By: Ing. Samuel Sabogal Pardo

UN WHITE HATS

1)

En este punto, se está verificando que tan fuerte es el auditor XSS de Chrome (XSS, que significa Cross Site Scripting, es un tipo de inyección de código, en el que se inyecta generalmente código javascript. El auditor XSS de Chrome, intenta prevenir estos ataques. Chrome probablemente tiene el auditor más fuerte de los 3 browsers más usados)

Ingrese a la dirección

http://www.samuelsabogalpardo.com/m1/inject.php

(Nota: si bien esta página está hecha en php, esta vulnerabilidad no tiene nada que ver con php, y no esta relacionada al lenguaje de programación en si)

Verá que dice:

Hola!
Nombre:
apellido:

Esta página, toma dos variables enviadas en la URL (Esta forma de enviar las variables se conoce como método GET)
Por ejemplo

?&nombre=Samuel&apellido=Sabogal

Envía la variable nombre con valor Samuel, y la variable apellido con valor Sabogal. La página lo que hace es tomar estas variables, y utilizarlas para imprimir:

Hola!
Nombre: Samuel
apellido: Sabogal

Esta página, imprime datos ingresados por el usuario, sin verificar que puedan tener código malicioso, lo cual es un error de seguridad muy común en las aplicaciones. Chrome en su buena labor, intenta ayudar al desarrollador y cubrir sus defectos como programador, por lo que tiene un auditor XSS, que sin embargo, puede ser engañado.

Su tarea es engañarlo, para mostrar que no nos podemos confiar nunca.

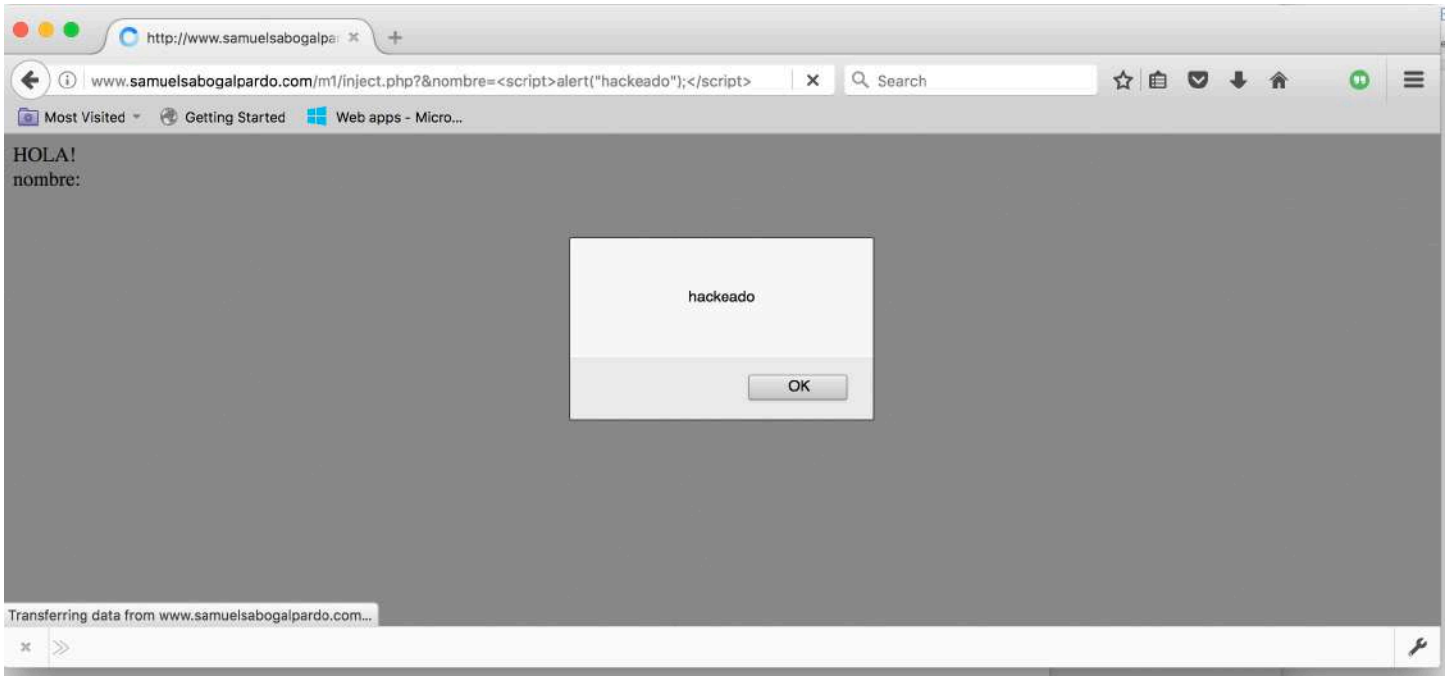
Si a una de las variables, le asigna un script de javascript como el siguiente (que simplemente dispara un mensaje que dice “hackeado”) :

```
<script>alert("hackeado");</script>
```

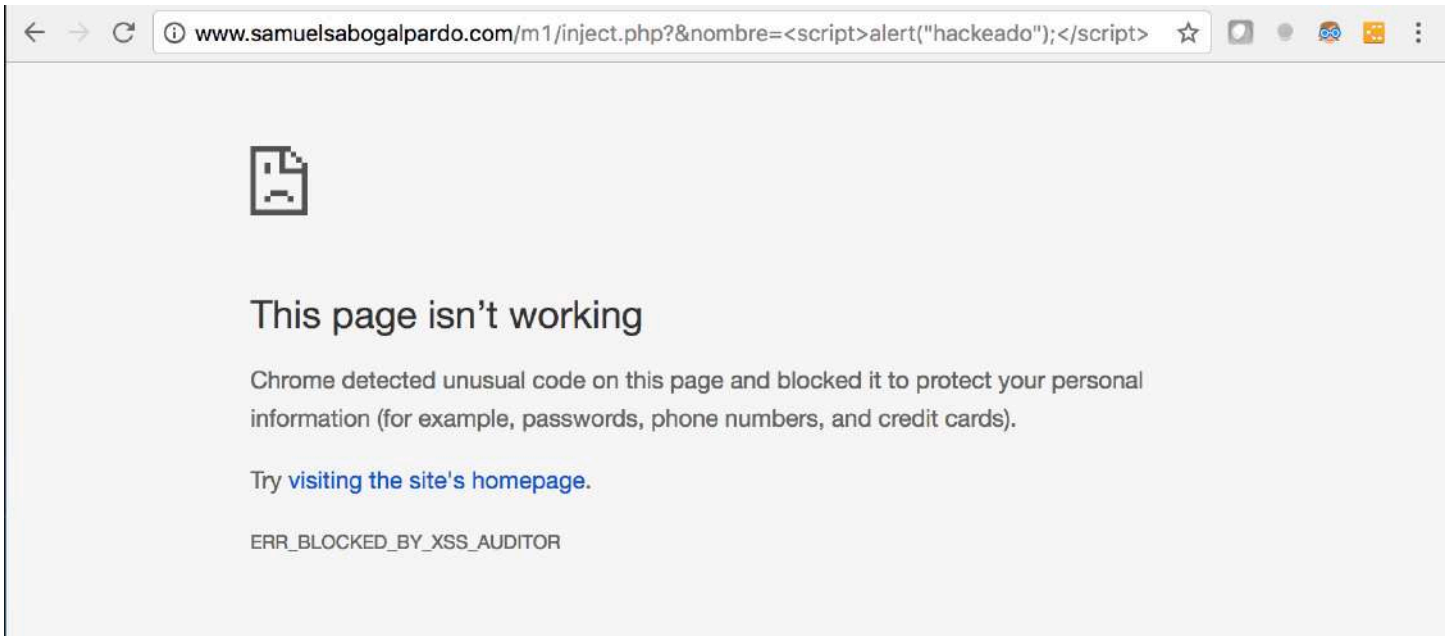
De la siguiente manera:

http://www.samuelsabogalpardo.com/m1/inject.php?&nombre=<script>alert("hackeado");</script>

En Firefox, vemos que si se dispara:



En cambio en Chrome vemos lo siguiente:

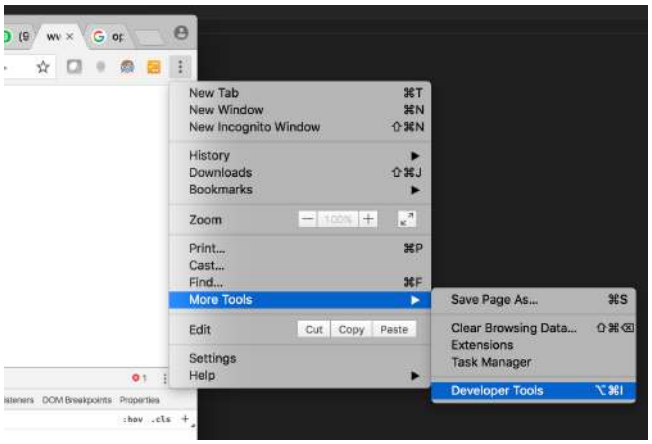


Que indica que Chrome detecto código malicioso que se podría utilizado para robar información personal.

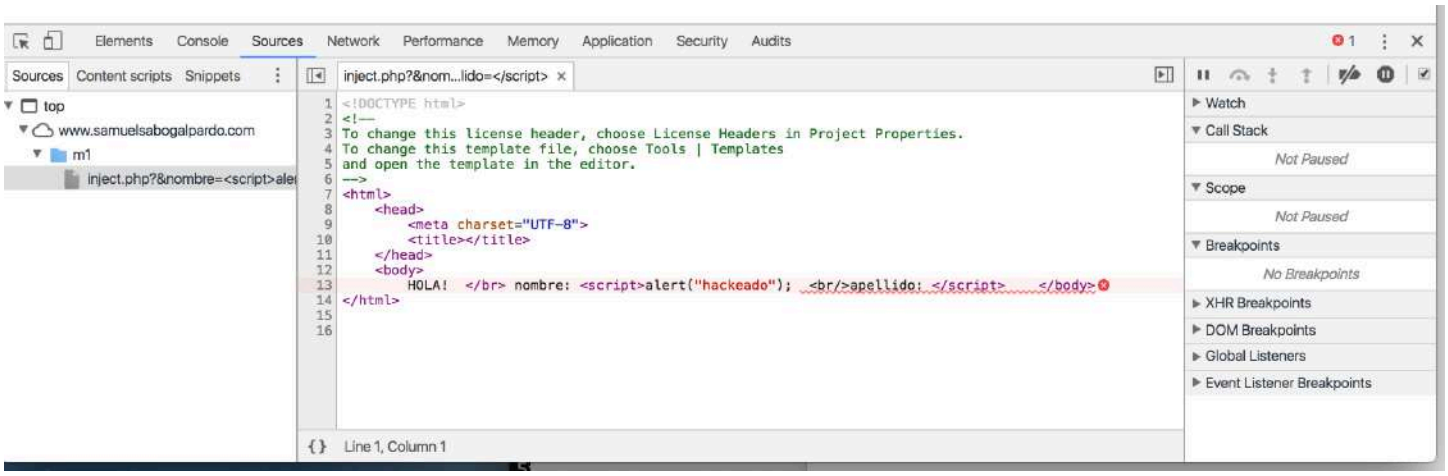
A ud se le ocurre partir el script en dos, y mandar una primera parte en la variable &nombre y otra en la variable &apellido, para probar si puede engañar al auditor, así:

http://www.samuelsabogalpardo.com/m1/inject.php?&nombre=**<script>alert("hackeado");**&apellido=</script>

Se da cuenta, que el Auditor no detecto un ataque, sin embargo tampoco tuvo éxito en la inyección. Por lo que abre las herramientas de desarrollador de Chrome ubicadas en:



Hace click en sources, y abre el código fuente de la página:



Puede ver que su script se imprimió, pero fue dañado porque le queda en medio parte del html original de la página. Si se fija, el pedazo que daña el script es este:
`<br/>apellido:`

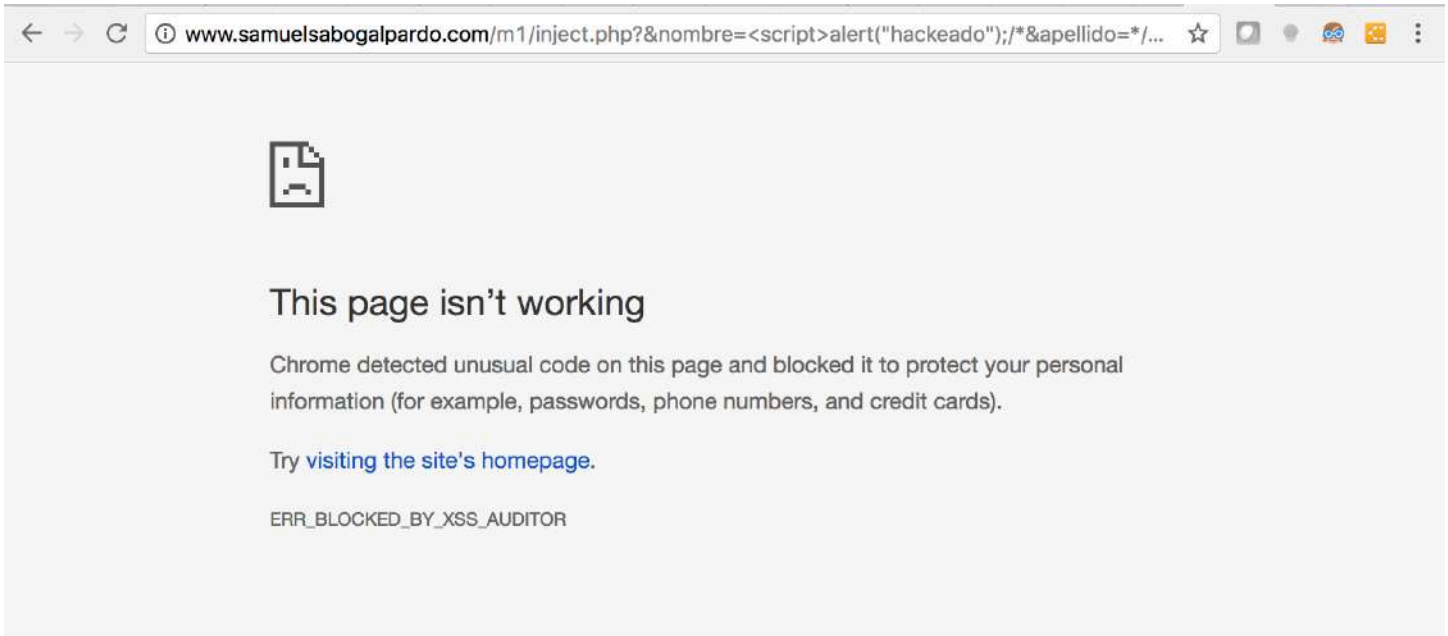
A ud se le ocurre, mandar el script abriendo un comentario así:
/*
al final de la parte de código que inyectó en la primera variable, y cerrarlo al principio del código que inyecta en la segunda variable, con la esperanza, de que el pedazo que daña el código, quede como un comentario así (y por lo tanto no dañe el código):

```
<script>alert("hackeado");  /* <br/>apellido: */ </script>
```

Por lo tanto ud inyecta:

http://www.samuelsabogalpardo.com/m1/inject.php?&nombre=<script>alert("hackeado");/*&apellido=*/</script>

Sin embargo, el Auditor de Chrome no es tan fácil de engañar, y ud se encuentra nuevamente con la pantalla:



Su tarea es demostrar que se puede engañar.
Pista: en javascript, existe una función llamada void(""), que no hace nada, lo cual es la respuesta a este punto prácticamente (solo por ser la primera maratón)

Debe responder con: la url con la variable que contenga el código inyectable, con el que logró la inyección en Chrome.
Ejemplo :
http://www.samuelsabogalpardo.com/m1/inject.php?XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
2)

Ud está investigando un fraude desgarrador en el sistema de salud. De algún
modo logro obtener una copia de la base de datos de la aplicación donde esta
la evidencia requerida. En la base de datos se encuentran almacenados los hashes
de los passwords, pero no tienen un SALT, es decir, el hash de cada password fue
obtenido directamente a partir del password. Sospecha que el algoritmo de hash
usado fue SHA1. Ud cuenta con un diccionario con todas las palabras del idioma (words.txt (solo son palabras en inglés, pero todos los passwords de esta maratón son en inglés. En la vida real puede tener un diccionario con todas las palabras del idioma, diferentes combinaciones de estas etc), también se encuentra una muestra de cómo leer un archivo en java, por si alguien ha visto muy poco java y no sabe cómo leer un archivo).
Sabe que en la aplicación en cuestión, para las contraseñas se exige poner un número al menos.
Le parece probable que la constraseña solo sea una palabra con un número al final.
El usuario que ve en la base de datos es:

"Bob_Fraud"

Y la representación hexadecimal del hash del password es:

2c4724d348726e9329e84d8ccb4c1ec20981bac9

Encuentre la contraseña que corresponde a ese hash

NOTA:
Para compilar el .java en consola, ubíquese en la carpeta donde se encuentra
el archivo .java y corra:

javac HashSample.java

Ahora, para ejecutarlo

java -cp . HashSample

Puede usar netbeans, eclipse o en sí el lenguaje de programación que prefiera. En la carpeta javasample, se encuentra un ejemplo de cómo generar un sha1 un hexadecimal en java.

Debe responder con: El password

3)

Nota: Guarde los archivos en la carpeta estudiante/documents para que los pueda ejecutar, de lo contrario no tendrá permisos. Cuando lo ejecute, el antivirus le dará una alerta de “Download Insight”, Indique “Allow this file”

Ejecute el programaX.exe en consola (cmd). Para correrlo solo se tiene que ubicarse en la misma carpeta desde cmd, y escribir:

programaX

Suponga que este programa es accedido remotamente, por lo tanto no puede abrir el compilado en un editor hexadecimal. Si quiere inténtelo pero probablemente tampoco tenga éxito porque este código está ofuscado (Igual sigue siendo un muy mala práctica dejar un password en el código así este ofuscado)

Cuando lo corra esta le pedirá un password. El cual ud no tiene, por lo tanto intentara hacer un ataque de Buffer Overflow, al cual el programaX, es vulnerable, porque está escrito en C, un lenguaje particularmente vulnerable (es posible hacer una aplicación segura en C, pero es considerablemente más difícil que en lenguajes más recientes).

La vulnerabilidad de Buffer Overflow consiste básicamente, en que los arreglos en C reservan su tamaño en memoria cuando son creados, pero cuando uno los está llenando con datos, no hay nada que le impida que siga rellenando la memoria aún así si ya se pasó del espacio correspondiente al arreglo, por ejemplo, si tenemos en memoria el arreglo de caracteres **char v[5];** y otro arreglo de caracteres **char p[5];**

La mayoría de los compiladores de C, los ubicarían contiguamente en el stack (que es la parte de la memoria donde se almacenan las variables locales de un programa. El heap es donde se almacena la memoria dinámica, por ejemplo lo que se crea con new. Y la estática guarda las variables globales. Adicionalmente existe un pedazo que almacena las instrucciones de código ya compilado, como lo muestra la siguiente figura:

El stack crece hacia arriba apilando variables locales. La dinámica crece hacia abajo. Cuando se encuentran el heap y el stack, habría un OutOfMemoryException) asi:

p[0]
p[1]
p[2]
p[3]
p[4]
v[0]
v[1]
v[2]
v[3]
v[4]

Por lo tanto, si el arreglo p es leido, y se le ingresan más de 5 caracteres, el apuntador seguría avanzando y escribiría encima del arreglo v (esto es el Buffer Overflow en si). Por ejemplo, si ingresamos “xxxxxxx”, quedaría asi:

p[0] = x
p[1] = x
p[2] = x
p[3] = x
p[4] = x
v[0] = x
v[1] = x
v[2] = x

v[3]
v[4]

Si ingresamos, “xxxxxxxxxx”, todas las posiciones de los dos arreglos quedarían con el mismo valor x. Y por ejemplo si p, era un password que se había leído de una base de datos y se tenía almacenado en ese arreglo para después ser comparado con v, cuando el programa los compare, va a identificar que son iguales, puesto que los hicimos iguales rellenándolos con el mismo valor por medio de un Buffer Overflow.

Si ingresa un texto cualquiera en el programaX, le responderá con Wrong Password. Pero si ingresa por ejemplo:

xx

Notará que le responderá con Correcto Password, y le entregará un código secreto. Este código secreto es un número de 4 digitos. Lo malo es que ud hizo un overflow tan grande que también sobre escribió el código secreto, por lo tanto solo verá xxxx. En los Bufferoverflow, que pesar de ser algo sencillo conceptualmente, en un programa de verdad son toda una “ciencia” y hay muchas técnicas para ejecutarlos. Un script de Windows, o un script de Shell, es una de ellas, la cual usará para atacar el **programaX**.

Como probablemente en este punto no muchos saben cómo codificar un .bat en Windows, tomen esté código de ejemplo, y escriban en un .bat (lo puede crear con **echo.>myBat.bat** , o también puede crear un .txt y cambiarle la extensión a .bat, y después editarlo en un bloc de notas)

```
@echo off
setlocal ENABLEDELAYEDEXPANSION

FOR /L %%s IN (1,1,200) DO (
  SET myinput=!myinput!v
)
```

El código es un ciclo, que con la línea `SET myinput=!myinput!v` hace crecer un string, como en java lo haría : String myInput = myInput+”v”;

(Nota aparte: en java hacer eso es ineficiente, porque los Strings son objetos inmutables, por lo tanto cada vez que hace una concatenación con +, se crea un nuevo objeto. Para cosas asi puede usar más bien algo como StringBuilder)

Ahora, si quiere llamar un programa, después pasarle un input, y además escribir la respuesta en un archivo, puede usar:

```
echo !myinput!|programaX >> temp.txt
```

Nota: Si ud quisiera ser más elegante, modificaría el batch para que leyera la respuesta del programa y verificara si es un número de 4 dígitos en vez de escribir en un archivo, pero, 1. En la vida real generalmente ud guarda todas las respuestas porque son más complejas y después las analiza en el archivo. 2. Probablemente ud no sabe cómo modificar el batch para que haga eso en este momento.

Debe responder con: El código secreto de 4 dígitos, concatenado al input, por ejemplo algo asi:
1234eeeeeeee

Nota: El código secreto NO es el password del programa (para este punto a ud no le interesa el password). Verifique que el input para el buffer overflow sea el correcto ingresándolo al programa manualmente, y le debe imprimir el código secreto de 4 dígitos

4)
En el programaY.exe, se pide un password, en este programa, el password está quemado en el código, por lo tanto el compilado lo tiene, y si lo abre en un editor hexadecimal, lo puede encontrar.

Pista: el password es una frase de 23 letras minúsculas (sin espacios ni caracteres especiales). Asegúrese que sea el correcto, corriendo el programa y probándolo.

Debe responder con: El password
Nota: En la página https://www.onlinehexeditor.com/ hay un editor hexa online.
5)

Este punto es similar al 2, pero ahora no es una palabra que este en el diccionario, es un password completamente aleatorio, que utiliza minúsculas, mayúsculas y números. El problema que tiene esta clave es que solo tiene 5 caracteres de longitud. Tiene el siguiente hash:

f5eaec568d5c415a5829307d1b7c9cdeff9327ea

Debe hacer un ataque de fuerza bruta, que es probar todos las posibles passwords de 5 caracteres, a ver cuál da ese hash. Haga un programa que haga esto.

Debe responder con: el password.

6)
En este punto, solo debe traer le tabla de todos los usuarios con una inyección SQL, que es el ataque más famoso y seguramente todos lo conocen. Esta página es vulnerable a las inyecciones sql más básicas:

http://www.samuelsabogalpardo.com/m1/sqlinject.php

(no tiene que saberse el nombre de ninguna tabla, de la base de datos, ni de nada, es una inyección elemental para solo mostrar los datos)

Debe responder con: El nombre y apellido concatenados, del segundo usuario de la tabla de usuarios.
Ejemplo:
pedrosperez

7)

Ahora, ha llegado el momento de generar un password que probablemente sea usado en la vida real. Piense en lss passwords que utiliza la gente comúnmente, que no parecen tan inseguros, pero sí lo son (una clave segura, tiene que ser completamente aleatoria).

El hash es:

77df9300fdc25ed51e002fdf0c26afc70e46ce69

Pista: Es un palabra de solo letras mayúsculas y números, de longitud 10 (debido a que no es aleatoria, se podría romper aun si tuviera caracteres especiales y minúsculas, pero no con el poder computacional de una de las máquinas que ud está utilizando y en tan poco tiempo como del que se dispone.) Utilice el diccionario words.txt como ayuda.