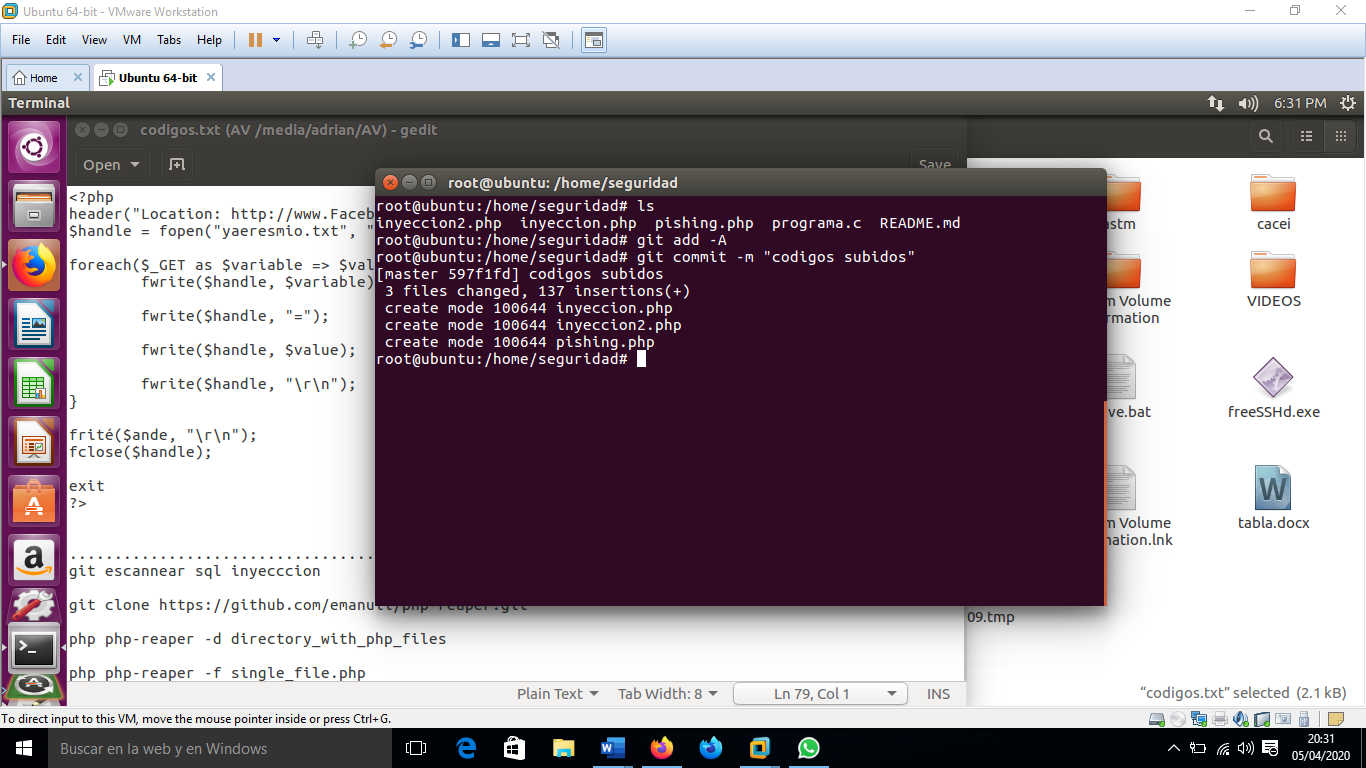
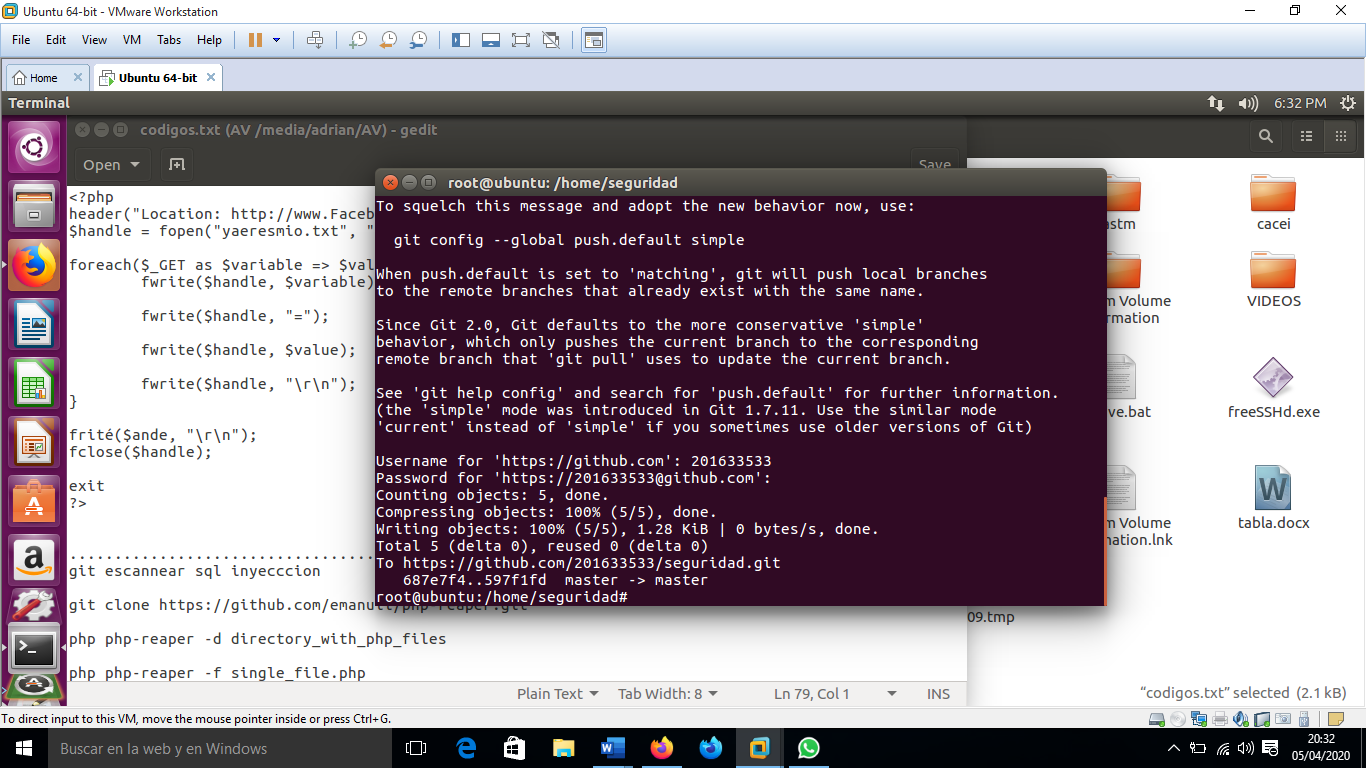
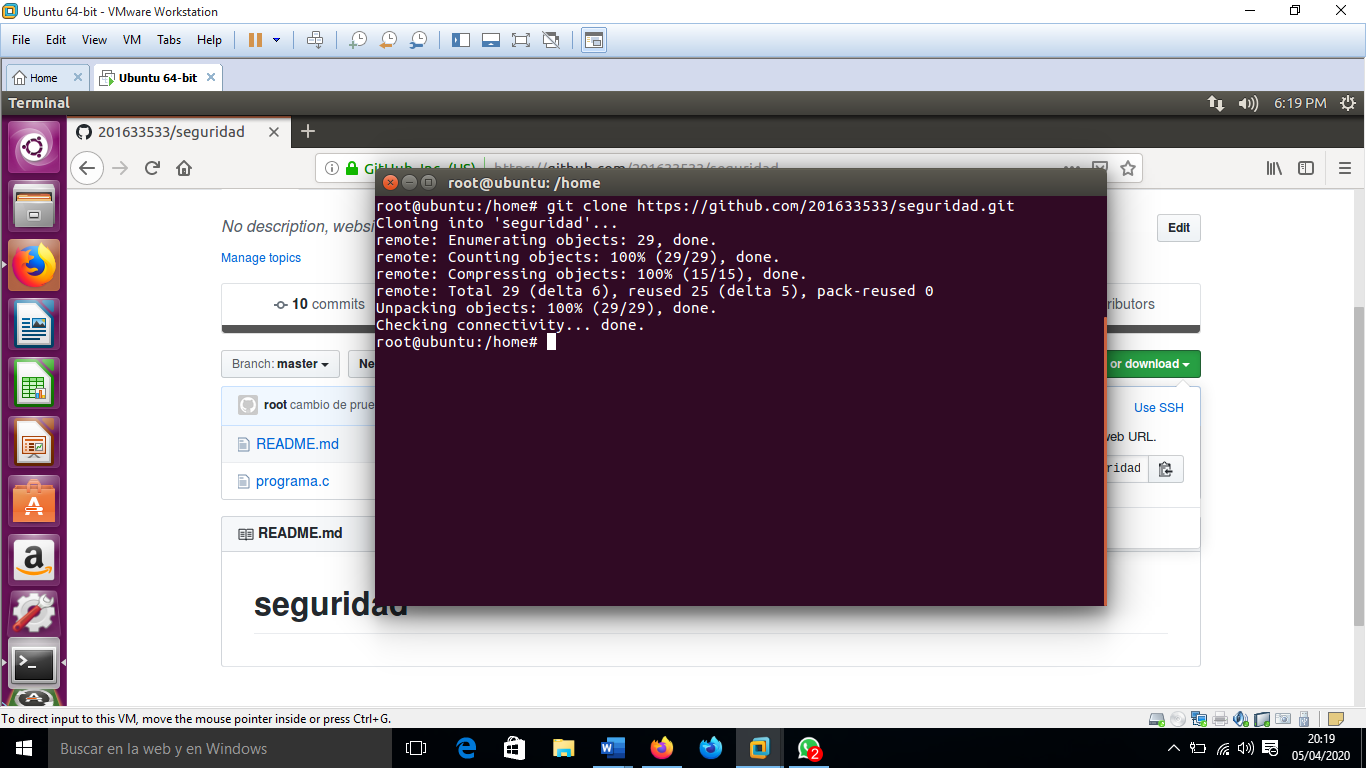
git escannear sql inyecccion

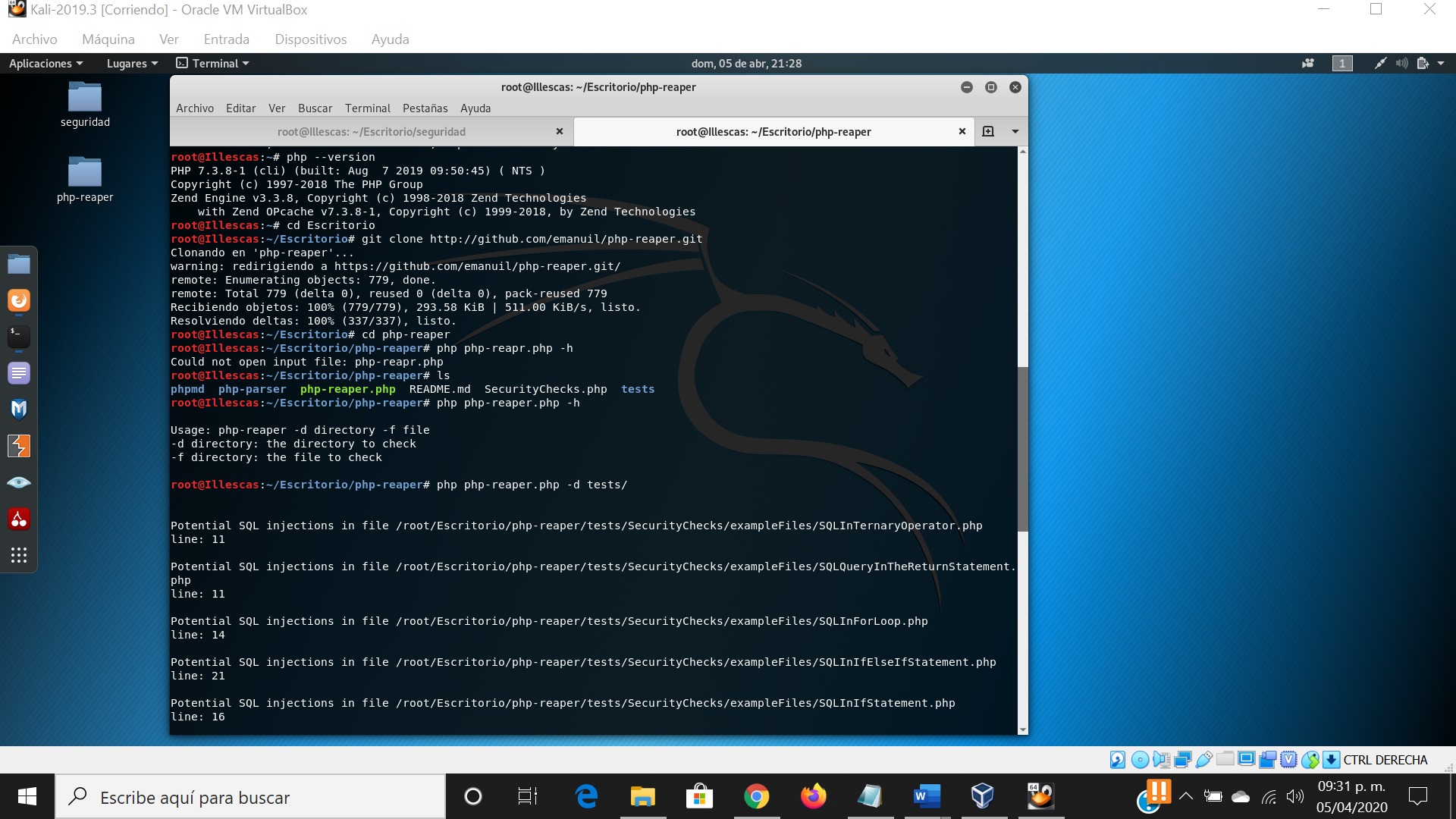
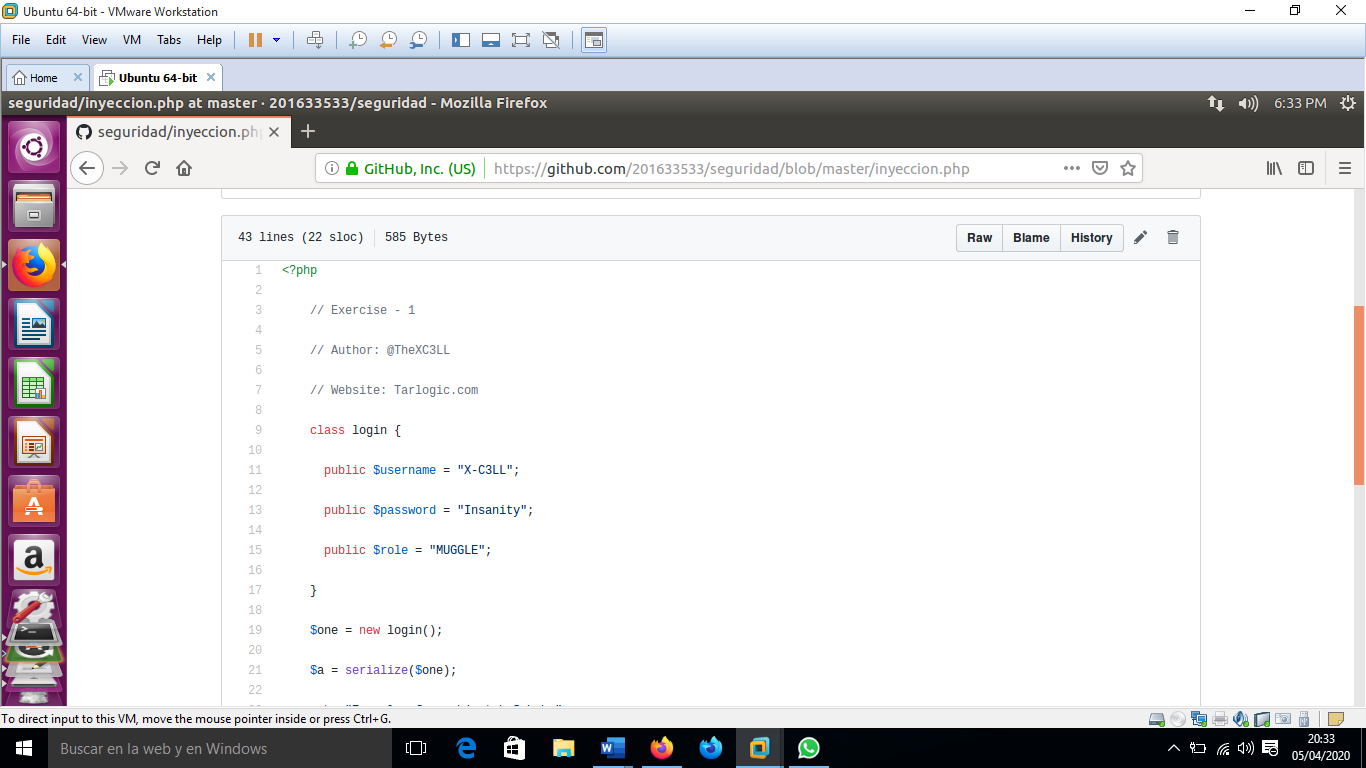
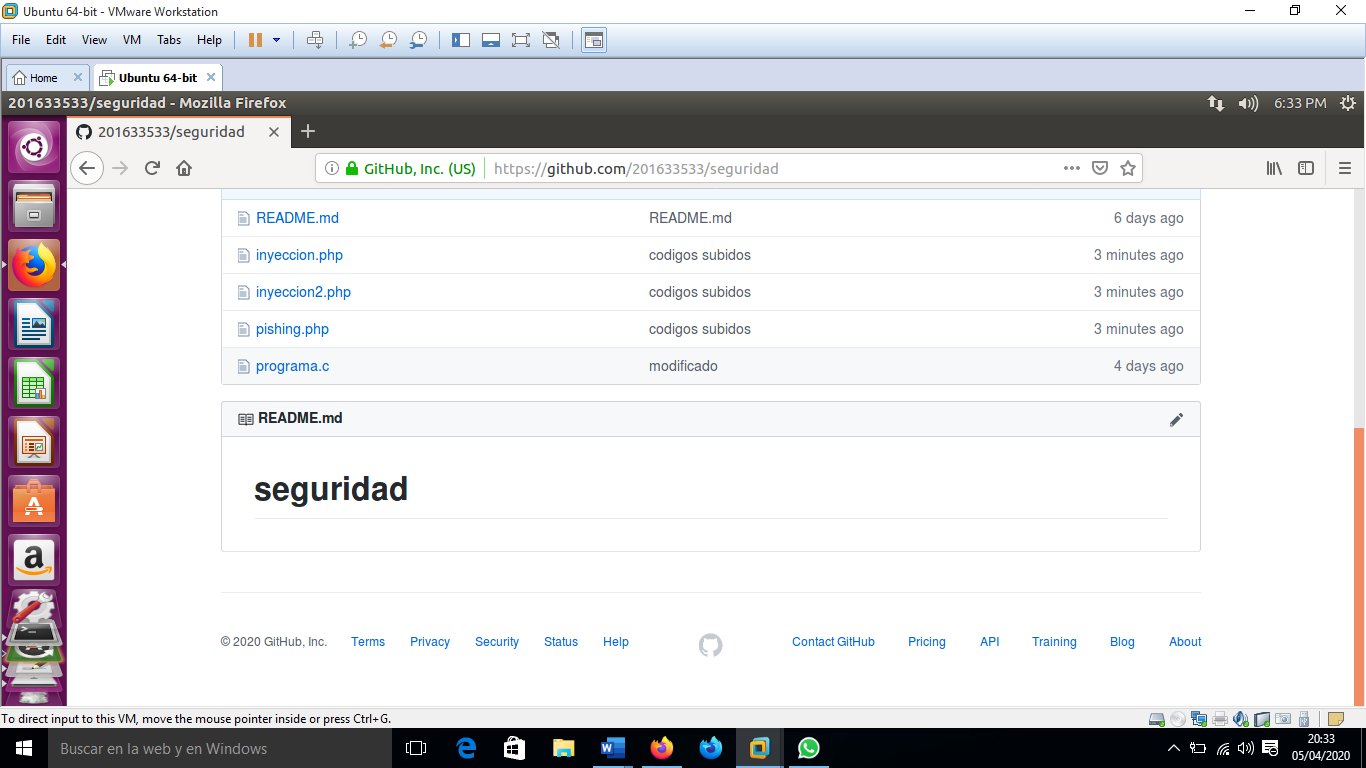
git clone https://github.com/emanuil/php-reaper.git

php php-reaper -d directory\_with\_php\_files

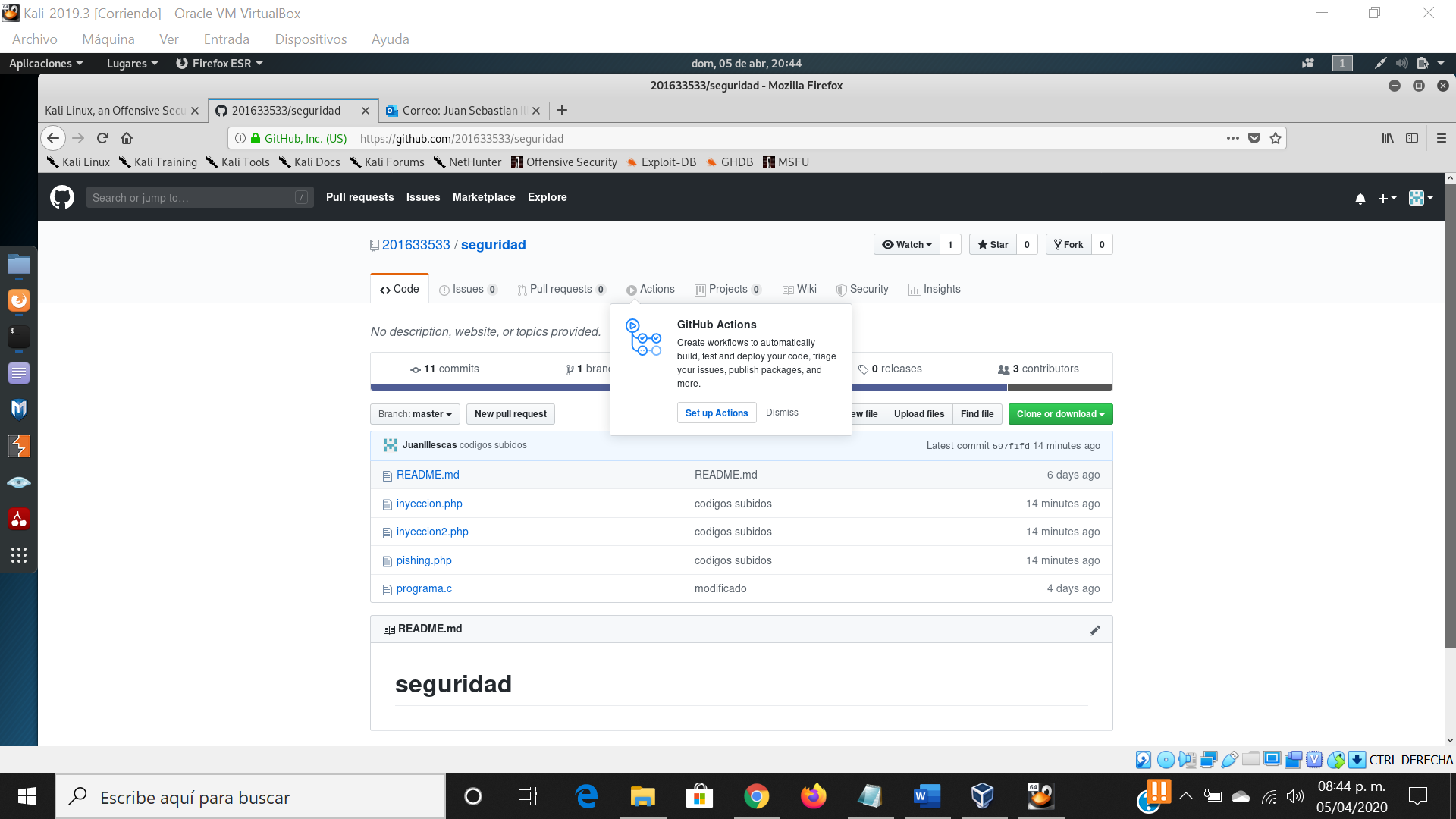
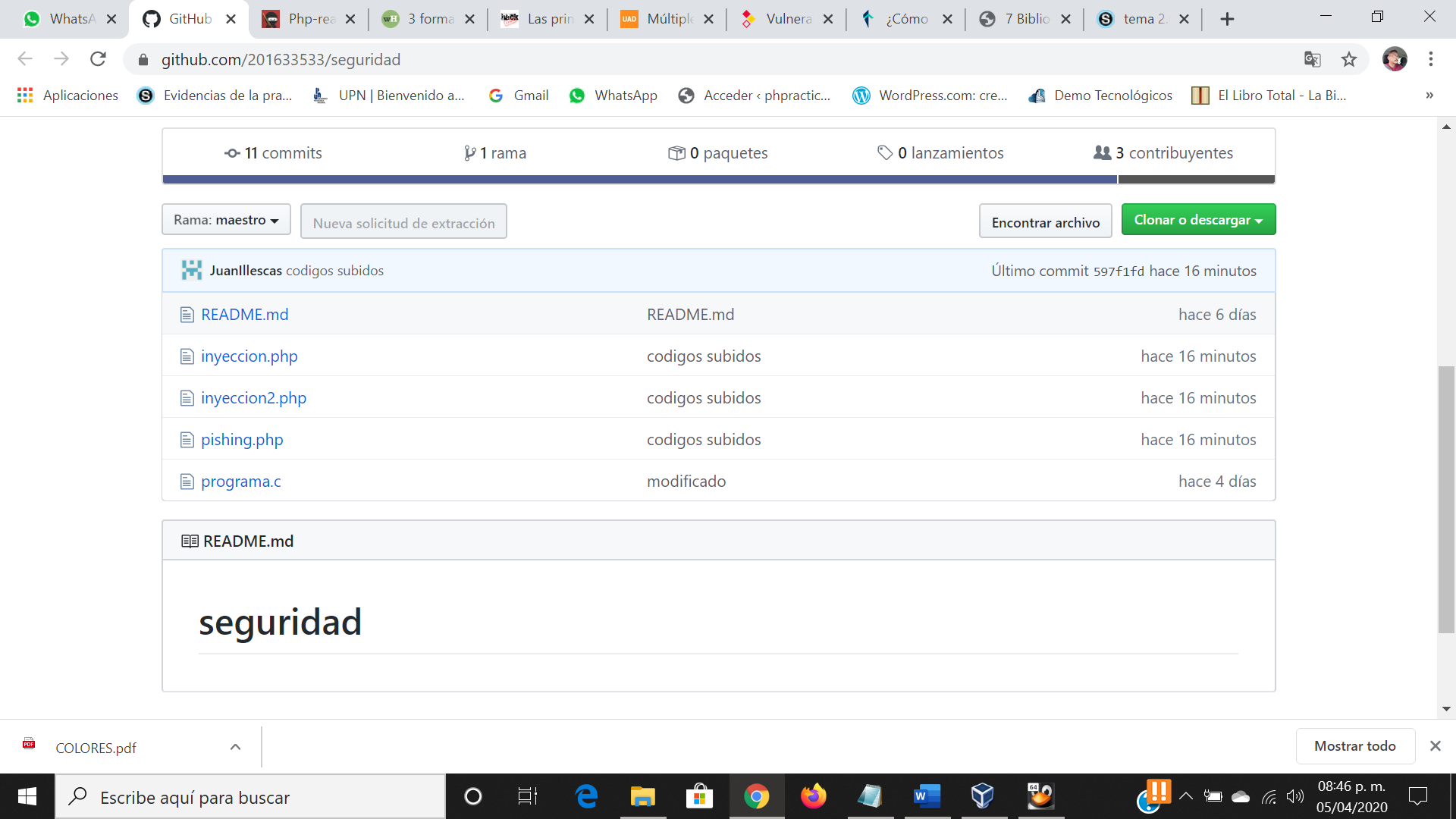
php php-reaper -f single\_file.php

**Códigos subidos a git**







Conclusiones:

El conocimiento de las diferentes vulnerabilidades de Php en la programación de una página web nos ofrece un mundo de mejoras dentro de este campo, el uso constante de “git escannear sql inyecccion” nos apoyará en una mejor creación si lo utilizamos desde los primeros pasos de nuestra web.

Por otro lado, el manejar las vulnerabilidades de fishing e inyección SQL, deja un parámetro más concreto de donde debemos mejorar nuestras páginas web entre otras aplicaciones, aunque no son las únicas vulnerabilidades que encontraremos, el tener en mente estas que son básicas, nos permitirá estar al pendiente de las nuevas que se presenten.

**La creciente vulnerabilidad de PHP,** el lenguaje de programación utilizado en más del 80% de los sitios web es cada vez más, Es por ello que la seguridad en aplicaciones Web involucra principalmente al desarrollador, aunque con gran frecuencia se encuentran defectos que pueden ser aprovechados por atacantes en las tecnologías en que se basan los sistemas web (Sistemas Operativos, Servidores Web, Servidor de Base de Datos, etc.) la atención principal debe dirigirse a los defectos propios al desarrollo nuestras aplicaciones, para saber cómo corregirlos y así no estar tan vulnerables contra ataques .

Referencias de consulta:

Stig Saether, Bakken. Alexander Aulbach, Egon Schmid, Jim Winstead, Lars Torben Wilson. (2001). Manual de PHP Manual de PHP. Editado por Rafael Martínez. Recuperado el 4 de abril del 2020 desde: <https://www.php.net/manual/es/index.php>

Joyce Park. Wiley Pub. (2002). PHP Bible Tim Converse. Editorial Php. Madrid España. Recuperado el 3 de abril del 2020 desde: <https://books.google.com.mx/books/about/PHP_Bible.html?id=UHHO7HRtL4YC&redir_esc=y>